University of Genova

PhD Program in Bioengineering and Robotics

Hybrid Architectures for Object Pose and Velocity Tracking at the Intersection of Kalman Filtering and Machine Learning

by

Nicola A. Piga

Thesis submitted for the degree of *Doctor of Philosophy* (34th cycle)
March, 2022

Supervisors:
Dr. Lorenzo Natale
Dr. Ugo Pattacini

Thesis Reviewers:
Prof. Alexandre Bernardino, *University of Lisbon, Portugal*
Prof. Ciro Natale, *Università degli Studi della Campania Luigi Vanvitelli, Italy*

Dibris

Department of Informatics, Bioengineering, Robotics and Systems Engineering

Declaration of Authorship

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Nicola A. Piga
March 2022

Abstract

The study of object perception algorithms is fundamental for the development of robotic platforms capable of planning and executing actions involving objects with high precision, reliability and safety. Indeed, this topic has been vastly explored in both the robotic and computer vision research communities using diverse techniques, ranging from classical Bayesian filtering to more modern Machine Learning techniques, and complementary sensing modalities such as vision and touch. Recently, the ever-growing availability of tools for synthetic data generation has substantially increased the adoption of Deep Learning for both 2D tasks, as object detection and segmentation, and 6D tasks, such as object pose estimation and tracking.

The proposed methods exhibit interesting performance on computer vision benchmarks and robotic tasks, e.g. using object pose estimation for grasp planning purposes. Nonetheless, they generally do not consider useful information connected with the physics of the object motion and the peculiarities and requirements of robotic systems. Examples are the necessity to provide well-behaved output signals for robot motion control, the possibility to integrate modelling priors on the motion of the object and algorithmic priors. These help exploit the temporal correlation of the object poses, handle the pose uncertainties and mitigate the effect of outliers. Most of these concepts are considered in classical approaches, e.g. from the Bayesian and Kalman filtering literature, which however are not as powerful as Deep Learning in handling visual data. As a consequence, the development of hybrid architectures that combine the best features from both worlds is particularly appealing in a robotic setting.

Motivated by these considerations, in this Thesis, I aimed at devising hybrid architectures for object perception, focusing on the task of object pose and velocity tracking. The proposed architectures use Kalman filtering supported by state-of-the-art Deep Neural Networks to track the 6D pose and velocity of objects from images. The devised solutions exhibit state-of-the-art performance, increased modularity and do not require training to implement the actual tracking behaviors. Furthermore, they can track even fast object motions despite the possible non-negligible inference times of the adopted neural networks. Also, by relying on data-driven Kalman filtering, I explored a paradigm that enables to track the state of systems that cannot be easily modeled analytically. Specifically, I used this approach to learn the measurement model of soft 3D tactile sensors and address the problem of tracking the sliding motion of hand-held objects.

Publications

The work has been carried out during my PhD in *Advanced an Humanoid Robotics* from November 2018 to November 2021. The three-year project resulted in the following publications:

- A.C. Holgado, N. A. Piga, T. P. Tomo, G. Vezzani, A. Schmitz, L. Natale, S. Sugano, "Magnetic 3-axis Soft and Sensitive Fingertip Sensors Integration for the iCub Humanoid Robot", *IEEE-RAS International Conference on Humanoid Robots*, pp. 328-335, 2019, Toronto
- N. A. Piga, F. Bottarel, C. Fantacci, G. Vezzani, U. Pattacini, L. Natale, "MaskUKF: An Instance Segmentation Aided Unscented Kalman Filter for 6D Object Pose and Velocity Tracking", *Frontiers in Robotics and AI, Humanoid Robotics*, Vol. 8, 2021
- N. A. Piga, U. Pattacini, L. Natale, "A Differentiable Extended Kalman Filter for Object Tracking Under Sliding Regime", *Frontiers in Robotics and AI, Humanoid Robotics, Research Topic "Get in Touch: Technologies for Tactile-Based Robot Control and Human-Robot Cooperation"*, Vol. 8, 2021
- N. A. Piga, Y. Onyshchuk, G. Pasquale, U. Pattacini, L. Natale, "ROFT: Real-Time Optical Flow-Aided 6D Object Pose and Velocity Tracking", *IEEE Robotics and Automation Letters*, pp. 159-166, Vol. 7, No. 1, January 2022

This Thesis provides a structured discussion on top of the aforementioned papers in order to describe the overall contribution of the PhD. As such, some ideas and results have already appeared in these publications.

Acknowledgements

First and foremost, I would like to thank Dr. Lorenzo Natale for giving me the extraordinary opportunity to carry out my PhD project within the Humanoid Sensing and Perception research line. The stimulating and friendly environment allowed me to conduct my research activities successfully and made room for my personal development and growth over these three years.

Special thanks to Dr. Ugo Pattacini for his precious help and countless advice not only on the technical side of my work but especially on how to face the challenges of this research experience.

I am deeply grateful to Claudio and Giulia who supervised me in the initial part of my project and kept in touch with me afterwards. They patiently listened to my ideas and transmitted to me the right attitude while I was making my first steps in Research.

I would like to thank again Claudio, Giulia, Ugo and also Vadim for being such a great team of teachers during the Easy Peasy Robotics initiative, during which my path with IIT started. I will never forget the excitement when I saw for the first time the uncovered head of iCub with its eyeballs following the famous red ball.

Very special thanks to all the friends and colleagues I met during the PhD in the IIT laboratories. Thank you for the innumerable chats, the cheerful moments and for sharing this demanding but at the same time rewarding experience with me.

Thanks to Giulio, long time friend and my housemate, with whom I shared all the difficulties and the achievements along the path. Thank you for always being positive and supportive.

A special mention goes to Lira for her unconditional affection and patience. This long journey would not have been really possible without your outstanding support and presence.

Last but not least, I would like to dedicate this important achievement to my family who gave me the opportunity to pursue my education from the early stages up to the doctoral programme. Thank you for always believing in me and motivating me with plenty of affection and enthusiasm.

Contents

List of figures

List of tables

Chapter 1

Introduction

On a daily basis, we humans interact naturally with surrounding objects to perform several essential tasks. To do so, we implicitly reason on their position in space and coordinate the motion of our limbs to eventually reach, grasp and manipulate them. All these actions involve precise and resolved motions. These are possible because our perceptual system consists of diverse and complementary senses that our brain combines to provide precise, accurate and reliable information about the status of the objects. If one of our senses cannot operate at its best in a given scenario, e.g. our sight when acting in a dark environment, we exploit complementary senses like touch.

Achieving precision, accuracy, reliability[1] and complementarity in robot perception is quite a challenging task.[2] This claim is put forward by the enormous body of literature dedicated to machine and robot perception from both the computer vision and robotics communities. Moreover, the diversity of the offered methodologies makes no doubt on the lack of consensus on the most favourable techniques for the development of modern robots perception skills.

At the same time, an indisputable fact is that Machine Learning, and especially Deep Learning, has reaped large consensus among researchers. Considering that vision is the dominant sense for humans, Deep Learning has paved the way to the development of data-driven techniques that elaborate unstructured data, such as images, and extract very targeted information with results not even foreseeable in the past. Nonetheless, beside the ceaseless development of such data-driven techniques, the risk of loosing

[1]In this context, we define a robot perception algorithm as reliable if it produces a well-behaved output signal without discontinuities and irregularities that would make it unsafe for the robot.

[2]Taking inspiration from humans does not necessarily represent the optimal strategy for the development of robotic perception skills. Nevertheless, precision, accuracy, reliability and complementarity seem to be, with high certainty, desirable properties of robot perception algorithms.

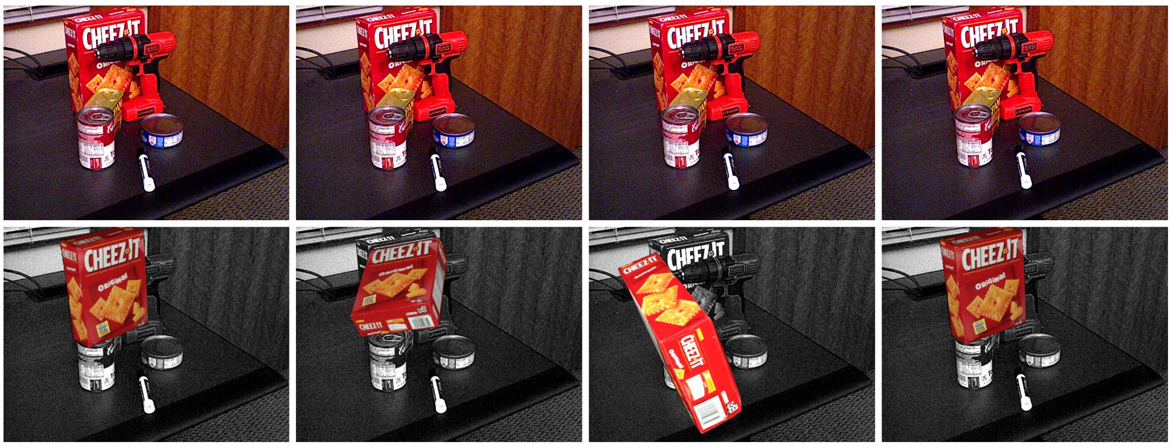


Figure 1.1 In the first row, sample images from a computer vision dataset. In the second row, the output of a learning-based method for object pose estimation, DOPE (Tremblay et al., 2018), superimposed on the input images. The estimated pose of the red box object is discontinuous and not well-behaved. The experiment was carried out by the author of this Thesis.

sight of certain desirable properties of the perception algorithms we develop is not to be overlooked.

A significant case is that of those learning-based methods, known as object pose estimation algorithms, which answer to the problem of *estimating* the pose of objects in the three-dimensional space from a single digital image (Peng et al., 2019; Song et al., 2020; Tremblay et al., 2018; Wang et al., 2019; Xiang et al., 2018). Methods designed to solve the task at hand should exhibit robustness to occlusions, challenging lightning conditions, visual ambiguities and be as less data hungry as possible. On the other hand, less attention is devoted to the requirements of robots that use the output of these methods to plan and execute actions accordingly. Example of such requirements are, but are not limited to, a well-behaved and regular output without discontinuities, a measure of the output uncertainty and principled mechanisms to inject prior information the robot could have at its disposal. Enforcing these latter properties, goes in the direction of developing robots that operate safely and react to unforeseen conditions smoothly.

Remarkably, the aforementioned requirements are among those that so-called *tracking*[3] methods could help achieve. Specifically, tracking the pose of objects may

[3]"At first sight, tracking might seem to be a special case of estimation. However, it is wider in scope: Not only does it use all the tools from estimation, but it also requires extensive use of statistical decision theory when some practical problems are considered." (Bar-Shalom et al., 2002)

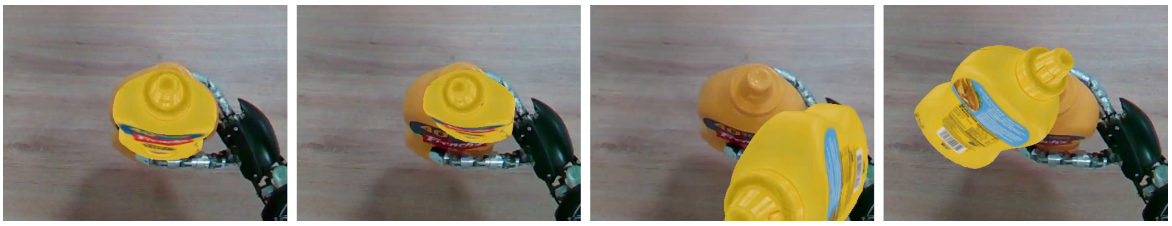


Figure 1.2 An example of robotic manipulation where a learning-based approach, DOPE (Tremblay et al., 2018), is used to estimate the pose of the bottle held in the hand of the robot. The estimated pose, that is superimposed on the input images, is not reliable and is characterized by discontinuities. Exploiting prior information on the interaction with the hand and alternative sensing modalities, e.g. touch, could help make the estimated pose more stable. The experiment was carried out by the author of this Thesis using the iCub humanoid robot platform.

help robots give a solid structure to the data collected from the sensors and account for how they relate with the object pose, their temporal evolution and their uncertainty. Methods that incorporate a tracking framework are usually referred to as object pose *tracking* algorithms. We remark that these methods are inherently different from object pose *estimation* algorithms that process one image at a time and usually discard the temporal evolution of the information.

One of the most adopted algorithms for object pose tracking is the renowned Kalman Filter[4] (Kalman, 1960). Formulated more than 60 years ago, it has been applied to a considerable variety of problems: from the estimation of the trajectories of a manned spacecraft going to the Moon and back, during the Apollo Project (McGee et al., 1985), to the development of the Global Positioning System (GPS) (Brown, 1991), to automotive applications (Rogers, 2003), image processing (Fronckova and Slaby, 2020), finance (Wells, 2013), sensor fusion (Gustafsson, 2010), inverse problems (Evensen, 2009) and robot vision (Chen, 2012). Nowadays, researchers continue to study this algorithm in order to enhance its performance and/or broaden its application fields.

Recently, Kalman filtering has been used to estimate the trainable parameters of neural networks (de Lima et al., 2017). In the opposite direction, Deep Neural Networks (DNNs) have been used to complement Kalman filters leading to *hybrid*

[4]"Filtering is the estimation of the (current) state of a dynamic system - the reason for the use of the word "filter" is that the process for obtaining the "best estimate" from noisy data amount to "filtering out" the noise. The term filtering is thus used in the sense of eliminating an undesired signal, which, in this case, is the noise." (Bar-Shalom et al., 2002)

approaches for tracking the pose of objects using depth and RGB data (Deng et al., 2019; Wong et al., 2017). DNNs are useful in this respect as they allow extracting information from the RGB channels that otherwise would be very difficult to integrate in the filtering algorithm. More in general, neural networks can be employed to learn measurement models for robots sensors that are difficult to model analytically (Kloss et al., 2021; Lee et al., 2020; Revach et al., 2021). On the other hand, the filter offloads the network from tasks that we could avoid learning from data, e.g. the tracking task itself. Nonetheless, we remark that full learning-based approaches for object pose tracking have been proposed as well (Garon and Lalonde, 2017; Ge and Loianno, 2021; Leeb et al., 2019; Wen et al., 2020a).

The coexistence of hybrid approaches for object pose tracking and full learning-based approaches for both object pose estimation *and* tracking emphasizes that we are witnessing a period of transition between the past and the more recent data-driven techniques. At the same time, one could argue that hybrid approaches represent a way to deliberately embrace techniques from the past and take the best from both worlds.

In this work of Thesis, I commit to the latter interpretation and propose several hybrid architectures for tracking the pose and the velocity of objects from RGB and depth images or using three-dimensional tactile sensors. The main contributions of the PhD project are summarized in the Sec. 1.1. Next, Sec. 1.2 outlines the Thesis structure.

1.1 Contribution

The activities carried out during the PhD project were aimed at addressing the following problems:

- The design of hybrid architectures for object pose tracking that try to restrict, as much as possible, the role of neural networks to 2D tasks, that are easier to learn, while leveraging Kalman filtering for the actual task of tracking the 6D pose and velocity of objects (Piga et al., 2021a, 2022).
- The exploitation of real-time external sources of the object velocities, such as the optical flow, to enhance the output of possibly slow and delayed neural networks in order to better handle fast moving objects (Piga et al., 2022).

- The benchmarking and comparison of data-driven approaches for 6D object pose estimation and tracking with hybrid approaches for object pose tracking (Piga et al., 2021a, 2022).
- The integration of robots sensors that are difficult to model mathematically, such as soft three-dimensional tactile sensors, within in-hand object pose tracking methods (Piga et al., 2021b).

Addressing the aforementioned problems resulted in the development of several methods for tracking the pose and the velocity of objects that represent the actual contribution of this Thesis:

- A method, named *MaskUKF* (Piga et al., 2021a), which combines Deep Neural Networks, for object segmentation, and depth information within a Kalman filtering framework and that does not require a separate training for each object of interest.
- A method, named *ROFT: Real-time Optical Flow-aided 6D object pose and velocity Tracking* (Piga et al., 2022), which combines real-time optical flow and depth, slow and delayed Deep Neural Networks for object segmentation and pose estimation from RGB images within a Kalman filtering framework, that is capable of handling fast moving objects.
- A method embracing the recent *Differentiable Kalman Filtering paradigm*[5] that tracks in-hand translational sliding motions of objects from tactile feedback only (Piga et al., 2021b).

1.2 Outline

The Thesis is organized as follows.

A thorough description of the Kalman Filter algorithm and its variants is provided in Chapter 2.

The Chapter 3 introduces the concepts of object pose and velocity and provides a formal and general definition of the tasks of tracking the pose and the velocity of an object. It also highlights the differences between the object pose estimation and

[5]Differentiable Kalman filters maintain their algorithmic inner structure while allowing dynamics and measurement models to be learned from data in a supervised manner.

tracking tasks. The Chapter ends with a detailed overview of classical, data-driven and hybrid state-of-the-art methods.

In Chapter 4, I present the MaskUKF algorithm, including the mathematical formulation and experimental evaluations. In the same Chapter I present the results of a simulated robot motion control task, with the iCub humanoid robot, where MaskUKF is compared with a data-driven pose estimation method and a baseline. The contents of this Chapter represent an extension of the work presented in (Piga et al., 2021a).

In Chapter 5, I describe the optical flow-aided pipeline ROFT. The mathematical derivation is followed by experimental evaluations, a comparison with data-driven pose estimation and tracking methods and several ablation studies aimed at revealing the modularity of the proposed pipeline. The contents of this Chapter are associated with the work presented in (Piga et al., 2022).

In Chapter 6, I present the data collection and training activities that led to the development of a differentiable Kalman filter which tracks in-hand translational sliding motions of objects using tactile feedback. In particular, the Chapter describes the setup involving the iCub humanoid robot and its tactile sensors and present the experimental results. The contents of this Chapter are associated with the work presented in (Piga et al., 2021b).

Finally, Chapter 7 ends the Thesis work with additional remarks and possible future research directions.

Chapter 2

Background

In this section I will cover the theory of Kalman filtering (Kalman, 1960) for state tracking and describe the most common variants of the main algorithm that have been used in this Thesis. In Sec. 2.1, I will present the basics of Kalman filtering, in Sec. 2.2 its extensions to high dimensional measurements and, in Sec. 2.3, to orientation tracking. Next, in Sec. 2.4, I will consider the theory of Kalman Smoothing. Finally, in Sec. 2.5, I will present the theory of differentiable Kalman filters.

2.1 Kalman filtering for linear and nonlinear systems

This section presents the basics of Bayesian filtering (Särkkä, 2013) and explains how it can be used to derive the equations of the Kalman filter algorithm (Kalman, 1960; Särkkä, 2013) and its most common extensions known as the Extended Kalman filter (EKF) (McGee et al., 1985; Smith et al., 1962; Särkkä, 2013) and the Unscented Kalman filter (UKF) (Särkkä, 2013; Wan and Merwe, 2000).

The adopted notation and the mathematical derivations presented in this Section are mostly adapted from (Särkkä, 2013).

2.1.1 Basics of Bayesian filtering

The Kalman filter belongs to the family of recursive Bayesian filtering algorithms. These algorithms estimate key probability distributions functions of the state x_t of a given system at time t starting from collections of noisy measurements $z_{1:T} = \{z_1, \dots, z_T\}$ with $T \lesseqqgtr t$. The measurements represent an indirect indication of the state x and are usually obtained by means of sensors of different nature. In the following, it is assumed that both the state and measurements are represented using Euclidean vector, i.e. $x_t \in \mathbb{R}^n$ and $z_t \in \mathbb{R}^m$ with n and m the dimensions of the state and measurement vectors, respectively.

Among the possible distribution functions, it is of interest to estimate the *filtering distribution function* of the state x_t given the current and previous measurements $z_{1:t} = \{z_1, ..., z_t\}$:

$$p(x_t|z_{1:t}). \tag{2.1}$$

This distribution can be evaluated recursively by resorting to the Bayes rule as follows:

$$p(x_t|z_{1:t}) \propto p(z_t|x_t)p(x_t|z_{1:t-1}), \tag{2.2}$$

where

- $p(z_t|x_t)$ is the *measurement likelihood* that measures the plausibility of the measurement z_t given the knowledge of the state x_t;
- $p(x_t|z_{1:t-1})$ is the *predictive distribution function* of the state x_t.

The predictive distribution describes how the state should evolve at time t according to a *state transition probability distribution* $p(x_t|x_{t-1})$ and taking into account the filtering distribution at time $t-1$, $p(x_{t-1}|z_{1:t-1})$. The current measurement z_t does not contribute to the predictive distribution. The predictive distribution can be computed using the Chapman-Kolmogorov equation:

$$\begin{aligned} p(x_t|z_{1:t-1}) &= \int p(x_t, x_{t-1}|z_{1:t-1})\mathrm{d}x_{t-1} \\ &= \int p(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1})\mathrm{d}x_{t-1}. \end{aligned} \tag{2.3}$$

The recursive update rules of both the predictive and filtering distributions are valid under the assumption that the states are a Markov sequence, i.e.

$$p(x_t|x_{1:t-1}, z_{1:t-1}) = p(x_t|x_{t-1}), \tag{2.4}$$

and that the measurements are conditionally independent given the states, i.e.

$$p(z_t|x_{1:t}, z_{1:t-1}) = p(z_t|x_t). \tag{2.5}$$

In general, the integral in Eq. (2.3) cannot be solved in closed form and its result depends heavily on the specific nature of the probability distributions at hand. Consequently, the filtering distribution in Eq. (2.2) cannot be evaluated in closed form.

2.1.2 Basics of Kalman filtering

In the Kalman filtering setting, the filtering distribution $p(x_t|z_{1:t})$ is approximated using a Gaussian distribution

$$p(x_t|z_{1:t}) \simeq \mathcal{N}(x;\mu_t,P_t)\bigg|_{x=x_t}, \tag{2.6}$$

where μ_t and P_t are the mean vector and the covariance matrix that characterize the distribution. The approximation is possible if the following assumptions are made:

- the state x_t evolves according to a *discrete difference equation*

$$\begin{aligned} x_t &= f_t(x_{t-1}, w_{t-1}), \\ w_t &\sim \mathcal{N}(0, Q_t), \end{aligned} \tag{2.7}$$

  with f_t a generic nonlinear function and w_t a sequence of zero mean Gaussian noise samples with covariance Q_t and independent from the states;

- the measurements z_t can be modelled once the state x_t is given as

$$\begin{aligned} z_t &= h_t(x_t, \nu_t), \\ \nu_t &\sim \mathcal{N}(0, R_t), \end{aligned} \tag{2.8}$$

  with h_t a generic nonlinear function and ν_t a sequence of zero mean Gaussian noise samples with covariance R_t and independent from the states;

- the joint probability distribution of consecutive states conditioned to the history of the measurements $p(x_{t-1}, x_t|z_{1:t-1})$ is Gaussian;

- the joint probability distribution of the current state and the current measurement conditioned to the history of the measurements $p(x_t, z_t|z_{1:t-1})$ is Gaussian;

- the prior $p(x_0)$ is Gaussian with mean μ_0 and covariance P_0.

The mathematical models in Eqs. (2.7) and (2.8) are called the *state model* and the *measurement model* respectively.

Under the aforementioned assumptions, it is possible to prove (Särkkä, 2013) that there exist closed form approximate solutions to the Bayesian recursive update rules of the predictive and filtering distributions. Using the Kalman filtering terminology, these rules are called the Kalman prediction and correction steps, respectively. The mathematical form of the updates is reported in the remainder of this chapter for several alternatives of the state and measurement models in Eqs. (2.7) and (2.8).

We will make the assumption that the filtering distribution at time $t-1$ is Gaussian in order to prove that the filtering distribution at time t is actually Gaussian in a Kalman filtering setting. This assumption is sound, given the recursive nature of the Bayesian update rules, and resembles the inductive step of the mathematical proof technique known as "mathematical induction". The base step, that is that the filtering distribution at time $t = 0$ is Gaussian, corresponds to one of the hypotheses. Indeed, the prior $p(x_0)$ corresponds to the filtering distribution $p(x_0|y_{1:0})$ at time $t = 0$ where $y_{1:0}$ is an empty set of measurements.

2.1.3 Linear Kalman filtering

Linear Kalman filters assume that the functions f_t and h_t are linear functions of the state x_t represented by the matrices F_t and H_t respectively. Furthermore, the contribution of the noise samples w_t and ν_t is additive. The resulting linear models are as follows:

$$\begin{aligned} x_t &= F_t x_{t-1} + w_{t-1}, \\ w_t &\sim \mathcal{N}(0, Q_t); \end{aligned} \tag{2.9}$$

$$\begin{aligned} z_t &= H_t x_t + \nu_t, \\ \nu_t &\sim \mathcal{N}(0, R_t). \end{aligned} \tag{2.10}$$

In this configuration, the joint probabilities $p(x_{t-1}, x_t|z_{1:t-1})$ and $p(x_t, z_t|z_{1:t-1})$ are Gaussian distributions by construction as shown in the following.

Prediction step

Using the definition of conditional probability, it is possible to rewrite $p(x_{t-1}, x_t|z_{1:t-1})$ as

$$\begin{aligned} p(x_{t-1}, x_t|z_{1:t-1}) &= p(x_t|x_{t-1}, z_{1:t-1})p(x_{t-1}|z_{1:t-1}) \\ &= p(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1}), \end{aligned} \tag{2.11}$$

where we used the assumption that the states are a Markov sequence. Considering the state model in Eq. (2.9), it is evident that the transition probability distribution $p(x_t|x_{t-1})$ is Gaussian and evaluates to

$$p(x_t|x_{t-1}) = \mathcal{N}(x; F_t x_{t-1}, Q_{t-1})\Big|_{x=x_t}. \tag{2.12}$$

The right-hand side factor in (2.11) corresponds to the filtering distribution at time $t-1$ that is a Gaussian distribution by hypothesis. The resulting joint distribution is Gaussian, as required, as it is the product of two Gaussian distributions:

$$\begin{aligned} p(x_{t-1}, x_t|z_{1:t-1}) &= \mathcal{N}\left(\begin{bmatrix} x_{t-1} \\ x_t \end{bmatrix}; \mu_t', P_t'\right), \\ \mu_t' &= \begin{bmatrix} \mu_{t-1} \\ F_t\mu_{t-1} \end{bmatrix}, \\ P_t' &= \begin{bmatrix} P_{t-1} & P_{t-1}F_t^T \\ F_tP_{t-1} & F_tP_{t-1}F_t^T + Q_{t-1} \end{bmatrix}, \end{aligned} \tag{2.13}$$

where μ_{t-1} and P_{t-1} are the mean and the covariance of the Gaussian filtering distribution $p(x_{t-1}|z_{1:t-1})$ at time $t-1$.

Starting from the joint distribution in Eq. (2.13), it can be shown (Särkkä, 2013) that the predictive distribution $p(x_t|z_{1:t-1})$ is a Gaussian distribution

$$\begin{aligned} p(x_t|z_{1:t-1}) &= \mathcal{N}(x; \mu_t^-, P_t^-)\Big|_{x=x_t}, \\ \mu_t^- &= F_t\mu_{t-1}, \\ P_t^- &= F_tP_{t-1}F_t^T + Q_{t-1}. \end{aligned} \tag{2.14}$$

This step is also called *Kalman prediction step.*

Correction step

To derive the correction step, we use again the definition of conditional probability and rewrite the joint probability $p(x_t, z_t|z_{1:t-1})$ as follows:

$$\begin{aligned} p(x_t, z_t|z_{1:t-1}) &= p(z_t|x_t, z_{1:t-1})p(x_t|z_{1:t-1}) \\ &= p(z_t|x_t)\mathcal{N}(x; \mu_t^-, P_t^-). \end{aligned} \tag{2.15}$$

Here, we made the assumption that the measurements are conditionally independent given the states. Given the definition of the measurement model in (2.10), we find that the measurement likelihood $p(z_t|x_t)$ is Gaussian and evaluates to

$$p(z_t|x_t) = \mathcal{N}(z; H_t x_t, R_t)\Big|_{z=z_t}. \tag{2.16}$$

As a result, the joint distribution is Gaussian, as required, because it is the product of two Gaussian distributions:

$$\begin{aligned} p(x_t, z_t|z_{1:t-1}) &= \mathcal{N}\left(\begin{bmatrix} x_t \\ z_t \end{bmatrix}; \mu_t'', P_t''\right), \\ \mu_t'' &= \begin{bmatrix} \mu_t^- \\ H_t\mu_t^- \end{bmatrix}, \\ P_t'' &= \begin{bmatrix} P_t^- & P_t^- H_t^T \\ H_t P_t^- & H_t P_t^- H_t^T + R_t \end{bmatrix}. \end{aligned} \tag{2.17}$$

Taking into account (2.17), we can show (Särkkä, 2013) that that the filtering distribution $p(x_t|z_{1:t})$ is a Gaussian distribution

$$\begin{aligned} p(x_t|z_{1:t}) &= \mathcal{N}(x; \mu_t, P_t)\Big|_{x=x_t}, \\ \mu_t &= \mu_t^- + K_t e_t, \\ P_t &= P_t^- - K_t P_{z,t} K_t^T, \\ e_t &= z_t - H_t\mu_t^-, \\ P_{z,t} &= H_t P_t^- H_t^T + R_t, \\ K_t &= P_t^- H_t^T P_{z,t}^{-1}, \end{aligned} \tag{2.18}$$

where e_t is called the *Kalman innovation*, $P_{z,t}$ is the predicted covariance of the measurement and K_t is called the *Kalman gain*. This step is also called *Kalman correction* step. The vector $H_t\mu_t^-$ and the covariance matrix $P_{z,t}$ have a clear meaning in probabilistic terms as they allow defining a predictive distribution $p(z_t|z_{1:t-1})$ in the measurement space:

$$p(z_t|z_{1:t-1}) = \mathcal{N}(z; H_t\mu_t^-, P_{z,t})\Big|_{z=z_t}. \tag{2.19}$$

We notice that the solution provided by the Linear Kalman filter is not approximate but rather exact, i.e. the resulting distributions are actually Gaussian.

The resulting algorithm

Algorithm 1. *Kalman filter*

Given a system described by the state model in Eq. (2.9) *and the measurement model in Eq.* (2.10) *and an initial prior*

$$p(x_0) \sim \mathcal{N}(\mu_0, P_0),$$

the predictive and filtering distributions of the state are Gaussian and can be computed with the following Kalman filter prediction and correction steps:

- *Prediction step:*

$$\begin{aligned} p(x_t|z_{1:t-1}) &= \mathcal{N}(x; \mu_t^-, P_t^-)\Big|_{x=x_t}, \\ \mu_t^- &= F_t\mu_{t-1}, \\ P_t^- &= F_tP_{t-1}F_t^T + Q_{t-1}. \end{aligned} \tag{2.20}$$

- *Correction step:*

$$\begin{aligned} p(x_t|z_{1:t}) &= \mathcal{N}(x; \mu_t, P_t)\Big|_{x=x_t}, \\ \mu_t &= \mu_t^- + K_te_t, \\ P_t &= P_t^- - K_tP_{z,t}K_t^T, \\ e_t &= z_t - H_t\mu_t^-, \\ P_{z,t} &= H_tP_t^-H_t^T + R_t, \\ K_t &= P_t^-H_t^TP_{z,t}^{-1}. \end{aligned} \tag{2.21}$$

2.1.4 Extended Kalman filtering

If the functions f_t and h_t in the models in Eqs. (2.7) and (2.8) are nonlinear, the joint distribution $p(x_{t-1}, x_t | z_{1:t-1})$ of x_{t-1} and x_t and the joint distribution $p(x_t, z_t | z_{1:t-1})$ of x_t and z_t are non-Gaussian in general, hence the hypotheses indicated in Sec. 2.1.2 are not valid anymore. In this configuration, it is not possible to use the standard Linear Kalman filter as presented in the previous section.

An alternative formulation of the Kalman filter, known as the Extended Kalman filter (EKF) (McGee et al., 1985; Smith et al., 1962; Särkkä, 2013), can be used in this case. The EKF relies on the Gaussian approximation of the joint distributions of interest made possible by the following algorithm (Särkkä, 2013).

Algorithm 2. *Linear approximation of a non-additive transform*

The linear Gaussian approximation of the distribution of the variable

$$\begin{bmatrix} x \\ y \end{bmatrix}, \tag{2.22}$$

where $y = g(x, q)$, $x \sim \mathcal{N}(\mu_x, P_x)$, $q \sim \mathcal{N}(0, P_q)$, x *and* q *are independent and* g *is a nonlinear and differentiable function is given as*

$$\begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} P_x & P_{xy} \\ P_{xy}^T & P_y \end{bmatrix} \right), \tag{2.23}$$

where

$$\begin{aligned} \mu_y &= g(\mu_x, 0), \\ P_y &= J_x(\mu_x) P_x J_x(\mu_x)^T + J_q(\mu_x) P_q J_q(\mu_x)^T, \\ P_{xy} &= P_x J_x(\mu_x)^T, \\ J_x(\mu) &= \left. \frac{\partial g(x, q)}{\partial x} \right|_{x=\mu, q=0}, \\ J_q(\mu) &= \left. \frac{\partial g(x, q)}{\partial q} \right|_{x=\mu, q=0}. \end{aligned} \tag{2.24}$$

The approximation takes advantage of the differentiability of the function g in order evaluate the Jacobians J_x and J_q. These Jacobians are those required to define a Taylor expansion of the first-order, or *linear* expansion, of the function g around the mean $(\mu_x, 0)$:

$$y = g(x,q) \simeq g(\mu_x, 0) + J_x(\mu_x)(x - \mu_x) + J_q(\mu_x)(q - 0). \tag{2.25}$$

For this reason, the approximation is termed linear.

Prediction step

Application of the Algorithm 2 to the state model in Eq. (2.7) allows evaluating a Gaussian approximation of the joint distribution $p(x_{t-1}, x_t|z_{1:t-1})$ as follows:

$$\begin{aligned} p(x_{t-1}, x_t|z_{1:t-1}) &\simeq \mathcal{N}\left(\begin{bmatrix} x_{t-1} \\ x_t \end{bmatrix}; \mu_t^{'}, P_t^{'}\right), \\ \mu_t^{'} &= \begin{bmatrix} \mu_{t-1} \\ f_t(\mu_{t-1}, 0) \end{bmatrix}, \\ P_t^{'} &= \begin{bmatrix} P_{t-1} & P_{t-1}F_{x,t}^T \\ F_{x,t}P_{t-1} & F_{x,t}P_{t-1}F_{x,t}^T + F_{w,t}Q_{t-1}F_{w,t}^T \end{bmatrix}, \end{aligned} \tag{2.26}$$

where μ_{t-1} and P_{t-1} are the mean and the covariance of the Gaussian filtering distribution $p(x_{t-1}|z_{1:t-1})$ at time $t-1$, and $F_{x,t}$ and $F_{w,t}$ are the Jacobians $J_x(\mu_{t-1})$ and $J_w(\mu_{t-1})$ evaluated using $g = f_t$.

Once a Gaussian approximation of the joint distribution in Eq. (2.26) is available, it is possible to evaluate the predictive distribution as done for the Linear Kalman filter as follows (Särkkä, 2013):

$$\begin{aligned} p(x_t|z_{1:t-1}) &\simeq \mathcal{N}(x; \mu_t^-, P_t^-)\Big|_{x=x_t}, \\ \mu_t^- &= f_t(\mu_{t-1}, 0), \\ P_t^- &= F_{x,t}P_{t-1}F_{x,t}^T + F_{w,t}Q_{t-1}F_{w,t}^T. \end{aligned} \tag{2.27}$$

This step is also called the *Extended Kalman prediction step.*

A comparison with the predictive distribution in the linear setting, as per Eq. (2.14), shows that the equivalent distribution in the nonlinear setting is obtained by substituting the state transition matrix F_t with the Jacobian $F_{x,t}$, the product $F_t\mu_{t-1}$ with $f_t(\mu_{t-1}, 0)$ and by weighting the noise covariance Q_t with the Jacobian $F_{w,t}$.

Correction step

The same reasoning can be applied to the measurement model in Eq. (2.8) resulting in the following Gaussian approximation of the joint distribution

$$\begin{aligned} p(x_t, z_t|z_{1:t-1}) &\simeq \mathcal{N}\left(\begin{bmatrix} x_t \\ z_t \end{bmatrix}; \mu_t'', P_t''\right), \\ \mu_t'' &= \begin{bmatrix} \mu_t^- \\ h_t(\mu_t^-, 0) \end{bmatrix}, \\ P_t'' &= \begin{bmatrix} P_t^- & P_t^- H_{x,t}^T \\ H_{x,t}P_t^- & H_{x,t}P_t^- H_{x,t}^T + H_{\nu,t}R_t H_{\nu,t}^T \end{bmatrix}, \end{aligned} \tag{2.28}$$

and filtering distribution

$$\begin{aligned} p(x_t|z_{1:t}) &\simeq \mathcal{N}(x; \mu_t, P_t)\Big|_{x=x_t}, \\ \mu_t &= \mu_t^- + K_t e_t, \\ P_t &= P_t^- - K_t P_{z,t} K_t^T, \\ e_t &= z_t - h_t(\mu_t^-, 0), \\ P_{z,t} &= H_{x,t}P_t^- H_{x,t}^T + H_{\nu,t}R_t H_{\nu,t}^T, \\ K_t &= P_t^- H_{x,t}^T P_{z,t}^{-1}, \end{aligned} \tag{2.29}$$

where $H_{x,t}$ and $H_{\nu,t}$ are the Jacobians $J_x(\mu_t^-)$ and $J_\nu(\mu_t^-)$ evaluated using $g = h_t$. This step is also called the *Extended Kalman correction step.*

Also in this case, the filtering distribution can be obtained starting from the same distribution in the linear setting, as per Eq. (2.18), by substituting the measurement matrix H_t with the Jacobian $H_{x,t}$, the product $H_t\mu_t^-$ with $h_t(\mu_t^-, 0)$ and by weighting the noise covariance R_t with the Jacobian $H_{\nu,t}$.

The resulting algorithm

Algorithm 3. *Extended Kalman filter*

Given a system described by the state model in Eq. (2.7) *and the measurement model in Eq.* (2.8), *where the functions f_t and h_t are nonlinear and differentiable, and an initial prior*

$$p(x_0) \sim \mathcal{N}(\mu_0, P_0),$$

the predictive and filtering distributions of the state are approximately Gaussian and can be computed with the following Extended Kalman filter prediction and correction steps:

- *Prediction step:*

$$\begin{aligned} p(x_t|z_{1:t-1}) &\simeq \mathcal{N}(x; \mu_t^-, P_t^-)\Big|_{x=x_t}, \\ \mu_t^- &= f_t(\mu_{t-1}, 0), \\ P_t^- &= F_{x,t} P_{t-1} F_{x,t}^T + F_{w,t} Q_{t-1} F_{w,t}^T, \end{aligned} \tag{2.30}$$

where

$$\begin{aligned} F_{x.t} &= \frac{\partial f_t(x,w)}{\partial x}\Big|_{x=\mu_{t-1}, w=0}, \\ F_{w,t} &= \frac{\partial f_t(x,w)}{\partial w}\Big|_{x=\mu_{t-1}, w=0}. \end{aligned} \tag{2.31}$$

- *Correction step:*

$$\begin{aligned} p(x_t|z_{1:t}) &\simeq \mathcal{N}(x; \mu_t, P_t)\Big|_{x=x_t}, \\ \mu_t &= \mu_t^- + K_t e_t, \\ P_t &= P_t^- - K_t P_{z,t} K_t^T, \\ e_t &= z_t - h_t(\mu_t^-, 0), \\ P_{z,t} &= H_{x,t} P_t^- H_{x,t}^T + H_{\nu,t} R_t H_{\nu,t}^T, \\ K_t &= P_t^- H_{x,t}^T P_{z,t}^{-1}, \end{aligned} \tag{2.32}$$

where

$$\begin{aligned} H_{x.t} &= \frac{\partial h_t(x,\nu)}{\partial x}\Big|_{x=\mu_t^-, \nu=0}, \\ H_{w,t} &= \frac{\partial h_t(x,\nu)}{\partial \nu}\Big|_{x=\mu_t^-, \nu=0}. \end{aligned} \tag{2.33}$$

2.1.5 Unscented Kalman filtering

The Extended Kalman filter allows applying the Kalman filtering methodology to any nonlinear system in the form given by Eqs. (2.7) and (2.8) under the assumption that the functions f_t and g_t can be approximated using a first-order linear Taylor expansion.

An alternative approach, known as the Unscented Kalman filter (UKF) (Särkkä, 2013; Wan and Merwe, 2000), tries to extend the Kalman filtering methodology to nonlinear functions using a different philosophy. Instead of approximating the nonlinear functions, used in the state and measurement models, it tries to directly approximate the mean and the covariance of the joint distributions $p(x_{t-1}, x_t|z_{1:t-1})$ and $p(x_t, z_t|z_{1:t-1})$. Similarly to the EKF, this is made possible by a suitable algorithm for the Gaussian approximation of a non-additive transform. The algorithm is the following (Särkkä, 2013).

Algorithm 4. *Unscented approximation of a non-additive transform*

The Unscented Gaussian approximation of the distribution of the variable

$$\begin{bmatrix} x \\ y \end{bmatrix}, \tag{2.34}$$

where $y = g(x, q)$, $x \sim \mathcal{N}(\mu_x, P_x)$, $q \sim \mathcal{N}(0, P_q)$, x and q are independent and g is a nonlinear function is given as

$$\begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} P_x & P_{xy} \\ P_{xy}^T & P_y \end{bmatrix}\right), \tag{2.35}$$

where the sub-vectors and sub-matrices can be computed as follows. Let the dimensionality of x and q be n and n_q, respectively, and let $\tilde{n} = n + n_q$.

1. *Form a set of $2\tilde{n} + 1$ points, called sigma points:*

$$\begin{aligned} \mathcal{X}^{(0)} &= \tilde{\mu}, \\ \mathcal{X}^{(i)} &= \tilde{\mu} + \sqrt{\tilde{n} + \lambda}\left[\sqrt{\tilde{P}}\right]_i, \\ \mathcal{X}^{(i+n)} &= \tilde{\mu} - \sqrt{\tilde{n} + \lambda}\left[\sqrt{\tilde{P}}\right]_i, \quad i = 1, \ldots, \tilde{n}, \\ \tilde{\mu} &= \begin{bmatrix} \mu_x \\ 0 \end{bmatrix}, \quad \tilde{P} = \begin{bmatrix} P_x & 0 \\ 0 & P_q \end{bmatrix}, \end{aligned} \tag{2.36}$$

where λ is a scaling parameter, later defined, the operator $[\cdot]_i$ indicates the i-th column of its argument and the matrix square root $\sqrt{\tilde{P}}$ denotes a matrix such that $\sqrt{\tilde{P}}\sqrt{\tilde{P}}^T = \tilde{P}$.

2. *Propagate the sigma points through the function g:*

$$\mathcal{Y}^{(i)} = g(\mathcal{X}^{(i),x}, \mathcal{X}^{(i),q}), \quad i = 0, \dots, 2\tilde{n}, \tag{2.37}$$

where $\mathcal{X}^{(i),x}$, $\mathcal{X}^{(i),q}$ denote the first n and n_q components of $\mathcal{X}^{(i)}$ respectively.

3. *Computer the sub-vectors and sub-matrices:*

$$\begin{aligned}
\mu_y &= \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(m)} \mathcal{Y}^{(i)}, \\
P_y &= \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(c)} \left(\mathcal{Y}^{(i)} - \mu_y\right) \left(\mathcal{Y}^{(i)} - \mu_y\right)^T, \\
P_{xy} &= \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(c)} \left(\mathcal{X}^{(i),x} - \mu_x\right) \left(\mathcal{Y}^{(i)} - \mu_y\right)^T,
\end{aligned} \tag{2.38}$$

where $\mathcal{W}_i^{(m)}$ and $\mathcal{W}_i^{(c)}$ are suitable weights, later defined.

The scaling parameter λ is defined as follows (Wan and Merwe, 2000):

$$\lambda = \alpha^2(\tilde{n} + \kappa) - \tilde{n}. \tag{2.39}$$

The parameters α and κ determine how the sigma points spread out around the mean $\tilde{\mu}$. The weights $\mathcal{W}_i^{(m)}$ and $\mathcal{W}_i^{(c)}$ are given as follows (Wan and Merwe, 2000):

$$\begin{aligned}
\mathcal{W}_0^{(m)} &= \frac{\lambda}{\tilde{n} + \lambda}, \\
\mathcal{W}_0^{(c)} &= \frac{\lambda}{\tilde{n} + \lambda} + (1 - \alpha^2 + \beta), \\
\mathcal{W}_i^{(m)} = \mathcal{W}_i^{(c)} &= \frac{1}{2(\tilde{n} + \lambda)}, \quad i = 1, \dots, 2\tilde{n}.
\end{aligned} \tag{2.40}$$

The parameter β is an additional parameter that can be used to incorporate prior knowledge of the distribution of x (Wan and Merwe, 2000).

The advantage of the Algorithm 4 over the Algorithm 2, used in the EKF, is that it is not based on a linear approximation at a single point and instead uses an ensemble of points, the sigma points, in order to approximate the effect of the function g on the input variable x. Furthermore, the function g is not required to be differentiable and its Jacobian with respect the inputs is not required. On the other hand, one disadvantage over the Algorithm 2 is that it requires a slightly increased computational effort. E.g. the Algorithm 2 requires evaluating the function g once in the mean $(\mu_x, 0)$ while the Algorithm 4 requires evaluating the function g on all the $2\tilde{n} + 1$ sigma points.

Prediction and correction steps

Application of the Algorithm 4 to the state and measurement models given by Eqs. (2.7) and (2.8) allows obtaining an Unscented Gaussian approximation of the joint distributions $p(x_{t-1}, x_t | z_{1:t-1})$ and $p(x_t, z_t | z_{1:t-1})$, respectively. Consequently, the predictive and filtering distribution can be evaluated in an analogous manner to the Linear and Extended Kalman filters.

The resulting algorithm

Algorithm 5. *Unscented Kalman filter*

Given a system described by the state model in Eq. (2.7) *and the measurement model in Eq.* (2.8)*, where the functions f_t and h_t are nonlinear, and an initial prior*

$$p(x_0) \sim \mathcal{N}(\mu_0, P_0),$$

the predictive and filtering distributions of the state are approximately Gaussian and can be computed with the following Uscented Kalman filter prediction and correction steps:

- *Prediction step:*

1. *Form the sigma points for the augmented variable* (x_{t-1}, w_{t-1})*:*

$$\begin{aligned}\mathcal{X}_{t-1}^{(0)} &= \tilde{\mu}_{t-1},\\ \mathcal{X}_{t-1}^{(i)} &= \tilde{\mu}_{t-1} + \sqrt{\tilde{n}+\lambda}\left[\sqrt{\tilde{P}_{t-1}}\right]_i,\\ \mathcal{X}_{t-1}^{(i+n)} &= \tilde{\mu}_{t-1} - \sqrt{\tilde{n}+\lambda}\left[\sqrt{\tilde{P}_{t-1}}\right]_i, \quad i = 1,\dots,\tilde{n},\\ \tilde{\mu}_{t-1} &= \begin{bmatrix}\mu_{t-1}\\ 0\end{bmatrix}, \quad \tilde{P}_{t-1} = \begin{bmatrix}P_{t-1} & 0\\ 0 & Q_{t-1}\end{bmatrix},\end{aligned} \tag{2.41}$$

where $\tilde{n}$ *is the size of the augmented variable and* λ *is defined as in Eq.* (2.39)*.*

2. *Propagate the sigma points through the function* f_t *in Eq.* (2.7)*:*

$$\mathcal{X}_t^{(i)} = f_t(\mathcal{X}_{t-1}^{(i),x}, \mathcal{X}_{t-1}^{(i),w}), \quad i = 0,\dots,2\tilde{n}, \tag{2.42}$$

where $\mathcal{X}_{t-1}^{(i),x}$ *and* $\mathcal{X}_{t-1}^{(i),w}$ *denote the part of the sigma point associated to* x *and* w *respectively.*

3. *Compute the approximation of the predictive distribution:*

$$\begin{aligned}p(x_t|z_{1:t-1}) &\simeq \mathcal{N}(x;\mu_t^-,P_t^-)\Big|_{x=x_t},\\ \mu_t^- &= \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(m)}\mathcal{X}_t^{(i)},\\ P_t^- &= \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(c)}\left(\mathcal{X}_t^{(i)}-\mu_t^-\right)\left(\mathcal{X}_t^{(i)}-\mu_t^-\right)^T,\end{aligned} \tag{2.43}$$

where the weights $\mathcal{W}_i^{(m)}$ *and* $\mathcal{W}_i^{(c)}$ *are defined as in Eq.* (2.40)*.*

- *Correction step:*

1. *Form the sigma points for the augmented variable* (x_t, ν_t)*:*

$$
\begin{aligned}
\mathcal{X}_t^{-(0)} &= \tilde{\mu}_t^-, \\
\mathcal{X}_t^{-(i)} &= \tilde{\mu}_t^- + \sqrt{\tilde{n}+\lambda}\left[\sqrt{\tilde{P}_t^-}\right]_i, \\
\mathcal{X}_t^{-(i+n)} &= \tilde{\mu}_t^- - \sqrt{\tilde{n}+\lambda}\left[\sqrt{\tilde{P}_t^-}\right]_i, \quad i = 1, \dots, \tilde{n}, \\
\tilde{\mu}_t &= \begin{bmatrix} \mu_t^- \\ 0 \end{bmatrix}, \quad \tilde{P}_t^- = \begin{bmatrix} P_t^- & 0 \\ 0 & R_t \end{bmatrix}.
\end{aligned} \tag{2.44}
$$

2. *Propagate the sigma points through the function* h_t *in Eq.* (2.8)*:*

$$
\mathcal{Z}_t^{(i)} = h_t(\mathcal{X}_t^{-(i),x}, \mathcal{X}_t^{-(i),\nu}), \quad i = 0, \dots, 2\tilde{n}, \tag{2.45}
$$

where $\mathcal{X}_t^{-(i),x}$ *and* $\mathcal{X}_t^{-(i),\nu}$ *denote the part of the sigma point associated to* x *and* ν *respectively.*

3. *Compute the approximation of the filtering distribution:*

$$
\begin{aligned}
p(x_t|z_{1:t}) &\simeq \mathcal{N}(x; \mu_t, P_t)\Big|_{x=x_t}, \\
\mu_t &= \mu_t^- + K_t e_t, \\
P_t &= P_t^- - K_t P_{z,t} K_t^T, \\
e_t &= z_t - \mu_{z,t}, \\
\mu_{z,t} &= \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(m)} \mathcal{Z}_t^{(i)}, \\
P_{z,t} &= \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(c)} \left(\mathcal{Z}_t^{(i)} - \mu_{z,t}\right)\left(\mathcal{Z}_t^{(i)} - \mu_{z,t}\right)^T, \\
P_{xz,t} &= \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(c)} \left(\mathcal{X}_t^{-(i),x} - \mu_t^-\right)\left(\mathcal{Z}_t^{(i)} - \mu_{z,t}\right)^T, \\
K_t &= P_{xz,t} P_{z,t}^{-1}.
\end{aligned} \tag{2.46}
$$

2.2 Sequential Kalman filtering for high-dimensional measurements

Algorithms for object perception use images acquired from a camera as input. A considerable number of these methods process the information stored in the image, e.g. intensity, depth, at the pixel level. In a Kalman filtering setting, this choice implies that the size of vector of the measurements z_t can be considerable, in the order of thousands or more. Measurement vectors with this characteristic, termed high-dimensional, make difficult to implement Kalman filtering algorithms efficiently (McManus and Barfoot, 2011; Simon, 2006).

In this section, I will briefly recall the reasons behind the inefficiency and explore the solutions from the state of the art for handling high-dimensional measurements in a Kalman filtering setting.

The mathematical treatment presented in this Section is mostly adapted from (Simon, 2006) and (McManus and Barfoot, 2011).

2.2.1 Inefficiency due to high-dimensional measurements

The most general recipe for implementing a Kalman *correction* step is the following:

$$\begin{aligned} \mu_t &= \mu_t^- + K_t e_t, \\ P_t &= P_t^- - K_t P_{z,t} K_t^T, \\ e_t &= z_t - \mu_{z,t}, \\ K_t &= P_{xz,t} P_{z,t}^{-1}, \end{aligned} \tag{2.47}$$

where μ_t^- is the predicted mean of the state, $\mu_{z,t}$ is the predicted mean of the measurement, $P_{z,t}$ is the predicted covariance of the measurements and $P_{xz,t}$ is the predicted state-measurement cross-covariance. Each Kalman filter alternative that we have discussed, i.e. the KF, EKF and UKF variants, can be obtained by evaluating the quantities $\mu_{z,t}$, $P_{z,t}$ and $P_{xz,t}$ accordingly as shown in the previous sections.

As it can be seen from Eq. (2.47), the computation of the Kalman gain K_t requires the inversion of the predicted covariance $P_{z,t}$. The covariance is formally defined as

$$P_{z,t} = \mathrm{E}\left[(z_t - \mu_{z,t})(z_t - \mu_{z,t})^T | z_{1:t-1}\right], \tag{2.48}$$

where E is the expectation operator. If m is the size of the measurement vector z_t, the resulting covariance has size $m \times m$. In the case of high-dimensional measurements, inverting a matrix of that size might require non-negligible computation times and make a real-time implementation of the filter unfeasible. For this reason, several alternatives have been proposed in the literature such as the Kalman Information filter (Anderson and Moore, 2012) or the *sequential* Kalman filtering paradigm (Simon, 2006). In this Thesis, we adopted the latter approach.

2.2.2 Sequential Kalman filtering

Sequential Kalman filtering tries to overcome the computational complexity due to the inversion of large matrices by decomposing the Kalman correction step into several smaller correction steps each requiring the inversion of a considerably smaller matrix. The following mathematical derivations are taken from (Simon, 2006).

Assumptions

The sequential approach makes the following assumptions:

- the measurement model model in Eq. (2.8) is characterized by additive noise:

$$\begin{aligned} z_t &= h_t(x_t) + \nu_t, \\ \nu_t &\sim \mathcal{N}(0, R_t); \end{aligned} \tag{2.49}$$

- the function h_t is linear or nonlinear and differentiable;
- the measurement vector $z_t \in \mathbb{R}^m$ is divided in subvectors $z_{j,t} \in \mathbb{R}^{\tilde{m}}$ such that

$$z_t = \begin{bmatrix} z_{1,t}^T & z_{2,t}^T & \cdots z_{j,t}^T & \cdots & z_{M,t}^T \end{bmatrix}^T, \tag{2.50}$$

  and $m = \tilde{m}M$ where $\tilde{m} \geq 1$;
- the noise covariance matrix R_t is block-diagonal, i.e. the subvectors $z_{j,t}$ are uncorrelated:

$$R_t = \begin{bmatrix} R_{1,t} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & R_{M,t} \end{bmatrix}. \tag{2.51}$$

The methodology presented in the following can be only applied in combination with the theory of the Linear and Extended Kalman filters. Extensions to the Unscented Kalman filter are also possible but require further development (McManus and Barfoot, 2011) and are discussed later. For this reason, in the following, the measurement model in Eq. (2.49) will be substituted with its first-order Taylor approximation around a given mean μ

$$\begin{aligned} z_t &= h_t(x_t) + \nu_t \simeq h(\mu) + H_{x,t}(\mu)(x_t - \mu) + \nu_t, \\ \nu_t &\sim \mathcal{N}(0, R_t). \end{aligned} \tag{2.52}$$

$H_{x,t}$ is the Jacobian

$$H_{x,t}(\mu) = \left.\frac{\partial h_t(x,\nu)}{\partial x}\right|_{x=\mu,\nu=0}. \tag{2.53}$$

In case of a linear function h_t, Eq. (2.49) simplifies to

$$\begin{aligned} z_t &= h_t(x_t) + \nu_t = H_t x_t + \nu_t, \\ \nu_t &\sim \mathcal{N}(0, R_t), \end{aligned} \tag{2.54}$$

where H_t is the measurement matrix as defined in Eq. (2.10).

Sequential correction step

The sequential approach implements the Kalman correction update as a sequence of Kalman correction updates, each applied to the j-th subvector $z_{j,t}$ when j ranges from 1 to M. In the following, $\mu_{j,t}$ will indicate the corrected state after the j-th subvector has been incorporated. By convention, $\mu_{0,t}$ corresponds to the predicted state μ_t^-, obtained from the Kalman prediction step, while the last iterate $\mu_{M,t}$ corresponds to the actual corrected state μ_t. Similarly, $P_{j,t}$ will indicate the corrected covariance of the state after the j-th subvector has been processed.

The actual correction step is as follows. For $j = 1, \cdots, M$:

$$\begin{aligned} \mu_{j,t} &= \mu_{j-1,t} + K_{j,t} e_{j,t}, \\ P_{j,t} &= P_{j-1,t} - K_{j,t} P_{z,j,t} K_{j,t}^T, \\ e_{j,t} &= z_{j,t} - h_{j,t}, \\ P_{z,j,t} &= H_{x,j,t} P_{j-1,t} H_{x,j,t}^T + R_{j,t}, \\ K_{j,t} &= P_{j-1,t} H_{x,j,t}^T P_{z,j,t}^{-1}, \end{aligned} \tag{2.55}$$

where $K_{j,t}$ is the Kalman gain at the j-th iteration, $P_{z,j,t}$ is the predicted covariance of the j-th subvector of the measurement z_t, $R_{j,t}$ is the j-th diagonal block of R_t and $h_{j,t}$ is the subvector obtained by considering the rows of the vector $h_t(\mu_{j-1,t})$ in the interval $[\tilde{m}(j-1)+1, \tilde{m}j]$. $H_{x,j,t}$ is the submatrix obtained by considering the rows of the Jacobian $H_{x,t}(\mu_t^-)$ in the interval $[\tilde{m}(j-1)+1, \tilde{m}j]$. The matrix $H_{x,j,t}$ has size $\tilde{m} \times n$ where n is the size of the state vector x_t.

As it can be seen, the Sequential paradigm requires inverting the matrix $P_{z,j,t} \in \mathbb{R}^{\tilde{m}\times\tilde{m}}$. The size of this matrix can be made as small as required by acting on the number of subvectors M. In the limit case where $M = m$, the matrix $P_{z,j,t}$ becomes a scalar and no matrix inversions are required.

In the linear case, the correction step is the same as in Eq. (2.55) after substituting the Jacobian $H_{x,t}(\mu)$ with the measurement matrix H_t and the evaluation $h_t(\mu_{j-1,t})$ with $H_t\mu_{j-1,t}$.

The resulting algorithm

Algorithm 6. *Sequential Kalman filter*

Given a system described by the state model in Eq. (2.7) *and the measurement model in Eq.* (2.8)*, where the functions f_t and h_t are linear or nonlinear and differentiable, the measurement z_t is divided in subvectors*

$$z_t = \begin{bmatrix} z_{1,t}^T & z_{2,t}^T & \cdots z_{j,t}^T & \cdots & z_{M,t}^T \end{bmatrix}^T,$$

the measurement noise ν_t is additive with block-diagonal covariance matrix R_t

$$R_t = \begin{bmatrix} R_{1,t} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & R_{M,t} \end{bmatrix},$$

and an initial prior

$$p(x_0) \sim \mathcal{N}(\mu_0, P_0),$$

the predictive and filtering distributions of the state are approximately Gaussian and can be computed with the following Sequential Kalman filter prediction and correction steps. The distributions are exactly Gaussian if the functions f_t and h_t are linear.

- *Prediction step:*

The prediction step is the same as that of the Kalman filter in Eq. (2.20) *or the Extended Kalman filter in Eq.* (2.30) *when the function* f_t *is linear or nonlinear and differentiable, respectively.*

- *Correction step:*

  1. *Initialize the iterates* $\mu_{0,t}$ *and* $P_{0,t}$ *as follows:*

$$\begin{aligned} \mu_{0,t} &= \mu_t^-, \\ P_{0,t} &= P_t^-. \end{aligned} \tag{2.56}$$

  2. *For* $j = 1, \cdots, M$*, perform the following:*

$$\begin{aligned} \mu_{j,t} &= \mu_{j-1,t} + K_{j,t} e_{j,t}, \\ P_{j,t} &= P_{j-1,t} - K_{j,t} P_{z,j,t} K_{j,t}^T, \\ e_{j,t} &= z_{j,t} - h_{j,t}, \\ P_{z,j,t} &= H_{x,j,t} P_{j-1,t} H_{x,j,t}^T + R_{j,t}, \\ K_{j,t} &= P_{j-1,t} H_{x,j,t}^T P_{z,j,t}^{-1}, \end{aligned} \tag{2.57}$$

  where

$$\begin{aligned} h_{j,t} &= h_t(\mu_{j-1,t})_{(\tilde{m}(j-1)+1:\tilde{m}j)}, \\ H_{x,j,t} &= H_{x,t}(\mu_t^-)_{(\tilde{m}(j-1)+1:\tilde{m}j,1:n)}. \end{aligned} \tag{2.58}$$

  3. *Compute the approximation of the filtering distribution:*

$$\begin{aligned} p(x_t|z_{1:t}) &\simeq \mathcal{N}(x;\mu_t,P_t)\Big|_{x=x_t}, \\ \mu_t &= \mu_{M,t}, \\ P_t &= P_{M,t}. \end{aligned} \tag{2.59}$$

If the function h_t *is linear, the Jacobian* $H_{x,t}(\mu)$ *is substituted with the measurement matrix* H_t *as defined in Eq.* (2.10)*, and the term* $h_t(\mu_{j-1,t})$ *with* $H_t\mu_{j-1,t}$*.*

2.2.3 Sequential Unscented Kalman filtering

The Sequential Kalman filter relies on the theory of the Kalman filter and the Extended Kalman filter. Hence, it can be applied to any system described by the Eqs. (2.7) and

(2.8) where the functions f_t and h_t are linear or nonlinear but at least differentiable. If the functions are not differentiable, we could restort to the Unscented Kalman filter theory and attempt to apply the Sequential paradigm to it. Nevertheless, it can be shown (McManus and Barfoot, 2011) that, while this approach is attainable, it would result in an incorrect evaluation of the predicted covariance of the measurement $P_{z,t}$, hence to incorrect estimates.

In the next sections, we follow the derivations proposed in (McManus and Barfoot, 2011) which allow adapting the sequential paradigm to the theory of the Unscented Kalman filter without incurring in the aforementioned problems.

The statistical Jacobian form of the Unscented Kalman correction step

In the following, it will prove useful to rewrite the Unscented Kalman correction step in Algorithm 5 using the so-called statistical Jacobian form (McManus and Barfoot, 2011).

Given that the sequential paradigm assumes additive measurement noise, the Correction step simplifies as it is not required to extract the sigma points from the augmented variable (x_t, ν_t) (see Eq. (2.44)). Instead, the propagated sigma points $\mathcal{X}_t^{(i)}$ in Eq. (2.42) can be used directly. The resulting correction step is as follows:

1. Form the sigma points for the variable x_t:

$$\mathcal{X}_t^{-(i)} = \mathcal{X}_t^{(i)}, \quad i = 0, \dots, 2n, \tag{2.60}$$

   where $\mathcal{X}_t^{(i)}$ are the propagated sigma points as in Eq. (2.42) and n is the size of the state x_t.

2. Propagate the sigma points through the function h_t in Eq. (2.49):

$$\mathcal{Z}_t^{(i)} = h_t(\mathcal{X}_t^{-(i)}), \quad i = 0, \dots, 2n. \tag{2.61}$$

3. Compute the approximation of the filtering distribution:

$$
\begin{aligned}
p(x_t|z_{1:t}) &\simeq \mathcal{N}(x; \mu_t, P_t)\Big|_{x=x_t}, \\
\mu_t &= \mu_t^- + K_t e_t, \\
P_t &= P_t^- - K_t P_{z,t} K_t^T, \\
e_t &= z_t - \mu_{z,t}, \\
\mu_{z,t} &= \sum_{i=0}^{2n} \mathcal{W}_i^{(m)} \mathcal{Z}_t^{(i)}, \\
P_{z,t} &= \sum_{i=0}^{2n} \mathcal{W}_i^{(c)} \left(\mathcal{Z}_t^{(i)} - \mu_{z,t}\right)\left(\mathcal{Z}_t^{(i)} - \mu_{z,t}\right)^T + R_t, \\
P_{xz,t} &= \sum_{i=0}^{2n} \mathcal{W}_i^{(c)} \left(\mathcal{X}_t^{-(i)} - \mu_t^-\right)\left(\mathcal{Z}_t^{(i)} - \mu_{z,t}\right)^T, \\
K_t &= P_{xz,t} P_{z,t}^{-1},
\end{aligned}
\tag{2.62}
$$

where the weights $\mathcal{W}_i^{(m)}$ and $\mathcal{W}_i^{(c)}$ are defined as in Eq. (2.40), but with $\tilde{n}$ replaced by n.

As it can be seen, the uncertainty due to the measurement noise ν_t is handled by adding the noise covariance matrix R_t directly to the weighted sum in the evaluation of covariance $P_{z,t}$. This represents an adaptation of the Algorithm 4 for the Unscented approximation of a non-additive transform to the case of an additive transform.

Under the hypothesis that weights $\mathcal{W}_i^{(m)}$ are strictly positive, the following matrices $\mathcal{X}$ and $\mathcal{Z}$ are introduced:

$$
\mathcal{X} = \left[\sqrt{\mathcal{W}_0^{(c)}}\left(\mathcal{X}_t^{(0)} - \mu_t^-\right) \quad \cdots \quad \sqrt{\mathcal{W}_{2n}^{(c)}}\left(\mathcal{X}_t^{(2n)} - \mu_t^-\right)\right], \tag{2.63}
$$

$$
\mathcal{Z} = \left[\sqrt{\mathcal{W}_0^{(c)}}\left(\mathcal{Z}_t^{(0)} - \mu_{z,t}\right) \quad \cdots \quad \sqrt{\mathcal{W}_{2n}^{(c)}}\left(\mathcal{Z}_t^{(2n)} - \mu_{z,t}\right)\right]. \tag{2.64}
$$

Furthermore, recalling that the sequential paradigm assumes that the measurement z_t is divided in M subvectors, as in Eq. (2.50), the matrix $\mathcal{Z}$ is partitioned along its rows as follows:

$$
\mathcal{Z} = \begin{bmatrix} \mathcal{Z}_1 \\ \vdots \\ \mathcal{Z}_M \end{bmatrix}, \tag{2.65}
$$

where each submatrix $\mathcal{Z}_j$ is associated to the j-th subvector $z_{j,t}$. Using these definitions, the step 3 of the Unscented Kalman correction can be rewritten as follows:

$$
\begin{aligned}
p(x_t|z_{1:t}) &\simeq \mathcal{N}(x; \mu_t, P_t)\Big|_{x=x_t}, \\
\mu_t &= \mu_t^- + K_t e_t, \\
P_t &= P_t^- - K_t P_{z,t} K_t^T, \\
e_t &= z_t - \mu_{z,t}, \\
\mu_{z,t} &= \sum_{i=0}^{2n} \mathcal{W}_i^{(m)} \mathcal{Z}_t^{(i)}, \\
P_{z,t} &= \mathcal{Z}\mathcal{Z}^T + R_t, \\
P_{xz,t} &= \mathcal{X}\mathcal{Z}^T, \\
K_t &= P_{xz,t} P_{z,t}^{-1}.
\end{aligned}
\tag{2.66}
$$

Noting that $P_t^- = \mathcal{X}\mathcal{X}^T$, $P_{z,t}K_t^T = P_{xz,t}^T$ and substituting for e_t, K_t and $P_{xz,t}$, the above equations can be further simplified to:

$$
\begin{aligned}
p(x_t|z_{1:t}) &\simeq \mathcal{N}(x; \mu_t, P_t)\Big|_{x=x_t}, \\
\mu_t &= \mu_t^- + \mathcal{X}\mathcal{Z}^T P_{z,t}^{-1}\left(z_t - \mu_{z,t}\right), \\
P_t &= \mathcal{X}\mathcal{X}^T - \mathcal{X}\mathcal{Z}^T P_{z,t}^{-1} \mathcal{Z}\mathcal{X}^T, \\
\mu_{z,t} &= \sum_{i=0}^{2n} \mathcal{W}_i^{(m)} \mathcal{Z}_t^{(i)}, \\
P_{z,t} &= \mathcal{Z}\mathcal{Z}^T + R_t.
\end{aligned}
\tag{2.67}
$$

The notation used in (2.67) is also termed the *statistical Jacobian form* of the Unscented Kalman correction step (McManus and Barfoot, 2011).

‘Naive’ Sequential Unscented Kalman correction

Application of the sequential approach to the Unscented Kalman correction as presented in Eq. (2.67) would result in the following iteration. For $j = 1, \cdots, M$:

$$\begin{aligned} \mu_{j,t} &= \mu_{j-1,t} + \mathcal{X}\mathcal{Z}_j^T P_{z,j,t}^{-1} \left(z_{j,t} - \mu_{z,j,t} \right), \\ P_{j,t} &= \mathcal{X}\mathcal{X}^T - \mathcal{X}\mathcal{Z}_j^T P_{z,j,t}^{-1} \mathcal{Z}_j \mathcal{X}, \\ \mu_{z,j,t} &= (\mu_{z,t})_{(\tilde{m}(j-1)+1:\tilde{m}j)}, \\ P_{z,j,t} &= \mathcal{Z}_j \mathcal{Z}_j^T + R_{j,t}. \end{aligned} \tag{2.68}$$

The resulting mean $\mu_t = \mu_{M,T}$ can be obtained as follows:

$$\begin{aligned} \mu_t &= \mu_t^- + \sum_{j=1}^{M} \left(\mathcal{X}\mathcal{Z}_j^T P_{z,j,t}^{-1} \left(z_{j,t} - \mu_{z,j,t} \right) \right) \\ &= \mu_t^- + \mathcal{X} \sum_{j=1}^{M} \left(\mathcal{Z}_j^T P_{z,j,t}^{-1} \left(z_{j,t} - \mu_{z,j,t} \right) \right) \\ &= \mu_t^- + \mathcal{X}\mathcal{Z}^T \tilde{P}_{z,t}^{-1} \left(z_t - \mu_{z,t} \right), \end{aligned} \tag{2.69}$$

where

$$\tilde{P}_{z,t} = \begin{bmatrix} \mathcal{Z}_1\mathcal{Z}_1^T + R_{1,t} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \mathcal{Z}_M\mathcal{Z}_M^T + R_{M,t} \end{bmatrix}. \tag{2.70}$$

We notice that the above equality holds because the inverse of the block diagonal matrix $\tilde{P}_{z,t}$ is equal to a block diagonal matrix where the j-th block corresponds to the inverse $P_{z,j,t}^{-1}$.

Similarly, the state covariance P_t can be obtained as follows:

$$P_t = \mathcal{X}\mathcal{X}^T - \mathcal{X}\mathcal{Z}^T \tilde{P}_{z,t}^{-1} \mathcal{Z}\mathcal{X}^T. \tag{2.71}$$

The mean and covariance update steps in Eqs. (2.69) and (2.71) are very similar to those of the Unscented Kalman correction steps in Eq. (2.67). However, the predicted covariance of the measurement $P_{z,t}$ is substituted with the block diagonal matrix $\tilde{P}_{z,t}$. Comparing $\tilde{P}_{z,t}$ with the definition of $P_{z,t} = \mathcal{Z}\mathcal{Z}^T + R_t$, it is clear that they are different as $P_{z,t}$ would expand to

$$P_{z,t} = \begin{bmatrix} \mathcal{Z}_1\mathcal{Z}_1^T + R_{1,t} & \cdots & \mathcal{Z}_1\mathcal{Z}_M^T \\ \vdots & \ddots & \vdots \\ \mathcal{Z}_M\mathcal{Z}_1^T & \cdots & \mathcal{Z}_M\mathcal{Z}_M^T + R_{M,t} \end{bmatrix}, \tag{2.72}$$

i.e. $\tilde{P}_{z,t}$ is missing all the off-diagonal terms. In addition, it can be shown (McManus and Barfoot, 2011) that using $\tilde{P}_{z,t}$ in the state covariance update cannot guarantee that P_t is a positive definite matrix, as required by the theory of Kalman filtering. For these reasons, 'naively' processing the measurements sequentially in an Unscented Kalman filtering setting is not feasible. A different approach, proposed by McManus and Barfoot (2011), is presented in the following.

Sequential Unscented Kalman correction

The Sequential Unscented Kalman correction step proposed in (McManus and Barfoot, 2011), makes use of the well known Sherman-Morrison-Woodbury (SMW) identities (Gentle, 2007):

$$A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1} = (A + BCD)^{-1}, \tag{2.73}$$

$$A^{-1}B(C^{-1} + DA^{-1}B)^{-1} = (A + BCD)^{-1}BC. \tag{2.74}$$

Here, A, B, C and D are any dimensionally compatible matrices.

The proposed approach is obtained by considering that the state covariance update rule in (2.67) can be rewritten as follows:

$$\begin{aligned} P_t &= \mathcal{X}\mathcal{X}^T - \mathcal{X}\mathcal{Z}^T P_{z,t}^{-1}\mathcal{Z}\mathcal{X}^T \\ &= \mathcal{X}(\mathcal{X}^T - \mathcal{Z}^T P_{z,t}^{-1}\mathcal{Z}\mathcal{X}^T) \\ &= \mathcal{X}(I - \mathcal{Z}^T P_{z,t}^{-1}\mathcal{Z})\mathcal{X}^T \\ &= \mathcal{X}(I - \mathcal{Z}^T(\mathcal{Z}\mathcal{Z}^T + R_t)^{-1}\mathcal{Z})\mathcal{X}^T \\ &= \mathcal{X}(I + \mathcal{Z}^T R_t^{-1}\mathcal{Z})^{-1}\mathcal{X}^T, \end{aligned} \tag{2.75}$$

where, in the last step, the first form of the SMW identity has been applied with $A = I$, $B = \mathcal{Z}^T$, $C = R_t^{-1}$ and $D = \mathcal{Z}$.

R_t is block diagonal by hypothesis, hence its inverse is a diagonal matrix where the j-th block is given by $R_{j,t}^{-1}$. As a consequence, Eq. (2.75) further expands to

$$\begin{aligned} P_t &= \mathcal{X}(I + \sum_{j=1}^{M} \mathcal{Z}_j^T R_{j,t}^{-1} \mathcal{Z}_j)^{-1} \mathcal{X}^T \\ &:= \mathcal{X}\mathcal{C}\mathcal{X}^T, \end{aligned} \tag{2.76}$$

where the quantity $\mathcal{C}$ has been defined as

$$\mathcal{C} := (I + \sum_{j=1}^{M} \mathcal{Z}_j^T R_{j,t}^{-1} \mathcal{Z}_j)^{-1}. \tag{2.77}$$

The quantity $\mathcal{C}^{-1}$ can be computed sequentially by summing up M contributions and has dimension $2n+1 \times 2n+1$ given that each Z_j has dimension $\tilde{m} \times 2n+1$ and each $R_{j,t}$ has size $\tilde{m} \times \tilde{m}$ where n is size of the state x_t. Given that in a high-dimensional measurements setting, where the measurement $z_t \in \mathbb{R}^m$, it is assumed that $m \gg n$, the matrix $\mathcal{C}^{-1}$ is relatively inexpensive to invert.

Similarly, it is possible to extend this approach to the state mean update rule in (2.67). It can be rewritten as follows:

$$\begin{aligned} \mu_t &= \mu_t^- + \mathcal{X}\mathcal{Z}^T P_{z,t}^{-1} (z_t - \mu_{z,t}) \\ &= \mu_t^- + \mathcal{X}\mathcal{Z}^T (\mathcal{Z}\mathcal{Z}^T + R_t)^{-1} (z_t - \mu_{z,t}) \\ &= \mu_t^- + \mathcal{X}(I + \mathcal{Z}^T R_t^{-1} \mathcal{Z})^{-1} \mathcal{Z}^T R_t^{-1} (z_t - \mu_{z,t}), \end{aligned} \tag{2.78}$$

where, in the last step, the second form of the SMW identity has been applied with $A = I$, $B = \mathcal{Z}^T$, $C = R_t^{-1}$ and $D = \mathcal{Z}$. Using again the fact that R_t is block diagonal, (2.78) further expands to

$$\begin{aligned} \mu_t &= \mu_t^- + \mathcal{X}(I + \sum_{j=1}^{M} \mathcal{Z}_j^T R_{j,t}^{-1} \mathcal{Z}_{j,t})^{-1} \left(\sum_{j=1}^{M} \mathcal{Z}_j^T R_{j,t}^{-1} (z_{j,t} - \mu_{z,j,t}) \right) \\ &:= \mu_t^- + \mathcal{X}\mathcal{C}d, \end{aligned} \tag{2.79}$$

where the quantity d has been defined

$$d := \sum_{j=1}^{M} \mathcal{Z}_j^T R_{j,t}^{-1} (z_t - \mu_{z,j,t}). \tag{2.80}$$

Remarkably, the state mean and state covariance update rules in Eqs. (2.79) and (2.76) are obtained from the statistical Jacobian form of the Unscented Kalman correction step, in Eq. (2.67), by applying the Sherman-Morrison-Woodbury identities. Hence, these rules provide an algebraically identical result to Unscented Kalman correction step where all the sub-measurements are condensed in a single measurement vector z_t. For this reason, these rules are also theoretically sound in terms of the positive definiteness of the obtained state covariance matrix P_t.

Resulting algorithm

Algorithm 7. *Sequential Unscented Kalman filter*

Given a system described by the state model in Eq. (2.7) *and the measurement model in Eq.* (2.8)*, where the functions f_t and h_t are nonlinear, the measurement z_t is divided in subvectors*

$$z_t = \begin{bmatrix} z_{1,t}^T & z_{2,t}^T & \cdots z_{j,t}^T & \cdots & z_{M,t}^T \end{bmatrix}^T,$$

the measurement noise ν_t is additive with block-diagonal covariance matrix R_t

$$R_t = \begin{bmatrix} R_{1,t} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & R_{M,t} \end{bmatrix},$$

and an initial prior

$$p(x_0) \sim \mathcal{N}(\mu_0, P_0),$$

the predictive and filtering distributions of the state are approximately Gaussian and can be computed with the following Sequential Unscented Kalman filter prediction and correction steps.

- *Prediction step:*

  The prediction step is the same as that of the Unscented Kalman filter in Eqs. (2.41)−(2.43)*.*

- *Correction step:*

  1. *Form the sigma points for the variable x_t:*

$$\mathcal{X}_t^{-(i)} = \mathcal{X}_t^{(i)}, \quad i = 0, \dots, 2n, \tag{2.81}$$

where $\mathcal{X}_t^{(i)}$ *are the propagated sigma points as in Eq.* (2.42) *and* n *is the size of the state* x_t*.*

2. *Propagate the sigma points through the function* h_t *in Eq.* (2.49)*:*

$$\mathcal{Z}_t^{(i)} = h_t(\mathcal{X}_t^{-(i)}), \quad i = 0, \dots, 2n. \tag{2.82}$$

3. *Evaluate the statistical Jacobian matrices* $\mathcal{X}_t$*,* $\mathcal{Z}_t$ *and the predicted mean of the measurements* $\mu_{z,t}$*:*

$$\begin{aligned} \mathcal{X}_t &= \left[\sqrt{\mathcal{W}_0^{(c)}} \left(\mathcal{X}_t^{(0)} - \mu_t^- \right) \quad \cdots \quad \sqrt{\mathcal{W}_{2n}^{(c)}} \left(\mathcal{X}_t^{(2n)} - \mu_t^- \right) \right], \\ \mathcal{Z}_t &= \left[\sqrt{\mathcal{W}_0^{(c)}} \left(\mathcal{Z}_t^{(0)} - \mu_{z,t} \right) \quad \cdots \quad \sqrt{\mathcal{W}_{2n}^{(c)}} \left(\mathcal{Z}_t^{(2n)} - \mu_{z,t} \right) \right] \\ &= \left[\mathcal{Z}_{1,t}^T \quad \dots \quad \mathcal{Z}_{M,t}^T \right]^T, \\ \mu_{z,t} &= \sum_{i=0}^{2n} \mathcal{W}_i^{(m)} \mathcal{Z}_t^{(i)}, \end{aligned} \tag{2.83}$$

where the weights $\mathcal{W}_i^{(m)}$ *and* $\mathcal{W}_i^{(c)}$ *are defined as in Eq.* (2.40)*, but with* $\tilde{n}$ *replaced by* n*, and are strictly positive.*

4. *Initialize the iterates* $d_{0,t}$ *and* $C_{0,t}^{-1}$ *as follows:*

$$\begin{aligned} d_{0,t} &= 0, \\ \mathcal{C}_{0,t}^{-1} &= I. \end{aligned} \tag{2.84}$$

5. *For* $j = 1, \cdots, M$*, perform the following:*

$$\begin{aligned} d_{j,t} &= d_{j-1,t} + \mathcal{Z}_{j,t}^T R_{j,t}^{-1} (z_{j,t} - \mu_{z,j,t}), \\ C_{j,t}^{-1} &= C_{j-1,t}^{-1} + \mathcal{Z}_{j,t}^T R_{j,t}^{-1} \mathcal{Z}_{j,t}, \end{aligned} \tag{2.85}$$

where

$$\mu_{z,j,t} = (\mu_{z,t})_{(\tilde{m}(j-1)+1:\tilde{m}j)}. \tag{2.86}$$

6. Compute the approximation of the filtering distribution:

$$\begin{aligned} p(x_t|z_{1:t}) &\simeq \mathcal{N}(x; \mu_t, P_t)\Big|_{x=x_t}, \\ \mu_t &= \mu_t^- + \mathcal{X}_t \mathcal{C}_t d_t, \\ P_t &= P_t^- + \mathcal{X}_t \mathcal{C}_t \mathcal{X}_t^T, \end{aligned} \tag{2.87}$$

where

$$\begin{aligned} d_t &= d_{M,t}, \\ \mathcal{C}_t^{-1} &= \mathcal{C}_{M,t}^{-1}. \end{aligned} \tag{2.88}$$

2.3 Kalman filtering for orientation tracking

In the previous sections, we presented the theory of Kalman filtering for linear and nonlinear systems and later extended to the case of high-dimensional measurements. As mentioned in the Sec. 2.1.1, all the algorithms presented thus far are valid under the assumption that both the state x_t and the measurements z_t admit a representation in terms of Euclidean vectors, i.e. they belong to a vector space.

The main focus of his Thesis is on tracking the 6D pose T_t

$$T_t = \begin{bmatrix} R_t & p_t \\ 0^T & 1 \end{bmatrix} \tag{2.89}$$

of a rigid body, where $p_t \in \mathbb{R}^3$ is a 3D Euclidean vector representing the position of the object and $R_t \in \mathrm{SO}(3)$ is a rotation matrix representing the orientation of the object. We recall that SO(3) indicates the set defined as follows (Lynch and Park, 2017):

$$\mathrm{SO}(3) = \{R \in \mathbb{R}^{3\times 3} \mid RR^T = I \text{ and } \det(R) = 1\}. \tag{2.90}$$

While the translational component p_t admits an Euclidean representation within a 3D vector space, the rotational component R_t belongs to a kind of set termed *group* that satisfies specific constraints as in Eq. (2.90). These constraints convert into additional design requirements in the development of Kalman filtering algorithms.

In this section, I will present some of the possible extensions to the Kalman prediction and correction steps when the state x_t and/or the measurements z_t contain rotational components. Specifically, I will consider two possible parametrizations of the rotation R_t: a more classical one based on Euler angles and a more modern one based on the theory of unit quaternions.

2.3.1 Parametrization with Euler angles

The Euler angles (Lynch and Park, 2017) represent one of the most classical ways of parametrizing the orientation of a rigid body with respect to a given fixed reference frame. Given a rotation matrix R representing the orientation, its parametrization in terms of the Euler angles ϕ, χ and ψ is obtained by expressing R as the product of three rotation matrices depending on those angles:

$$R = R_1(\phi)R_2(\chi)R_3(\psi). \tag{2.91}$$

Here, R_i is an elementary rotation matrix around the x, y or z axis of the fixed reference frame. The number of possible combinations of the three axes is limited to two sets of combinations termed the proper Euler angles and the Tait-Byran angles. A very common combination is the so called Roll-Pitch-Yaw (RPY) convention, or ZYX,

$$R = R_z(\phi)R_y(\chi)R_x(\psi), \tag{2.92}$$

consisting in a rotation around the x axis of the fixed reference frame, followed by a rotation around the y fixed axis and finally a rotation around the z fixed axis.

Using the Euler angles, it is possible to represent a rotation using three scalars, ϕ, χ and ψ, and interpret the vector

$$\begin{bmatrix} \phi \\ \chi \\ \psi \end{bmatrix} \in \mathbb{R}^3 \tag{2.93}$$

as a 3D Euclidean representation of the rotation. Although this assumption is not totally sound, this choice allows using the Kalman prediction and correction steps presented in the previous section with some minimal changes. In the following discussion, it will be assumed that the state x_t and/or the measurements z_t are in the form of three Euler angles.

Definition of suitable operators for the angular components

The most critical aspect in using this representation is that some of the steps within a Kalman filter requires the evaluation of the sum, the difference and the weighted mean among components belonging to the state space or the measurement space. Examples are:

- the evaluation of the innovation:

$$e_t = z_t - \mu_{z,t};$$

- the evaluation of the corrected mean of the state:

$$\mu_t = \mu_t^- + K_t e_t;$$

- the evaluation of the sigma points (in the Unscented filtering setting):
  - in the prediction step

$$\mathcal{X}_t^{-(i)} = \mu_{t-1} \pm \sqrt{n+\lambda}\left[\sqrt{P_{t-1}}\right]_i;$$

  - in the correction step

$$\mathcal{X}_t^{(i)} = \mu_t^- \pm \sqrt{n+\lambda}\left[\sqrt{P_t^-}\right]_i;$$

- the evaluation of the predicted mean and covariance of the state (in the Unscented filtering setting):

$$\mu_t^- = \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(m)} \mathcal{X}_t^{(i)}, \quad P_t^- = \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(c)} \left(\mathcal{X}_t^{(i)} - \mu_t^-\right)\left(\mathcal{X}_t^{(i)} - \mu_t^-\right)^T;$$

- the evaluation of the predicted mean and covariance of the measurements (in the Unscented filtering setting):

$$\mu_{z,t} = \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(m)} \mathcal{Z}_t^{(i)}, \quad P_{z,t} = \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(c)} \left(\mathcal{Z}_t^{(i)} - \mu_{z,t}\right)\left(\mathcal{Z}_t^{(i)} - \mu_{z,t}\right)^T.$$

When these operations act on angular entities, as the Euler angles, it is not guaranteed that the result of the operation is still meaningful in a Kalman filtering setting. As an example, evaluating a scalar innovation e_t where the measurement z_t assumes the value 2π and the predicted mean of the measurement $\mu_{z,t}$ assumes the value -2π would result in

$$e_t = 2\pi - (-2\pi) = 4\pi.$$

However, given that 2π and -2π actually represent the same angle, we would expect that the innovation evaluates to zero.

In order to properly evaluates the result of the operations acting on the angular components, a crude but commonly adopted solution consists in substituting the sum operator $+$ between scalar angular components with the following operator $\oplus$:

$$\alpha \oplus \beta \coloneqq \arg(\exp(j(\alpha + \beta))), \tag{2.94}$$

where α and β are angular components, j is the complex imaginary unit such that $j^2 = -1$, exp() is the complex exponential function and arg() is the argument of a complex number. It is recalled that the argument of a complex number $z = x + jy$ is formally defined as the solution of the pair of equations

$$\begin{aligned} \frac{x}{|z|} &= \cos(\arg(z)), \\ \frac{y}{|z|} &= \sin(\arg(z)). \end{aligned} \tag{2.95}$$

Using the redefined operator, the sum between the angular components is evaluated within the *complex unit circle* which takes into account the circular nature of the angular components α and β.

Similarly, the difference operator $-$ can be substituted with the $\ominus$ operator defined as:

$$\alpha \ominus \beta \coloneqq \alpha \oplus (-\beta). \tag{2.96}$$

Finally, the weighted mean of angular components α_i

$$\sum \lambda_i \alpha_i, \tag{2.97}$$

where $\lambda_i \in \mathbb{R}$ are the weights, can be substituted with the following *circular mean*

$$\arg\left(\sum \lambda_i \exp(j\alpha_i)\right), \tag{2.98}$$

which corresponds to finding the mean direction among the directions $\exp(j\alpha_i)$ in the complex plane given the weights λ_i.

Example

As an example, the following algorithm shows how to apply the operators defined in Eqs. (2.94) – (2.98) in order to implement an Unscented Kalman filter when the state x_t and/or the measurements z_t represent the orientation of a rigid object parametrized using the Euler angles.

Algorithm 8. *Unscented Kalman filtering using Euler angles*

Given a system described by the state model in Eq. (2.7) *and the measurement model in Eq.* (2.8)*, where the functions f_t and h_t are nonlinear and the state x_t and/or the measurements z_t represent the orientation of a rigid body in terms of three Euler angles*

$$\begin{bmatrix} \phi \\ \chi \\ \psi \end{bmatrix},$$

and an initial prior

$$p(x_0) \sim \mathcal{N}(\mu_0, P_0),$$

the predictive and filtering distributions of the state are approximately Gaussian and can be computed with the following prediction and correction steps:

- *Prediction step:*

  1. *Form the sigma points for the augmented variable (x_{t-1}, w_{t-1}):*

$$\begin{aligned}
\mathcal{X}_{t-1}^{(0)} &= \tilde{\mu}_{t-1}, \\
\mathcal{X}_{t-1}^{(i)} &= \tilde{\mu}_{t-1} \oplus \sqrt{\tilde{n}+\lambda} \left[\sqrt{\tilde{P}_{t-1}} \right]_i, \\
\mathcal{X}_{t-1}^{(i+n)} &= \tilde{\mu}_{t-1} \ominus \sqrt{\tilde{n}+\lambda} \left[\sqrt{\tilde{P}_{t-1}} \right]_i, \quad i = 1, \dots, \tilde{n}, \\
\tilde{\mu}_{t-1} &= \begin{bmatrix} \mu_{t-1} \\ 0 \end{bmatrix}, \quad \tilde{P}_{t-1} = \begin{bmatrix} P_{t-1} & 0 \\ 0 & Q_{t-1} \end{bmatrix},
\end{aligned} \tag{2.99}$$

  where $\tilde{n}$ is the size of the augmented variable and λ is defined as in Eq. (2.39)*.*

2. *Propagate the sigma points through the function f_t in Eq. (2.7):*

$$\mathcal{X}_t^{(i)} = f_t(\mathcal{X}_{t-1}^{(i),x}, \mathcal{X}_{t-1}^{(i),w}), \quad i = 0, \ldots, 2\tilde{n}, \tag{2.100}$$

where $\mathcal{X}_{t-1}^{(i),x}$ and $\mathcal{X}_{t-1}^{(i),w}$ denote the part of the sigma point associated to x and w respectively.

3. *Compute the approximation of the predictive distribution:*

$$\begin{aligned} p(x_t|z_{1:t-1}) &\simeq \mathcal{N}(x;\mu_t^-, P_t^-)\Big|_{x=x_t}, \\ \mu_t^- &= \arg\left(\sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(m)} \exp\left(j\mathcal{X}_t^{(i)}\right)\right), \\ P_t^- &= \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(c)} \left(\mathcal{X}_t^{(i)} \ominus \mu_t^-\right)\left(\mathcal{X}_t^{(i)} \ominus \mu_t^-\right)^T, \end{aligned} \tag{2.101}$$

where the weights $\mathcal{W}_i^{(m)}$ and $\mathcal{W}_i^{(c)}$ are defined as in Eq. (2.40).

- *Correction step:*

1. *Form the sigma points for the augmented variable (x_t, ν_t):*

$$\begin{aligned} \mathcal{X}_t^{-(0)} &= \tilde{\mu}_t^-, \\ \mathcal{X}_t^{-(i)} &= \tilde{\mu}_t^- \oplus \sqrt{\tilde{n}+\lambda}\left[\sqrt{\tilde{P}_t^-}\right]_i, \\ \mathcal{X}_t^{-(i+n)} &= \tilde{\mu}_t^- \ominus \sqrt{\tilde{n}+\lambda}\left[\sqrt{\tilde{P}_t^-}\right]_i, \quad i = 1, \ldots, \tilde{n}, \\ \tilde{\mu}_t &= \begin{bmatrix} \mu_t^- \\ 0 \end{bmatrix}, \quad \tilde{P}_t^- = \begin{bmatrix} P_t^- & 0 \\ 0 & R_t \end{bmatrix}. \end{aligned} \tag{2.102}$$

2. *Propagate the sigma points through the function h_t in Eq. (2.8):*

$$\mathcal{Z}_t^{(i)} = h_t(\mathcal{X}_t^{-(i),x}, \mathcal{X}_t^{-(i),\nu}), \quad i = 0, \ldots, 2\tilde{n}, \tag{2.103}$$

where $\mathcal{X}_t^{-(i),x}$ and $\mathcal{X}_t^{-(i),\nu}$ denote the part of the sigma point associated to x and ν respectively.

3. Compute the approximation of the filtering distribution:

$$
\begin{aligned}
p(x_t|z_{1:t}) \simeq \mathcal{N}(x;\mu_t,P_t)\Big|_{x=x_t}&,\\
\mu_t &= \mu_t^- \oplus K_t e_t,\\
P_t &= P_t^- - K_t P_{z,t} K_t^T,\\
e_t &= z_t \ominus \mu_{z,t},\\
\mu_{z,t} &= \arg\left(\sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(m)} \exp(j\mathcal{Z}_t^{(i)})\right),\\
P_{z,t} &= \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(c)} \left(\mathcal{Z}_t^{(i)} \ominus \mu_{z,t}\right)\left(\mathcal{Z}_t^{(i)} \ominus \mu_{z,t}\right)^T,\\
P_{xz,t} &= \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(c)} \left(\mathcal{X}_t^{-(i),x} \ominus \mu_t^-\right)\left(\mathcal{Z}_t^{(i)} \ominus \mu_{z,t}\right)^T,\\
K_t &= P_{xz,t} P_{z,t}^{-1}.
\end{aligned}
\tag{2.104}
$$

We notice that the operators exp(), arg(), $\oplus$ and $\ominus$ are to be intended as they act separately on each scalar component of the vectors representing the state, the measurements and derived quantities (e.g. means and sigma points). Furthermore, if a vector contains components that are not angular, e.g. because the state and/or the measurements contain additional Euclidean components or noise components, as in Eqs. (2.99) and (2.102), the redefined operators are to be applied only to the parts of the vector that contain the angular components.

2.3.2 Parametrization with unit quaternions

Unit quaternions (Lynch and Park, 2017; Sola et al., 2018) represent an alternative way to parametrize the orientation of a rigid body. The following definitions are taken from (Lynch and Park, 2017).

By analogy with the complex numbers, a quaternion $q \in \mathbb{H}$ is an expression of the form

$$q = q_w + q_x i + q_y j + q_z k, \tag{2.105}$$

where q_w is the real part, q_x, q_y and q_z are the imaginary parts and the symbols i, j and k are such that $i^2 = j^2 = k^2 = ijk = -1$. The quaternion q is sometimes expressed directly as a four dimensional vector

$$q = (q_w, q_v), \tag{2.106}$$

where $q_v = (q_x, q_y, q_z)$ and q_w are called the vectorial and scalar part of the quaternion q, respectively. The quaternions have a multiplicative group structure. In this respect, the quaternion product q_p between two quaternions q_1 and q_2 is defined as follows:

$$\begin{aligned} q_p &= q_1 q_2 = (q_{p,w}, q_{p,v}), \\ q_{p,w} &= q_{1,w} q_{2,w} - q_{1,v} q_{2,v}, \\ q_{p,v} &= q_{1,w} q_{2,v} + q_{2,w} q_{1,v} + q_{1,v} \times q_{2,v}, \end{aligned} \tag{2.107}$$

where $q_{1,v} q_{2,v}$ and $q_{1,v} \times q_{2,v}$ are the scalar and cross product between the vector components of the two quaternions, respectively. For a given quaternion q, it is possible to define a norm $\|q\|$ as follows:

$$\|q\| = \sqrt{q_w^2 + \|q_v\|^2}, \tag{2.108}$$

where $\|q_v\|$ is the norm of the vectorial part of the quaternion.

Quaternions with unitary norm, termed *unit quaternions*, can be used to parametrize the orientation of a rigid body. We recall that for a given rotation matrix $R \in \mathrm{SO}(3)$ it is possible to associate an axis-angle representation (n, θ) corresponding to a rotation around the 3D axis $n = (n_x, n_y, n_z)$ by an amount of θ. R can be written as a function of the axis and the angle as follows:

$$R(n, \theta) = \cos(\theta) I + \sin(\theta)\hat{n} + (1 - \cos(\theta))\hat{n}^2, \tag{2.109}$$

where $\hat{n}$ is the cross product matrix of the axis

$$\hat{n} = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix}. \tag{2.110}$$

Starting from the axis-angle representation, it can be shown (Lynch and Park, 2017) that the quaternion

$$q(n, \theta) = \left(\cos\left(\frac{\theta}{2}\right), n \sin\left(\frac{\theta}{2}\right)\right) \tag{2.111}$$

is a proper quaternion, it has unitary norm and can be used to represent the rotation of a rigid body. Given a quaternion q representing a rotation R, the same rotation is also represented by the quaternion $-q$. In this respect, it is said that the quaternions double cover the space of the rotations. The inverse of a quaternion, i.e. the quaternion representing the rotation R^T, can be obtained as $q^{-1} = (-q_w, q_v)$.

One of the advantages of using the unit quaternions as opposed to the Euler angles is that they are a proper Lie group (Sola et al., 2018), i.e. a smooth manifold for which a tangent space can be defined at each point. The tangent space is a vector space on which it is possible to properly define operations like additions, subtractions and averaging, as required by the Kalman filter algorithm. From a geometric standpoint, using the unit quaternion implies that the state vector x_t and/or the measurements z_t evolve on a spherical 3-dimensional surface in 4-dimensional space that is called the 3-sphere S^3. The fact that the surface is spherical is a direct consequence of the unitary norm constraint, i.e. $\|q\| = 1$, which in turn is a direct consequence of the constraints defining the SO(3) group.

The following mathematical treatment is adapted from the theory presented in Chiella et al. (2019); Lynch and Park (2017); Sola et al. (2018).

The Lie algebra of the unit quaternions

As for any Lie group, it is is possible to define a tangent space to any point of the manifold of the unit quaternions. Among the possible tangent spaces, it is of interest to consider the so called Lie algebra $\mathfrak{so}(3)$, defined as the tangent space at the identity rotation. The elements belonging to this space are the 3×3 skew-symmetric matrices.

The Lie algebra is a vector space, hence its elements can be identified with vectors ω in $\mathbb{R}^n$ where $n = 3$ is the number of degrees of freedom of SO(3). Intuitively, these vectors can be seen as *perturbation vectors* or *angular velocities* such that if the reference frame attached to the rigid body follows that angular velocity for one unit time, the overall orientation changes from the identity rotation to a new rotation R.

Given a Lie group, it is always possible to define two maps, the exponential map exp() and its inverse log(), that allows moving from the Lie algebra to the group and vice versa. Given that the quaternions are used to represent the rotations and that $\mathfrak{so}(3)$ can be identified with $\mathbb{R}^3$, these maps transform a quaternion $q = (q_w, q_v) \in \mathbb{H}$ to the vector representation $\omega \in \mathbb{R}^3$ in its Lie algebra and vice versa. The maps are the following:

$$q = \exp(\omega) := \cos\left(\frac{\|\omega\|}{2}\right) + \frac{\omega}{\|\omega\|}\sin\left(\frac{\|\omega\|}{2}\right) \quad \in \mathbb{H}, \ \ \|\omega\| \neq 0, \tag{2.112}$$

$$\omega = \log(q) := 2q_v \frac{\mathrm{atan}(\|\mathrm{q_v}\|, \mathrm{q_w})}{\|q_v\|} \quad \in \mathbb{R}^3, \ \ \|q_v\| \neq 0. \tag{2.113}$$

In case $\|\omega\| = 0$, the exponential evaluates to the identity quaternion, $q = 1$. Similarly, in the case $\|q_v\| = 0$, the logarithm evaluates to the null vector $\omega = 0$. In order to avoid problems due to the double cover of the quaternions, it is necessary to ensure that the scalar part q_w is positive before evaluating the logarithm. If it is not, the quaternion $-q$ should be used instead.

Definition of suitable operators for the quaternion components

As done for the Euler angles, it is now possible to proceed with the definition of suitable operators for the sum, the difference and the weighted mean involving unit quaternions and their representatives in the Lie algebra (Chiella et al., 2019).

In the following, it will be useful to define the "sum" between a quaternion $q_1 \in \mathbb{H}$ and a vector $\omega_2 \in \mathbb{R}^3$ in the Lie algebra as follows:

$$q_1 \oplus \omega_2 := \exp(\omega_2)q_1 \in \mathbb{H}. \tag{2.114}$$

It will also be required to define the "difference" between two quaternions q_1 and q_2 as follows:

$$q_1 \ominus q_2 := \log(q_1 q_2^{-1}) \in \mathbb{R}^3. \tag{2.115}$$

Finally, the weighted mean of a set of quaternions $q_i \in \mathbb{H}$ with weights $\lambda_i \in \mathbb{R}$ can be computed by considering the eigenvector corresponding to the maximum eigenvalue of the matrix

$$\sum \lambda_i q_i q_i^T \in \mathbb{R}^{4\times 4}. \tag{2.116}$$

The above defined mean will be indicated as $\bar{q}(\{q_i\}, \{\lambda_i\})$.

Example

As done for the Euler angles, we provide an example showing how to implement an Unscented Kalman filter when the state x_t and/or the measurements z_t represent the orientation of a rigid body parametrized using unit quaternions.

In the following, we assume that the columns belonging to the state covariance P_t can be interpreted as "perturbation" vectors belonging to the Lie algebra of the unit quaternions, i.e. they can be represented as vectors in $\mathbb{R}^3$ (Chiella et al., 2019). As a result, the dimension of the covariance matrix is 3×3 (instead of 4×4).

Algorithm 9. *Unscented Kalman filtering using unit quaternions*

Given a system described by the state model in Eq. (2.7) *and the measurement model in Eq.* (2.8)*, where the functions f_t and h_t are nonlinear and the state x_t and/or the measurements z_t represent the orientation of a rigid body in terms of a unit quaternion $q \in \mathbb{H}$, and an initial prior*

$$p(x_0) \sim \mathcal{N}(\mu_0, P_0),$$

the predictive and filtering distributions of the state are approximately Gaussian and can be computed with the following prediction and correction steps:

- *Prediction step:*

  1. *Form the sigma points for the augmented variable (x_{t-1}, w_{t-1}):*

$$\begin{aligned} \mathcal{X}_{t-1}^{(0)} &= \tilde{\mu}_{t-1}, \\ \mathcal{X}_{t-1}^{(i)} &= \tilde{\mu}_{t-1} \oplus \sqrt{\tilde{n}+\lambda}\left[\sqrt{\tilde{P}_{t-1}}\right]_i, \\ \mathcal{X}_{t-1}^{(i+n)} &= \tilde{\mu}_{t-1} \oplus \sqrt{\tilde{n}+\lambda}\left[-\sqrt{\tilde{P}_{t-1}}\right]_i, \quad i = 1, \dots, \tilde{n}, \\ \tilde{\mu}_{t-1} &= \begin{bmatrix} \mu_{t-1} \\ 0 \end{bmatrix}, \quad \tilde{P}_{t-1} = \begin{bmatrix} P_{t-1} & 0 \\ 0 & Q_{t-1} \end{bmatrix}, \end{aligned} \tag{2.117}$$

  where $\tilde{n}$ is the size of the augmented variable and λ is defined as in Eq. (2.39)*.*

  2. *Propagate the sigma points through the function f_t in Eq.* (2.7)*:*

$$\mathcal{X}_t^{(i)} = f_t(\mathcal{X}_{t-1}^{(i),x}, \mathcal{X}_{t-1}^{(i),w}), \quad i = 0, \dots, 2\tilde{n}, \tag{2.118}$$

  where $\mathcal{X}_{t-1}^{(i),x}$ and $\mathcal{X}_{t-1}^{(i),w}$ denote the part of the sigma point associated to x and w respectively.

3. *Compute the approximation of the predictive distribution:*

$$\begin{aligned}
p(x_t|z_{1:t-1}) &\simeq \mathcal{N}(x;\mu_t^-,P_t^-)\Big|_{x=x_t}, \\
\mu_t^- &= \bar{q}(\{\mathcal{X}_t^{(i)}\},\{\mathcal{W}_i^{(m)}\}), \\
P_t^- &= \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(c)} \left(\mathcal{X}_t^{(i)} \ominus \mu_t^-\right)\left(\mathcal{X}_t^{(i)} \ominus \mu_t^-\right)^T,
\end{aligned} \tag{2.119}$$

where the weights $\mathcal{W}_i^{(m)}$ *and* $\mathcal{W}_i^{(c)}$ *are defined as in Eq.* (2.40)*.*

- *Correction step:*

1. *Form the sigma points for the augmented variable* (x_t, ν_t)*:*

$$\begin{aligned}
\mathcal{X}_t^{-(0)} &= \tilde{\mu}_t^-, \\
\mathcal{X}_t^{-(i)} &= \tilde{\mu}_t^- \oplus \sqrt{\tilde{n}+\lambda}\left[\sqrt{\tilde{P}_t^-}\right]_i, \\
\mathcal{X}_t^{-(i+n)} &= \tilde{\mu}_t^- \oplus \sqrt{\tilde{n}+\lambda}\left[-\sqrt{\tilde{P}_t^-}\right]_i, \quad i=1,\ldots,\tilde{n}, \\
\tilde{\mu}_t &= \begin{bmatrix} \mu_t^- \\ 0 \end{bmatrix}, \quad \tilde{P}_t^- = \begin{bmatrix} P_t^- & 0 \\ 0 & R_t \end{bmatrix}.
\end{aligned} \tag{2.120}$$

2. *Propagate the sigma points through the function* h_t *in Eq.* (2.8)*:*

$$\mathcal{Z}_t^{(i)} = h_t(\mathcal{X}_t^{-(i),x}, \mathcal{X}_t^{-(i),\nu}), \quad i=0,\ldots,2\tilde{n}, \tag{2.121}$$

where $\mathcal{X}_t^{-(i),x}$ *and* $\mathcal{X}_t^{-(i),\nu}$ *denote the part of the sigma point associated to* x *and* ν *respectively.*

3. Compute the approximation of the filtering distribution:

$$\begin{aligned}
p(x_t|z_{1:t}) &\simeq \mathcal{N}(x;\mu_t,P_t)\Big|_{x=x_t}, \\
\mu_t &= \mu_t^- \oplus K_t e_t, \\
P_t &= P_t^- - K_t P_{z,t} K_t^T, \\
e_t &= z_t \ominus \mu_{z,t}, \\
\mu_{z,t} &= \bar{q}(\{\mathcal{Z}_t^{(i)}\},\{\mathcal{W}_i^{(m)}\}), \\
P_{z,t} &= \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(c)} \left(\mathcal{Z}_t^{(i)} \ominus \mu_{z,t}\right)\left(\mathcal{Z}_t^{(i)} \ominus \mu_{z,t}\right)^T, \\
P_{xz,t} &= \sum_{i=0}^{2\tilde{n}} \mathcal{W}_i^{(c)} \left(\mathcal{X}_t^{-(i),x} \ominus \mu_t^-\right)\left(\mathcal{Z}_t^{(i)} \ominus \mu_{z,t}\right)^T, \\
K_t &= P_{xz,t} P_{z,t}^{-1}.
\end{aligned} \tag{2.122}$$

We notice that if the state and/or the measurements contain components that are not represented using quaternions, e.g. because they contain additional Euclidean components or noise components, as in Eqs. (2.117) and (2.120), the redefined operators $\oplus$, $\ominus$ and $\bar{q}$ are to be applied only to the parts that contain the quaternion components. Furthermore, special care must be taken when defining the dimension $\tilde{n}$, used in Eqs. (2.117) and (2.120), that decides the number of sigma points. When considering states (or measurements) evolving in a vector space, it corresponds to the sum of the size n of the state x_t and of the size of the noise variable w_t (or ν_t in the Correction step). More generally, it is defined as the sum of the number of degrees of freedom of the manifold where the state (or the measurements) evolves in and the size of the noise variable. In the case of the space of the unit quaternions, although four scalars are required to actually represent a quaternion, the number of degrees of freedom of the rotations, i.e. of the group SO(3), is three. Hence $\tilde{n}$ evaluates to $3 + n_q$ (or $3 + n_\nu$ in the Correction step).

2.4 Kalman smoothing

Thus far, I considered the problem of estimating the filtering distribution $p(x_t|z_{1:t})$ of the state x_t given the current and previous measurements $z_{1:t} = \{z_1, \cdots, z_t\}$. The presented algorithms, under the name of Kalman *filter* and associated variants, provide an approximation of the filtering distribution in terms of a Gaussian distribution that can be updated recursively (and online) once a new measurement z_t is available.

However, sometimes it is of interest to estimate the distribution of the state x_t given all the available measurements $z_{1:T} = \{z_1, \cdots, z_T\}$, known as the *smoothing distribution*:

$$p(x_t|z_{1:T}). \tag{2.123}$$

Estimating the smoothing distribution can be useful in those cases in which there is the necessity to estimate the state of the system *offline* in order to provide a ground truth signal for evaluation purposes or for training purposes (e.g. to train a neural network). The smoothing distribution provides information on the state at time t using all the measurements, i.e. also the future measurements, hence resulting in more precise estimates than using a Bayesian filtering approach (Särkkä, 2013).

As for the filtering distribution, there exists a branch of the Bayesian theory, termed Bayesian smoothing, that formalizes the problem of estimating the smoothing distribution. In the next sections, we will closely follow the theory of Bayesian and Kalman smoothing as presented in (Särkkä, 2013).

2.4.1 Basics of Bayesian smoothing

The smoothing distribution can be evaluated by resorting to the *Bayesian smoothing equation* (Kitagawa, 1994)

$$p(x_t|z_{1:T}) = p(x_t|z_{1:t}) \int \frac{p(x_{t+1}|x_t)p(x_{t+1}|z_{1:T})}{p(x_{t+1}|z_{1:t})} \mathrm{d}x_{t+1}, \tag{2.124}$$

where $p(x_t|z_{1:t})$ is the usual filtering distribution at time t, $p(x_{t+1}|x_t)$ is the transition probability distribution at time $t+1$, $p(x_{t+1}|z_{1:t})$ is the usual predictive distribution at time $t+1$ and, finally, $p(x_{t+1}|z_{1:T})$ is the smoothing distribution at time $t+1$. The smoothing acts recursively as the smoothing equation allows obtaining $p(x_t|z_{1:T})$ given the distribution $p(x_{t+1}|z_{1:T})$, i.e. it is a *backward* recursion (while the Bayesian filtering recursion is a forward recursion). Intuitively, the reason for using a backward recursion is simply that the initial condition of the recursion, i.e. the smoothing distribution at time T, $p(x_T|z_{1:T})$, corresponds to the filtering distribution at time T that is available after processing all the measurements using the Bayesian filtering equations.

In order to show that the Bayesian smoothing equation is as presented in Eq. (2.124), it suffices to consider the conditional probability $p(x_t|x_{t+1}, z_{1:T})$ which links two consecutive states x_t and x_{t+1} under the assumption that all the measurements are known. Using the fact that the states are a Markov sequence, as required by the Bayesian filtering theory, it holds that

$$p(x_t|x_{t+1}, z_{1:T}) = p(x_t|x_{t+1}, z_{1:t}). \tag{2.125}$$

The usual interpretation of the Markov property is that the state x_t is independent of the past states and measurements given the state x_{t-1}. In order to better explain the Eq. (2.125), it is worth mentioning that a more general interpretation (Särkkä, 2013) also assumes that the past state is independent of the future states and measurements given the present, i.e.:

$$p(x_{t-1}|x_{t:T}, z_{t:T}) = p(x_{t-1}|x_t). \tag{2.126}$$

Substituting $t-1$ with t and t with $t+1$, the property becomes

$$p(x_t|x_{t+1:T}, z_{t+1:T}) = p(x_t|x_{t+1}). \tag{2.127}$$

Finally, by conditioning both sides with the past measurements $z_{1:t}$, the following holds:

$$\begin{aligned} p(x_t|x_{t+1:T}, z_{t+1:T}, z_{1:t}) &= p(x_t|x_{t+1:T}, z_{1:T}) \\ &= p(x_t|x_{t+1}, z_{1:t}), \end{aligned} \tag{2.128}$$

which is the same as Eq. (2.125) if the set of future states $x_{t+1:T}$ is limited to the future state x_{t+1}.

Expanding Eq. (2.125) using the Bayes rule, we find that

$$\begin{aligned} p(x_t|x_{t+1}, z_{1:T}) &= p(x_t|x_{t+1}, z_{1:t}) \\ &= \frac{p(x_{t+1}|x_t, z_{1:t})p(x_t|z_{1:t})}{p(x_{t+1}|z_{1:t})} \\ &= \frac{p(x_{t+1}|x_t)p(x_t|z_{1:t})}{p(x_{t+1}|z_{1:t})}, \end{aligned} \tag{2.129}$$

where we used the usual Markov property for the first factor in the numerator. Using the definition of the conditional probability, the joint distribution of x_t and x_{t+1} given all the measurements is obtained as:

$$\begin{aligned} p(x_t, x_{t+1}|z_{1:T}) &= p(x_t|x_{t+1}, z_{1:T})p(x_{t+1}|z_{1:T}) \\ &= \frac{p(x_{t+1}|x_t)p(x_t|z_{1:t})p(x_{t+1}|z_{1:T})}{p(x_{t+1}|z_{1:t})}, \end{aligned} \tag{2.130}$$

where $p(x_{t+1}|z_{1:T})$ is the smoothing distribution at time $t+1$. Marginalization of the variable x_t via integration over x_{t+1} provides the Bayesian smoothing equation

$$\begin{aligned} p(x_t|z_{1:T}) &= \int p(x_t, x_{t+1}|z_{1:T})\mathrm{d}x_{t+1} \\ &= \int \frac{p(x_{t+1}|x_t)p(x_t|z_{1:t})p(x_{t+1}|z_{1:T})}{p(x_{t+1}|z_{1:t})}\mathrm{d}x_{t+1} \\ &= p(x_t|z_{1:t}) \int \frac{p(x_{t+1}|x_t)p(x_{t+1}|z_{1:T})}{p(x_{t+1}|z_{1:t})}\mathrm{d}x_{t+1}, \end{aligned} \tag{2.131}$$

where the filtering distribution at time t $p(x_t|z_{1:t})$ is not considered in the integration as it does not depend on the state x_{t+1}.

As in the case of the Bayesian filtering equations, the integral in Eq. (2.131) cannot be solved in closed form in general.

2.4.2 Basics of Kalman smoothing

Similarly to the Kalman filtering case, it is possible to tackle the Bayesian smoothing problem by approximating the smoothing distribution using a Gaussian distribution

$$p(x_t|z_{1:T}) \simeq \mathcal{N}(x; \mu_t^s, P_t^s)\Big|_{x=x_t}, \tag{2.132}$$

where μ_t^s and P_t^s are the mean vector and the covariance matrix that characterize the distribution. The approximation is possible if the following assumptions are made:

- the state x_t evolves according to a *discrete difference equation*

$$\begin{aligned} x_t &= f_t(x_{t-1}, w_{t-1}), \\ w_t &\sim \mathcal{N}(0, Q_t), \end{aligned} \tag{2.133}$$

  with f_t a generic nonlinear function and w_t a sequence of zero mean Gaussian noise samples with covariance Q_t and independent from the states;

- the joint probability distribution of consecutive states conditioned to the history of the measurements $p(x_t, x_{t+1}|z_{1:t})$ is Gaussian;
- the joint probability distribution of consecutive states conditioned to all the measurements $p(x_t, x_{t+1}|z_{1:T})$ is Gaussian;
- the prior $p(x_T^s)$ is Gaussian with mean μ_T^s and covariance P_T^s.

Under the aforementioned assumptions, it is possible to prove (Särkkä, 2013) that there exist closed form approximate solutions to the Bayesian backward recursive update rule of the smoothing distribution. The mathematical form of the update is reported in the following for the case of a linear motion model as in Eq. (2.9).

2.4.3 Linear Kalman smoothing

Linear Kalman smoothers assume that the function f_t is a linear function of the state x_t represented by the matrix F_t. Furthermore, the contribution of the noise samples w_t is additive. The resulting linear model is as follows:

$$\begin{aligned} x_t &= F_t x_{t-1} + w_{t-1}, \\ w_t &\sim \mathcal{N}(0, Q_t). \end{aligned} \tag{2.134}$$

In this configuration, the joint probabilities $p(x_{t+1}, x_t|z_{1:t})$ and $p(x_{t+1}, x_t|z_{1:T})$ are Gaussian distributions by construction as shown in the following.

Smoothing step (I)

Using the definition of conditional probability, it is possible to rewrite $p(x_t, x_{t+1}|z_{1:t})$ as

$$\begin{aligned} p(x_t, x_{t+1}|z_{1:t}) &= p(x_{t+1}|x_t, z_{1:t})p(x_t|z_{1:t}) \\ &= p(x_{t+1}|x_t)p(x_t|z_{1:t}), \end{aligned} \tag{2.135}$$

where we used the assumption that the states are a Markov sequence. Considering the state model in Eq. (2.134), it is evident that the transition probability distribution $p(x_{t+1}|x_t)$ is Gaussian and evaluates to

$$p(x_{t+1}|x_t) = \mathcal{N}(x; F_t x_t, Q_t)\Big|_{x=x_{t+1}}. \tag{2.136}$$

Furthermore, the second factor $p(x_t|z_{1:t})$ is simply the filtering distribution at time t that is Gaussian with mean μ_t and covariance P_t as per Eq. (2.21). The resulting joint distribution is Gaussian, as required, as it is the product of two Gaussian distributions:

$$\begin{aligned} p(x_t, x_{t+1}|z_{1:t}) &= \mathcal{N}\left(\begin{bmatrix} x_t \\ x_{t+1} \end{bmatrix}; \mu_t^{s'}, P_t^{s'}\right), \\ \mu_t^{s'} &= \begin{bmatrix} \mu_t \\ F_t\mu_t \end{bmatrix}, \\ P_t^{s'} &= \begin{bmatrix} P_t & P_tF_t^T \\ F_tP_t & F_tP_tF_t^T + Q_t \end{bmatrix}. \end{aligned} \tag{2.137}$$

Starting from the joint distribution in Eq. (2.137), it can be shown (Särkkä, 2013) that the distribution of the state at time x_t given the knowledge of the state at time x_{t+1} and the measurements $z_{1:t}$ is as follows

$$\begin{aligned} p(x_t|x_{t+1}, z_{1:t}) &= \mathcal{N}(x; \mu_t^{s''}, P_t^{s''})\Big|_{x=x_t}, \\ \mu_t^{s''} &= \mu_t + K_t(x_{t+1} - F_t\mu_t), \\ P_t^{s''} &= P_t - K_t(F_tP_tF_t^T + Q_t)K_t^T \\ K_t &= P_tF_t^T(F_tP_tF_t^T + Q_t)^{-1}. \end{aligned} \tag{2.138}$$

Using the Markov property in Eq. (2.125), it is clear that the above distribution is the same as $p(x_t|x_{t+1}, z_{1:T})$.

Smoothing step (II)

Using again the definition of conditional probability, the joint probability $p(x_{t+1}, x_t|z_{1:T})$ can be rewritten as

$$p(x_{t+1}, x_t|z_{1:T}) = p(x_t|x_{t+1}, z_{1:T})p(x_{t+1}|z_{1:T}), \tag{2.139}$$

where the first factor is Gaussian and has been evaluated in the Smoothing step (I) while the second factor is the smoothing distribution at time $t+1$

$$p(x_{t+1}|z_{1:T}) = \mathcal{N}(x; \mu_{t+1}^s, P_{t+1}^s)\Big|_{x=x_{t+1}}, \tag{2.140}$$

that is Gaussian by hypothesis.

The resulting joint distribution is Gaussian, as required, as it is the product of two Gaussian distributions:

$$\begin{aligned} p(x_{t+1}, x_t|z_{1:T}) &= \mathcal{N}\left(\begin{bmatrix} x_{t+1} \\ x_t \end{bmatrix}; \mu_t^{s'''}, P_t^{s'''}\right), \\ \mu_t^{s'''} &= \begin{bmatrix} \mu_{t+1}^s \\ \mu_t + K_t(\mu_{t+1}^s - F_t\mu_t) \end{bmatrix}, \\ P_t^{s'''} &= \begin{bmatrix} P_{t+1}^s & P_{t+1}^s K_t^T \\ K_t P_{t+1}^s & K_t P_{t+1}^s K_t^T + P_t^{s''} \end{bmatrix}. \end{aligned} \tag{2.141}$$

Starting from the joint distribution in Eq. (2.141), it can be shown (Särkkä, 2013) that the smoothing distribution $p(x_t|z_{1:T})$ is a Gaussian distribution

$$\begin{aligned} p(x_t|z_{1:T}) &= \mathcal{N}(x; \mu_t^s, P_t^s)\Big|_{x=x_t}, \\ \mu_t^s &= \mu_t + K_t(\mu_{t+1}^s - F_t\mu_t), \\ P_t^s &= P_t + K_t(P_{t+1}^s - F_tP_tF_t^T - Q_t)K_t^T. \end{aligned} \tag{2.142}$$

We notice that the solution provided by the Linear Kalman smoother is not approximate but rather exact, i.e. the resulting distribution is actually Gaussian. The Linear Kalman smoother is sometimes also called the Rauch-Tung-Striebel smoother (RTSS) (Rauch, 1963).

The resulting algorithm

Algorithm 10. *Kalman smoother (Rauch-Tung-Striebel smoother)*

Given a system described by the state model in Eq. (2.9) *and an initial prior*

$$p(x_T) \sim \mathcal{N}(\mu_T^s, P_T^s),$$

the smoothing distribution of the state is Gaussian and can be computed with the following Kalman smoothing step. The recursion starts from the last time step T, with $\mu_T^s = \mu_T$ and $P_T^s = P_T$ where μ_T and P_T are the mean and the covariance of the filtering distribution at time T.

- *Smoothing step:*

$$\begin{aligned} p(x_t|z_{1:T}) &= \mathcal{N}(x; \mu_t^s, P_t^s)\Big|_{x=x_t}, \\ \mu_t^s &= \mu_t + K_t(\mu_{t+1}^s - F_t\mu_t), \\ P_t^s &= P_t + K_t(P_{t+1}^s - F_tP_tF_t^T - Q_t)K_t^T, \\ K_t &= P_tF_t^T(F_tP_tF_t^T + Q_t)^{-1}, \end{aligned} \tag{2.143}$$

where μ_t and P_t are the mean and the covariance of the filtering distribution at time t and μ_{t+1}^s and P_t^s are the mean and the covariance of the smoothing distribution at time $t+1$.

2.5 Differentiable Kalman filtering

The implementation of a Kalman filtering-based algorithm for state estimation requires the definition of suitable motion and measurement models as per the Eqs. (2.7) and (2.8).

Motion and measurement models are usually first principles models, i.e. they are obtained form the combination of suitable physical laws that describe the system of interest. However, deriving such models is not always possible if the system is considerably complex or its working principles are not completely known. Recent advances in differential Kalman filtering (Haarnoja et al., 2016; Kloss et al., 2021) allow learning the required models from experimental data using a backpropagation-based training process.

In this section, I will briefly introduce the differentiable Kalman filtering paradigm that has been used in this Thesis to integrate tactile sensing within object pose tracking algorithms. The adopted mathematical formulation is similar to that presented in (Kloss et al., 2021) and (Lee et al., 2020).

2.5.1 Motion models using neural networks

The differentiable paradigm substitutes the function f_t in Eq. (2.7) with an alternative functions $\tilde{f}_t$ that might be in part specified by the user and in part modelled using one ore more neural networks. To better contextualize the definition of $\tilde{f}$, the following example is made.

Consider the state of a system evolving over the real line that is described in terms of its zero-th and first derivative $p_t \in \mathbb{R}$ and $v_t = \dot{p}_t \in \mathbb{R}$. The state of the system is given by $x_t = (p_t, v_t) \in \mathbb{R}^2$. A possible motion model is as follows:

$$\begin{aligned}
\begin{bmatrix} p_t \\ v_t \end{bmatrix} &= \tilde{f}_t(x_{t-1}, w_{t-1}) \\
&= \begin{bmatrix} p_{t-1} + v_{t-1}\Delta_t \\ v_{t-1} \end{bmatrix} + NN_f(x_{t-1}, v_{t-1}) + w_{t-1}, \\
w_t &\sim \mathcal{N}(0, Q_t), \\
Q_t &= NN_w(x_{t-1}).
\end{aligned} \tag{2.144}$$

The function $\tilde{f}$ contains a part that is specified by the user and indicates that p_t is the integral of v_t under the assumption that Δ_t is the interval of time between consecutive steps. This part is combined with the output of a neural network NN_f which uses the previous state x_{t-1} as input and whose parameters are to be optimized using a training procedure. The network models those parts of the system that are not known a priori. The overall model is perturbed with additive Gaussian noise with zero mean and covariance Q_t. The covariance matrix itself can be parametrized using a neural network NN_w depending on the state at time $t-1$.

2.5.2 Measurement models using neural networks

With respect to the measurement process, a commonly adopted strategy (Kloss et al., 2021; Lee et al., 2020) is to train a neural network NN_z which maps the output of the sensing system z_t to the state x_t or a partition of the state Hx_t where H is as follows:

$$H = \begin{bmatrix} \vdots \\ h_j^T \\ \vdots \end{bmatrix}. \tag{2.145}$$

Here, $h_j = e_i$ if the j-th component of the measurement corresponds to the i-th component of the state x_t, e_i is the i-th vector of the canonical base of R^n and n is the size of the state x_t. As a result, $\tilde{z}_t = NN_z(z_t)$ is used as the actual measurement, instead of z_t, and the measurement model is simply

$$\begin{aligned}\tilde{z}_t &= Hx_t + \nu_t,\\ \nu_t &\sim \mathcal{N}(0, R_t),\\ R_t &= NN_\nu(x_t, z_t).\end{aligned} \tag{2.146}$$

The measurement noise covariance can be parametrized using a neural network NN_ν depending on both the current state and measurement.

An alternative strategy, that I adopted in this Thesis, consists in using a neural network also to predict the measurement given the state. Similarly to the function $\tilde{f}_t$, the function h_t in Eq. (2.8) is substituted with an alternative function $\tilde{h}_t$ that might be in part specified by the user and in part modelled using a neural network. An example of definition of $\tilde{h}$ for the system indicated above could be as follows:

$$\begin{aligned}\tilde{z}_t &= \tilde{h}_t(x_t, \nu_t)\\ &= g(x_t) + NN_h(x_t) + \nu_t,\\ \nu_t &\sim \mathcal{N}(0, R_t),\\ R_t &= NN_\nu(x_t, z_t).\end{aligned} \tag{2.147}$$

Here, $g(x_t)$ models any prior information that is available on the measurement process while the neural network NN_h models those part of the measurement process that are not known and could be learned from data. Also in this configuration, a network NN_z is used to map the real measurements z_t to the same space of the predicted measurements, making possible the evaluation of the Kalman innovation term.

2.5.3 Training protocol

Once the motion and measurement models, suitably augmented using neural networks, have been chosen, the networks architecture can be trained using appropriate data.

Specifically, it is assumed that the following are available:

- a set X of N sequences $\{x^{gt}_{t,i}\}$ of the full state of the system;
- a set Y of N sequences $\{z_{t,i}\}$ of measurements collected from the sensing system and that are compatible with the ground truth states.

The index i indicates the specific sequence among those available. The sequences can be acquired from the real system under investigation or simulated, whenever possible, to reduce the time and effort required for data collection.

Each sequence belonging to X and Y is divided in subsequences of length $T_s < T$ with T the total length of each sequence. Each subsequence starts at $t = t_{0,s}$. Given a batch size $B < N$ and a selection X_B of B sequences among those available, a set of B filters, that share the same parameters for the networks NN_f, NN_w, NN_h and NN_ν, is initialized by sampling from the normal distribution

$$\mathcal{N}(x_t; x_{t,i}^{gt}, P_0), \tag{2.148}$$

with $i \in X_B$, $t = t_{0,s}$ for a given subsequence s and P_0 the initial state covariance provided by the user. Then, the state of each filter is updated using the measurements from Y for T_s steps via suitable Kalman prediction and correction steps.

Given the nonlinearity of the neural networks used in the motion and measurement models, the Extended or Unscented Kalman prediction and correction steps are required. In case the Extended Kalman filter is used, the requirement to evaluate the Jacobians of the functions $\tilde{f}_t$ ad $\tilde{h}_t$ is satisfied by resorting to modern automatic differentiation techniques as commonly used in Machine Learning software frameworks.

Given a subsequence s, the training loss can be evaluated as follows (Lee et al., 2020):

$$L_s = \frac{1}{T_s B n} \sum_{i \in X_B} \left(\sum_{t=t_{0,s}+1}^{t_{0,s}+T_s} (x_{t,i}^{gt} - \mu_{t,i})^T (x_{t,i}^{gt} - \mu_{t,i}^T) \right), \tag{2.149}$$

where n is the size of the state x_t and $\mu_{t,i}$ is the corrected mean from the i-th filter at time t. Remarkably, the corrected mean depends on the Kalman gain K_t which in turn depends on the process and measurement covariance matrices Q_t and R_t. Hence, the associated networks N_w and N_ν are also taken into account in the evaluation of the training loss.

The loss L_s is used to optimize the parameters of all the networks resulting in the so-called backpropagation through time (BPTT) mechanism (Mozer, 1989). Within each training epoch, the optimization procedure is repeated for all the sub-sequences within a sequence and for all the sets of B sequences among those available in X. After that, the training epoch is completed and a new one is processed until all the epochs have been processed.

Once the training process is complete, the resulting motion and measurement models can be used as part of the Kalman filter to process new measurements, not seen at training time, and obtain the associated estimates of the state x_t.

Chapter 3

Object Pose and Velocity Tracking

Object pose tracking is the computer vision and robotic task that estimates the 3D position and the 3D orientation of an object of interest using images as primary input. Additional sensory modalities, e.g. touch sensing, can be used in alternative or in combination with visual inputs. Solving this task is fundamental for the development of autonomous robotic platforms that could identify the pose of objects in the surrounding environment and execute assigned tasks accordingly. Examples of these tasks are robot navigation in an environment with obstacles, grasping and manipulation of objects.

The pose tracking task can be extended to tracking the linear and angular velocity of the object, in addition to the pose. The information on the velocity is sometimes required if the robot interacts with moving objects in highly dynamic environments. Furthermore, robot motion control architectures often require the velocity as feedback to close the control loop.

In this chapter, I will first provide, in Sec. 3.1, a formal definition of the pose and the velocity of an object and of the object pose and velocity tracking tasks. In Sec. 3.2, I will concisely define the concept of triangle mesh representation of an object, that will be used to describe the 3D geometry of the object in the following sections. In Sec. 3.3, I will contextualize the concepts of object pose and velocity in a scenario where images are used as inputs to the tracking algorithm. Next, in Sec. 3.4, I will describe the most commonly adopted metrics for evaluating object pose and velocity tracking algorithms in the computer vision and tracking communities. The chapter concludes with Secs. 3.5, 3.6 and 3.7 where I propose an overview of object pose tracking methods from the computer vision and robotics literature. The presented methods range from classical filtering or optimization-based approaches to more recent ones leveraging deep learning or a combination of deep learning and filtering or optimization.

3.1 Theory and problem formulation

The object pose tracking problem is defined as the task of updating the position and the orientation of a rigid object $\mathcal{O}$ of interest given an initial condition and a set of indirect measurements of these quantities. In a robotic context, the indirect measurements are usually provided by vision sensors, e.g. RGB-D cameras, Lidar systems, or tactile sensors, i.e. sensors that provide information on the contact between the object and the robot end-effector and the associated magnitude.

In the following, we first provide a formal definition of the pose of an object and of the associated object pose tracking task. Next, the definition of the velocity of an object is presented and the object pose tracking task is extended to the task of tracking both the pose and the velocity of the object.

The mathematical notation and the concepts presented in this Section are adapted from (Lynch and Park, 2017).

3.1.1 Definition of object pose

More formally, the position and the orientation of a rigid object $\mathcal{O}$ are represented as the *relative pose* of a reference frame $\{B\}$ attached to the object body

$$\{B\} = \{O_b; i_b, j_b, k_b\}, \tag{3.1}$$

with origin O_b and axes i_b, j_b and k_b, with respect to a fixed frame $\{I\}$

$$\{I\} = \{O_i; i_i, j_i, k_i\}, \tag{3.2}$$

with origin O_i and axes i_i, j_i and k_i (see Fig. 3.1).

Numerically, the pose is represented using a 4×4 matrix T

$$T = \begin{bmatrix} R & p \\ 0^T & 1 \end{bmatrix}, \tag{3.3}$$

where $R \in \mathrm{SO}(3)$ is a rotation matrix describing the relative orientation between the frames and $p \in \mathbb{R}^3$ is the vector connecting the origin of the two frames.

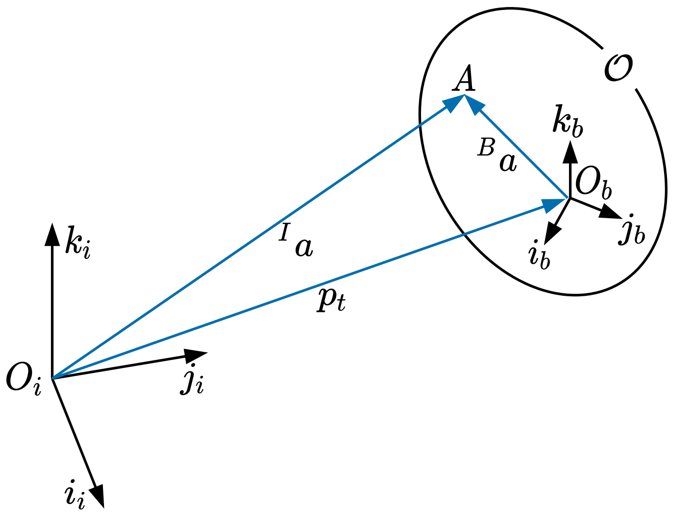


Figure 3.1 Representation of the fixed frame $\{I\}$ and object frame $\{B\}$. The vector p_t connects the origin of the two reference frames. A generic point A can be localized with respect to both reference frames using the vectors ${}^{I}a$ and ${}^{B}a$.

The vector p can be expressed as

$$p = \overline{O_i O_b} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}, \tag{3.4}$$

where p_x, p_y and p_z are simply the coordinates of the origin O_b expressed in the frame $\{I\}$ (see Fig. 3.1).

If i_b, j_b and k_b are considered as vectors of coordinates in $\mathbb{R}^3$, expressed in the frame $\{I\}$, then the rotation matrix R assumes the form

$$R = \begin{bmatrix} | & | & | \\ i_b & j_b & k_b \\ | & | & | \end{bmatrix}. \tag{3.5}$$

Given a point A, the matrix T can be also interpreted as a linear transformation that converts the coordinates ${}^{B}a \in \mathbb{R}^3$ of a vector $\overline{O_b A}$ expressed in the frame $\{B\}$ to the coordinates ${}^{I}a$ of the vector $\overline{O_iA}$ expressed in the frame $\{I\}$:

$$\begin{aligned}\begin{bmatrix} {}^{I}a \\ 1 \end{bmatrix} &= T \begin{bmatrix} {}^{B}a \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} R\,{}^{B}a + p \\ 1 \end{bmatrix}.\end{aligned} \tag{3.6}$$

The relationship between ${}^{I}a$ and ${}^{B}a$ is also represented in Fig. 3.1.

The matrix T can be inverted as follows:

$$T^{-1} = \begin{bmatrix} R^T & -R^T p \\ 0^T & 1 \end{bmatrix}. \tag{3.7}$$

If the pose of the frame $\{B\}$ is not stationary but changes over time, the transformation T becomes a function of the time:

$$T_t = \begin{bmatrix} R_t & p_t \\ 0^T & 1 \end{bmatrix}, \tag{3.8}$$

where p_t and R_t are the position vector and the rotation matrix at time t, respectively.

3.1.2 A definition of object pose tracking

Using the above notation, the task of tracking an object pose, also termed 6D object pose tracking, can be defined as the process of updating the pose of the object T_t for each time t given:

- the pose T_{t-1} at previous time $t-1$;
- a new measurement $z_t = z_t(T_t)$ that depends on the pose T_t.

The tracking process is initiated by the specification of an initial pose T_0 provided by the user. The dependency on these terms can be made explicit by indicating the pose as $T_t(z_t(T_t), T_{t-1}, T_0)$.

We remark that the 6D object pose tracking task is related to but fundamentally different from the computer vision task known as "6D object pose estimation". The latter task is is regarded with the estimation of the pose of the object T_t at time t usually from an RGB, or RGB-D, image I_t. An object pose estimation method does not use the information of the pose T_{t-1} at time $t-1$. In other words, it solves the

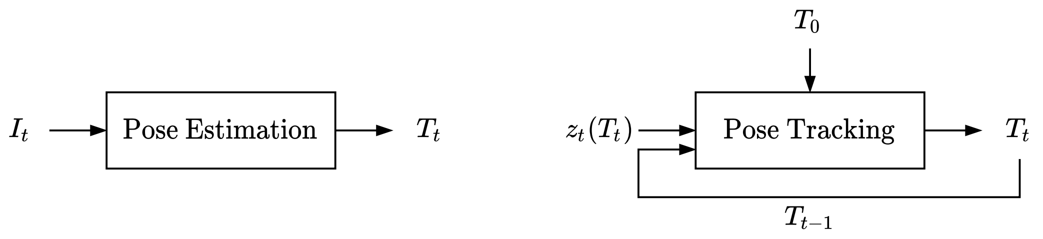


Figure 3.2 A scheme representing the difference between the object pose estimation and object pose tracking tasks.

estimation problem by considering each input image independently. Furthermore, it does not need an initial condition. The dependency on the input image I_t can be made explicit by indicating the pose as $T_t(I_t)$. The difference between the two tasks is schematized in Fig. 3.2.

The object pose estimation and pose tracking tasks are evenly important. Pose estimation methods are typically used to identify the pose of one or more objects in a scene and execute robotic tasks such as planning the trajectory of the robot end-effector in a cluttered scene or finding grasping candidates. Furthermore, they can be used to start, at any time, the pose tracking process as the estimated pose $T_t(I_t)$ can be employed as an initial condition T_0 for a pose tracking method.

On the other hand, there exist scenarios and tasks where a pose tracking methodology might be more suited. These scenarios include for example:

1. object manipulation requiring continuous and well-behaved state feedback;
2. interaction with objects characterized by moderate-to-fast motions;
3. integration of a priori information on the motion of the object.

In the following I will identify possible advantages in using a pose tracking methodology when dealing with the scenarios indicated above.

Scenario 1

Object pose tracking algorithms combine the history of the object pose with the incoming observations from the sensors. Using past information is reasonable as consecutive object poses are not independent but rather linked by differential constraints, i.e. the pose T_t depends on the pose T_{t-1} at time $t-1$ and the velocity of the object in between.

By taking into account these temporal constraints, these methods are able to produce continuous and well-behaved pose signals T_t that are suitable for object manipulation tasks involving closed loop control.

Scenario 2

The pose of an object characterized by moderate-to-fast motions is in general very difficult to estimate. In this scenario, 6D object pose estimation methods could be used if the execution time of the algorithm is compatible with the velocity of the object. However, fast motions usually produce blurred images I_t that could make the estimates $T_t(I_t)$ much less reliable.

If external sources of information about the velocity of the object are available, e.g. from an Inertial Measurement Unit (IMU) or using blur robust Optical Flow (OF) methods, these can be naturally integrated as measurements z_t within object pose tracking methods, making them preferable to object pose estimation methods.

Scenario 3

Object pose tracking methods usually provide explicit mechanisms to incorporate a priori information on the motion of the object. As an example, Kalman filters (see Chapter 2) require the specification of a suitable motion model.

Whenever available, using prior information in the form of a motion model is beneficial for integrating the information encoded in the measurements and reducing the effect of the measurement noise. Furthermore, motion models can be used to predict the pose of the object in case the measurements are not available for short periods of time.

3.1.3 Definition of object velocity

As discussed in the previous section, the velocity of the object plays an important role in the process of tracking the pose of a rigid object. It is possible to extend the task of object pose tracking to that of tracking the pose and the velocity of the object.

Definition of the angular velocity in 3D space

In order to properly define this task, it is necessary to formally define the concept of velocity of a rigid object. To this end, it is convenient to first define the concept of velocity associated to the orientation of the object. For the sake of clarity, we omit the subscript t indicating the time dependency of the rotation matrix R_t.

By taking the time derivative of the rotation matrix R the following holds:

$$\begin{aligned}\dot{R} &= \frac{\mathrm{d}}{\mathrm{d}t}(R) \\ &= \begin{bmatrix} | & | & | \\ \dot{i}_b & \dot{j}_b & \dot{k}_b \\ | & | & | \end{bmatrix},\end{aligned} \tag{3.9}$$

where we expand R according to Eq. (3.5) and i_b, j_b and k_b are the unit vectors of the axes of the frame $\{B\}$ expressed in the frame $\{I\}$.

By assuming that the frame $\{B\}$ rotates with an angular velocity ω, the time derivative of the unit vectors can be expressed using the Poisson formula (Miele, 2016) as follows:

$$\begin{bmatrix} | & | & | \\ \dot{i}_b & \dot{j}_b & \dot{k}_b \\ | & | & | \end{bmatrix} = \begin{bmatrix} | & | & | \\ \omega \times i_b & \omega \times j_b & \omega \times k_b \\ | & | & | \end{bmatrix}. \tag{3.10}$$

From a physical point of view, the derivative of a unit vector is perpendicular to the plane identified by the vector itself and the angular velocity. Given that the vectors i_b, j_b and k_b are expressed in the frame $\{I\}$, we assume that the angular velocity is expressed in the same frame too.

It is recalled that the cross product $a \times b$ between two vectors $a = (a_x, a_y, a_z) \in \mathbb{R}^3$ and $b \in \mathbb{R}^3$ can be expressed as the product between a skew-symmetric matrix $\hat{a}$

$$\hat{a} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \in \mathbb{R}^{3\times 3}, \tag{3.11}$$

and the vector b, i.e.

$$a \times b = \hat{a}b. \tag{3.12}$$

The operator $\hat{}$ is sometimes called the *hat* operator. Its inverse operator, the *vee* operator $^\vee$, is defined such that

$$(\hat{a})^\vee = a \in \mathbb{R}^3. \tag{3.13}$$

Taking into account the operators defined above, the time derivative of R becomes

$$\begin{aligned}\dot{R} &= \begin{bmatrix} | & | & | \\ \hat{\omega} i_b & \hat{\omega} j_b & \hat{\omega} k_b \\ | & | & | \end{bmatrix} \\ &= \hat{\omega} \begin{bmatrix} | & | & | \\ i_b & j_b & k_b \\ | & | & | \end{bmatrix} \\ &= \hat{\omega} R.\end{aligned} \tag{3.14}$$

The angular velocity can be expressed as

$$\hat{\omega} = \dot{R}R^{-1} = \dot{R}R^T \in \mathfrak{so}(3), \tag{3.15}$$

or, using the vee operator, as

$$\omega = (\dot{R}R^T)^\vee \in \mathbb{R}^3. \tag{3.16}$$

As indicated above, the hat of the angular velocity belongs to the Lie algebra $\mathfrak{so}(3)$ of the group SO(3), both defined in Sec. 2.3. In that section, the Lie algebra of SO(3) was used to represent a given rotation R as the "angular velocity" that the frame $\{B\}$ has to travel with in one unit time to move from the identity rotation to R. In this section, instead, it is used to represent the *instantaneous axis of rotation* $\frac{\omega}{\|\omega\|} \in \mathbb{R}^3$ the object is rotating around and the rate of rotation $\|\omega\| \in \mathbb{R}$ around that axis.

The exp() and log() maps which transform a rotation matrix $R \in \mathrm{SO}(3)$ to its vector representation $\omega \in \mathbb{R}^3$ in the Lie algebra and viceversa are defined as follows:

$$R = \exp(\omega) := I_3 + \sin(\|\omega\|)\frac{\hat{\omega}}{\|\omega\|} + (1 - \cos(\|\omega\|))\left(\frac{\hat{\omega}}{\|\omega\|}\right)^2, \ \ \|\omega\| \neq 0, \tag{3.17}$$

$$\omega = \log(R) := n\theta, \ \ \theta \neq 0, \tag{3.18}$$

where (n, θ) is a proper axis-angle representation of the rotation matrix R. In case $\|\omega\| = 0$, the exponential evaluates to the identity rotation, $R = I_3$. Similarly, in the case $\theta = 0$, the axis n is not defined and the logarithm evaluates to the null vector $\omega = 0$.

The final expression of the angular velocity at time t is as follows:

$$\omega_t = (\dot{R}_t R_t^T)^\vee \in \mathbb{R}^3. \tag{3.19}$$

Definition of the spatial and mixed velocities in 6D space

Similarly to the rotation R_t, that belongs to the group SO(3), the full 6D pose of the rigid object T_t belongs to the group SE(3) defined as

$$\mathrm{SE}(3) = \left\{ \begin{bmatrix} R & p \\ 0^T & 1 \end{bmatrix} \mid p \in \mathbb{R}^3,\ \ R \in \mathrm{SO}(3) \right\}. \tag{3.20}$$

In this respect, it is reasonable to extend to SE(3) the property that post-multiplying $\dot{R}$ by R^{-1} results in a matrix representation of the angular velocity expressed in the frame $\{I\}$. For the sake of clarity, we omit the subscript t indicating the time dependency of the pose T_t.

The product $\dot{T}T^{-1}$, with T^{-1} as per Eq. (3.7), expands to

$$\begin{aligned} \dot{T}T^{-1} &= \begin{bmatrix} \dot{R} & \dot{p} \\ 0^T & 0 \end{bmatrix} \begin{bmatrix} R^T & -R^T p \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \dot{R}R^T & \dot{p} - \dot{R}R^T p \\ 0^T & 0 \end{bmatrix} \\ &= \begin{bmatrix} \hat{\omega} & \dot{p} + \hat{\omega}(-p) \\ 0^T & 0 \end{bmatrix} \\ &= \begin{bmatrix} \hat{\omega} & \dot{p} + \omega \times (-p) \\ 0^T & 0 \end{bmatrix} \\ &= \begin{bmatrix} \hat{\omega} & v \\ 0^T & 0 \end{bmatrix}. \end{aligned} \tag{3.21}$$

The resulting expression is called the *spatial velocity* $\mathcal{V}$ of the object

$$\hat{\mathcal{V}} = \dot{T}T^{-1} = \begin{bmatrix} \hat{\omega} & v \\ 0^T & 0 \end{bmatrix} \in \mathfrak{se}(3), \tag{3.22}$$

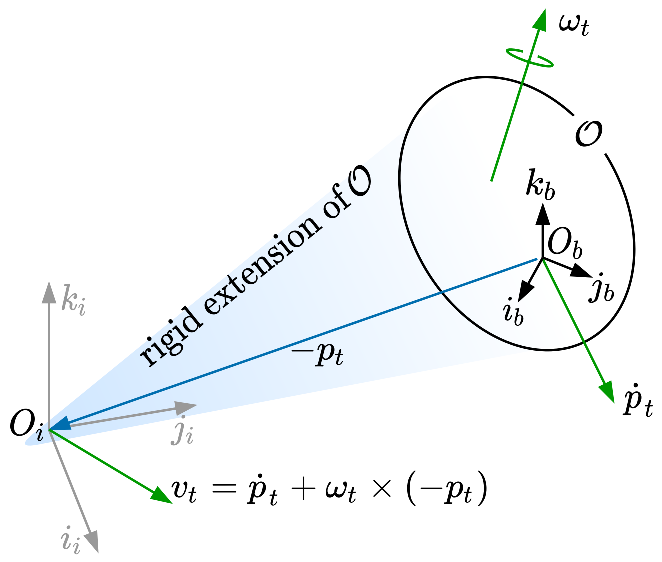


Figure 3.3 Representation of the angular velocity ω_t, the velocity $\dot{p}_t$ of the point O_b and the velocity v_t of a point belonging to a rigid extension of the object $\mathcal{O}$ and instantaneously coincident with O_i.

and, similarly to the angular velocity, it belongs to the Lie algebra $\mathfrak{se}(3)$ of the group SE(3). The associated vector representation in $\mathbb{R}^6$ is given by

$$\mathcal{V} = (\dot{T}T^{-1})^{\vee} = \begin{bmatrix} v \\ \omega \end{bmatrix} \in \mathbb{R}^6, \tag{3.23}$$

where we used an equivalent vee operator for $\mathfrak{se}(3)$.

Remarkably, the velocity v is not simply the velocity v_{O_b} of the origin O_b of the frame $\{B\}$ expressed in the frame $\{I\}$. That velocity would be instead $v_{O_b} = \dot{p}$.

From a physical point of view, $v = \dot{p} + \omega \times (-p)$ represents the velocity, expressed in the frame $\{I\}$, of a point belonging to the rigid body $\mathcal{O}$ that is instantaneously coincident with the origin O_i. In other words, it is the velocity of a *rigid extension* of the rigid body the extends up to the origin of the fixed frame (see Fig. 3.3). This interpretation comes from the classical formula linking the linear velocities of two points A, B belonging to the rigid object $\mathcal{O}$, i.e.

$$v_B = v_A + \omega \times \overline{AB}, \tag{3.24}$$

with $\overline{AB}$ the vector from A to B. In our case, $A = O_b$, $B = O_i$ hence $\overline{AB} = \overline{O_bO_i} = -p$, according to the definition of p in Eq. (3.4). As previously noticed, $v_{O_b} = \dot{p}$. The resulting velocity $v_B = v_{O_i}$ is equal to v, as expected.

Sometimes, it is required to work with a different definition of the 6D velocity of the object

$$\mathcal{V}^m = \begin{bmatrix} v^m \\ \omega \end{bmatrix} = \begin{bmatrix} \dot{p} \\ \omega \end{bmatrix} \in \mathbb{R}^6, \tag{3.25}$$

that is termed *mixed velocity* of the object. We remark that it does not exist any element in the Lie algebra $\mathfrak{se}(3)$ such that the application of the vee operator on it corresponds to the mixed velocity $\mathcal{V}^m$ expressed in the frame $\{I\}$. In this Thesis, we used both formulations of the velocity in order to accomplish different tasks.

The final expressions of the 6D velocities of the rigid body at time t are as follows:

$$\mathcal{V}_t = (\dot{T}_t T_t^{-1})^\vee = \begin{bmatrix} v_t = \dot{p}_t + \omega_t \times (-p_t) \\ \omega_t \end{bmatrix} \in \mathbb{R}^6, \tag{3.26}$$

$$\mathcal{V}_t^m = \begin{bmatrix} v_t^m = \dot{p}_t \\ \omega_t \end{bmatrix} \in \mathbb{R}^6. \tag{3.27}$$

3.1.4 A definition of object pose and velocity tracking

We define the task of object pose and velocity tracking as the process of updating the pose T_t and the mixed velocity $\mathcal{V}_t^m$ of the object for each time t given:

- the pose T_{t-1} at previous time $t-1$;
- the mixed velocity $\mathcal{V}_{t-1}^m$ at previous time $t-1$;
- a new measurement $z_t = z_t(T_t, \mathcal{V}_t^m)$ that depends on the pose T_t or the velocity $\mathcal{V}_t^m$ or both.

The tracking process is initiated by the specification of an initial pose T_0 and an initial velocity $\mathcal{V}_0^m$ provided by the user. The dependency on these terms can be made explicit by indicating the pose as $T_t(z_t(T_t, \mathcal{V}_t^m), T_{t-1}, \mathcal{V}_{t-1}^m, T_0, \mathcal{V}_0^m)$.

3.2 Three-dimensional object representation using triangle meshes

Before presenting the concepts of the following sections, it is useful to define the representation that we will use to model the shape and the geometry of the object $\mathcal{O}$ throughout the Thesis, namely the polygon mesh representation (Smith, 2006).

In 3D computer graphics, a polygon mesh is a collection of vertices, edges and faces that describe the shape of a given object. Each vertex is a point on the surface of the object while an edge represents a connection between two vertices. Given a list of edges, a face is then described as a closed set of a certain number of edges. The faces typically consist of triangles, i.e. closed sets of three edges, although more general convex polygons can be used. If triangular faces are used, the polygon mesh is usually referred to as a triangle mesh. In this Thesis, we only consider triangle meshes.

In order to actually describe a triangle mesh, it is required to provide a numerical representation of the above mentioned concepts. The representation of the edges and of the faces is typically very dependant on the specific file format used to store the polygon mesh data. For this reason, it is not discussed.

As regards the set of vertices, they consist of a set of points A_j sampled on the surface of the object $\mathcal{O}$. For each vertex, several properties are considered. The most important is the position of the vertex in the object frame $\{B\}$. It is usually described using a vector $\overline{O_b A_j}$ from the origin of the frame $\{B\}$ to the point A_j corresponding to the vertex. The vector is represented by means of its coordinates ${}^{B}a_j$ expressed in the frame $\{B\}$.

Other possible properties associated to each vertex are the normals, the colors and the texture coordinates.

Each normal describes the local curvature of the object at the vertex position and defines the direction that points out from the surface of the object at the same position.

As regards the texture, it refers to a broad set of properties of the surface of the object, such as the color, the reflectivity, the transparency or the tactile texture. However, in most cases, the texture description of a mesh only takes into account the colors of the surface. In this respect, it is possible to assign a color to each vertex in the mesh. As an alternative, it is possible to use the so-called texture coordinates which represent, for each vertex, the 2D coordinates of a digital image, termed texture map. The texture map contains the colors to be assigned to each vertex. If the texture coordinates are available, the mesh is usually referred to as a textured mesh.

In this Thesis, we will mostly use the concepts of triangle mesh and vertex positions. Nonetheless, the concepts of vertex normals and of textured mesh will be useful to describe some of the state-of-the-art methods for object pose tracking in Secs. 3.5, 3.6 and 3.7.

3.3 Object pose and velocity from the camera perspective

Most algorithms for 6D object pose and velocity tracking assume that the measurements of interest z_t are extracted from a stream of RGB images I_t and depth images D_t. Two of the methods presented in this Thesis share the same assumption. For this reason, this section provides a mathematical model of the camera and highlights how it connects with the concepts of object pose and velocity presented in the previous sections.

In the following, we assume that the camera is attached to the fixed frame $\{I\}$. Although the camera might move freely in space, we assume that its pose is fixed and that the object $\mathcal{O}$ undergoes an additional apparent motion that reflects the actual motion of the camera. This choice is sound as long as we make the reasonable assumption that the task at hand requires to track the relative motion between the camera and the object.

3.3.1 Connection between the object pose and the object point cloud

Each image I_t is made of pixels whose coordinates in the image plane are usually indicated with a tuple (u, v). The actual content of the image I_t consists of the brightness values of the three RGB channels for each pixel. If an RGB-D camera is adopted, an additional image D_t, called depth image, is provided that contains the distance in meters from the origin of the camera to the surrounding environment for each pixel along the axis k_i (see Fig. 3.4).

The depth image allows associating the coordinates (u, v) of a pixel to the 3D coordinates of the point P_{uv}, corresponding to that pixel, expressed in the camera frame. Given that the camera frame is coincident with the frame $\{I\}$, these coordinates, indicated as ${}^{I}p_{\mathrm{uv}}$, are that of the vector $\overline{O_i P_{\mathrm{uv}}}$ expressed in the frame $\{I\}$.

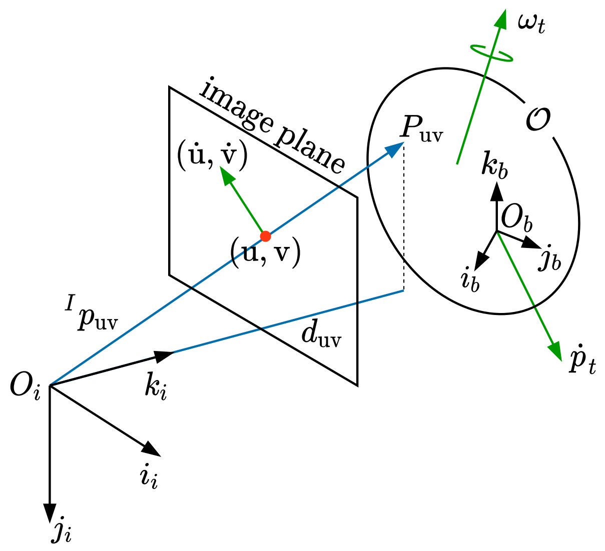


Figure 3.4 Representation of the vector ${}^I p_{\text{uv}}$ from the origin O_i to the point P_{uv} belonging to the object $\mathcal{O}$, the depth d_{uv}, i.e. the z coordinate of the vector, and the projection of the vector onto the image plane, corresponding to the pixel coordinates (u, v). The object linear velocity $\dot{p}_t$ and angular velocity ω_t, and the corresponding velocity of the pixel location $(\dot{\text{u}}, \dot{\text{v}})$ are also represented.

The pinhole camera model (Ma et al., 2004) provides the expression of these coordinates as a function of the pixel coordinates (u, v) and the depth of that pixel $d_{\text{uv}} \in D_t$:

$$ {}^I p_{\text{uv}} = {}^I p_{\text{uv}}(d_{\text{uv}}) = \begin{bmatrix} p_{\text{uv},x} \\ p_{\text{uv},y} \\ p_{\text{uv},z} \end{bmatrix} = \begin{bmatrix} \frac{(\text{u}-c_x)}{f_x} d_{\text{uv}} \\ \frac{(\text{v}-c_y)}{f_y} d_{\text{uv}} \\ d_{\text{uv}} \end{bmatrix} \in \mathbb{R}^3. \tag{3.28} $$

Here, f_x and f_y are the camera focal lengths while c_x and c_y are the coordinates of the camera principal point. The relationship between the pixel coordinates (u, v) and the vector ${}^I p_{\text{uv}}$ is represented in Fig. 3.4.

Given a set of pixels Ω in the image that contain the object of interest $\mathcal{O}$, we define the *partial* point cloud of the object $\mathcal{O}$ given the depth image D_t as the set of point coordinates

$$ C(\Omega, D_t) := \{ p_{\text{uv}}(d_{\text{uv}}) \quad \forall (\text{u}, \text{v}) \in \Omega \mid d_{\text{uv}} \in D_t \} . \tag{3.29} $$

The knowledge of the pose of the object T_t allows making a prediction of the object point cloud. Suppose that a set of points $\{A_j\}$ are sampled on the surface of the object $\partial\mathcal{O}$ and that the vectors $\overline{O_b A_j}$ from the origin of the frame $\{B\}$ to each point are available. These vectors are represented using the coordinates ${}^{B}a_j(A_j) \in \mathbb{R}^3$ that are easily obtained from a triangle mesh representation of the object, as described in Sec. 3.2. Using the definition of T_t as a transformation matrix, as in Eq. (3.6), it is possible to express the coordinates ${}^{I}a_j(A_j, T_t) \in \mathbb{R}^3$ of the vectors $\overline{O_i A_j}$ as follows:

$$\begin{bmatrix} {}^{I}a_j \\ 1 \end{bmatrix} = T_t \begin{bmatrix} {}^{B}a_j \\ 1 \end{bmatrix}. \tag{3.30}$$

We define the *predicted* point cloud of the object $\mathcal{O}$ given the object pose T_t as the set of point coordinates

$$C(\{A_j\}, T_t) \coloneqq \left\{ {}^{I}a_j(A_j, T_t) \quad \forall A_j \right\}. \tag{3.31}$$

The definition of the partial and predicted point clouds of the object and their comparison make it possible to relate the object pose T_t with the depth image D_t and is fundamental for the development of object pose tracking algorithms from depth images.

3.3.2 Connection between the object velocity and the object optical flow

Similarly to the object pose, it is possible to relate the velocity of the pixels locations

$$\begin{bmatrix} \dot{\mathrm{u}} \\ \dot{\mathrm{v}} \end{bmatrix} = \frac{d}{\mathrm{d}t} \begin{bmatrix} u \\ v \end{bmatrix} \in \mathbb{R}^2 \tag{3.32}$$

(see Fig. 3.4) with the object spatial velocity $\mathcal{V}$. To this end, we first recall the concept of optical flow.

Given consecutive images I_{t-1} and I_t, the optical flow can be defined as the set of displacements

$$\begin{bmatrix} \Delta_{\mathrm{u}} \\ \Delta_{\mathrm{v}} \end{bmatrix} \in \mathbb{R}^2 \tag{3.33}$$

a given pixel $(\mathrm{u}, \mathrm{v}) \in I_{t-1}$ has travelled in order to reach its new location $(\mathrm{u} + \Delta_{\mathrm{u}}, \mathrm{v} + \Delta\mathrm{v}) \in I_t$. We assume that the displacements can be expressed as

$$\begin{aligned} \Delta_{\mathrm{u}} &= \dot{\mathrm{u}} \Delta t, \\ \Delta_{\mathrm{v}} &= \dot{\mathrm{v}} \Delta t, \end{aligned} \tag{3.34}$$

where Δ_t is the time elapsed between the instants of time $t-1$ and t. The optical flow can be easily evaluated in real-time using suitable algorithms (NVIDIA, 2021) given two consecutive frames I_{t-1} and I_t.

Given a set of pixels Ω in the image that contain the object of interest, we define the optical flow associated to the object $\mathcal{O}$ given the images I_t and I_{t-1} as the set of displacements

$$F(\Omega, I_t, I_{t-1}) \coloneqq \left\{ F(\mathrm{u}, \mathrm{v}) = \begin{bmatrix} \Delta_{\mathrm{u}}(I_t, I_{t-1}) \\ \Delta_{\mathrm{v}}(I_t, I_{t-1}) \end{bmatrix} \quad \forall (\mathrm{u}, \mathrm{v}) \in \Omega \right\}. \tag{3.35}$$

The knowledge of the object spatial velocity $\mathcal{V}_t$ and of the depth image D_t at the same instant of time, makes it possible to predict the object optical flow. To this end, we rewrite the pinhole camera model in Eq. (3.28) by moving the pixel coordinates u and v to the left-hand side:

$$\begin{aligned} \mathrm{u} &= c_x + \frac{p_{\mathrm{uv},x}}{d_{\mathrm{uv}}} f_x = c_x + \frac{p_x}{d} f_x, \\ \mathrm{v} &= c_y + \frac{p_{\mathrm{uv},y}}{d_{\mathrm{uv}}} f_y = c_y + \frac{p_y}{d} f_y. \end{aligned} \tag{3.36}$$

For the sake of clarity, we omit the subscript uv in the x and y coordinates p_x and p_y and in the depth d.

Differentiating the above expression with respect to time, the velocity of the u coordinate is as follows:

$$\begin{aligned} \dot{u} &= f_x \left(\frac{1}{d} \dot{p}_x + p_x (\dot{d^{-1}}) \right) \\ &= f_x \left(\frac{1}{d} \dot{p}_x - \frac{p_x}{d^2} \dot{d} \right) \\ &= f_x \left(\frac{1}{d} \dot{p}_x - \frac{p_x}{d^2} \dot{p}_z \right), \end{aligned} \tag{3.37}$$

where we use the fact that $d = p_z$. Taking into account Eq. (3.28), it can be further expanded to:

$$\begin{aligned}\dot{\mathrm{u}} &= f_x\left(\frac{1}{d}\dot{p}_x - \frac{\mathrm{u}-c_x}{f_x}\frac{d}{d^2}\dot{p}_z\right)\\ &= \frac{f_x}{d}\dot{p}_x - \frac{\cancel{f_x}}{\cancel{f_x}}\frac{\mathrm{u}-c_x}{d}\dot{p}_z\\ &= \frac{f_x}{d}\dot{p}_x - \frac{\mathrm{u}-c_x}{d}\dot{p}_z.\end{aligned} \tag{3.38}$$

The same reasoning extends to the other coordinate v. Using the vector notation, the following holds:

$$\begin{aligned}\begin{bmatrix}\dot{\mathrm{u}}\\ \dot{\mathrm{v}}\end{bmatrix} &= \begin{bmatrix}\frac{f_x}{d} & 0 & -\frac{\mathrm{u}-c_x}{d}\\ 0 & \frac{f_y}{d} & -\frac{\mathrm{v}-c_y}{d}\end{bmatrix}\begin{bmatrix}\dot{p}_x\\ \dot{p}_y\\ \dot{p}_z\end{bmatrix}\\ &:= J_v(d)^I\dot{p}_{\mathrm{uv}},\end{aligned} \tag{3.39}$$

where we defined the matrix $J_v(d)$ and ${}^I\dot{p}_{\mathrm{uv}}$ is simply the velocity of the point P_{uv} expressed in the camera frame. In order to make explicit the connection between the above model and the object spatial velocity $\mathcal{V}$, we rewrite the velocity ${}^I\dot{p}_{\mathrm{uv}}$, resorting to Eq. (3.24), as follows

$${}^I\dot{p}_{\mathrm{uv}} = v_{O_i} + \omega \times {}^Ip_{\mathrm{uv}}, \tag{3.40}$$

where ω is the angular velocity of the object $\mathcal{O}$ and v_{O_i} is the velocity of a rigid extension of the object that is instantaneously coincident with the origin of the camera frame O_i, i.e. the linear velocity v part of the spatial velocity $\mathcal{V}$. Substituting the velocity in (3.39), the pixel locations velocities become:

$$\begin{aligned}\begin{bmatrix}\dot{\mathrm{u}}\\ \dot{\mathrm{v}}\end{bmatrix} &= J_v(d)^I\dot{p}_{\mathrm{uv}}\\ &= J_v(d)(v - {}^I\hat{p}_{\mathrm{uv}}\omega)\\ &= J_v(d)v - J_v(d)^I\hat{p}_{\mathrm{uv}}\omega\\ &= \begin{bmatrix}J_v(d) & -J_v(d)^I\hat{p}_{\mathrm{uv}}\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix}\\ &:= \begin{bmatrix}J_v(d) & J_\omega\end{bmatrix}\mathcal{V},\end{aligned} \tag{3.41}$$

where we defined the matrix J_ω and we used the fact that $\omega \times {}^I p_{\mathrm{uv}} = -{}^I p_{\mathrm{uv}} \times \omega = -{}^I \hat{p}_{\mathrm{uv}} \omega$. The term J_ω can be further expanded to

$$\begin{aligned}
-J_v(d)^I \hat{p}_{\mathrm{uv}} &= \begin{bmatrix} \frac{f_x}{d} & 0 & -\frac{u-c_x}{d} \\ 0 & \frac{f_y}{d} & -\frac{v-c_y}{d} \end{bmatrix} \begin{bmatrix} 0 & p_z & -p_y \\ -p_z & 0 & p_x \\ p_y & -p_x & 0 \end{bmatrix} \\
&= \begin{bmatrix} -\frac{\mathrm{u}-c_x}{d} p_y & \frac{f_x}{d} p_z + \frac{\mathrm{u}-c_x}{d} p_x & -\frac{f_x}{d} p_y \\ -\frac{f_y}{d} p_z - \frac{\mathrm{v}-c_y}{d} p_y & \frac{\mathrm{v}-c_y}{d} p_x & \frac{f_y}{d} p_x \end{bmatrix} \\
&= \begin{bmatrix} -\frac{(\mathrm{u}-c_x)(\mathrm{v}-c_y)}{f_y} & \frac{f_x^2+(\mathrm{u}-c_x)^2}{f_x} & -\frac{(\mathrm{v}-c_y) f_x}{f_y} \\ -\frac{f_y^2+(\mathrm{v}-c_y)^2}{f_y} & \frac{(\mathrm{u}-c_x)(\mathrm{v}-c_y)}{f_x} & \frac{(\mathrm{u}-c_x) f_y}{f_x} \end{bmatrix},
\end{aligned}$$

where we substituted p_x, p_y and p_z according to Eq. (3.28).

By recalling that $\Delta_{\mathrm{u}} = \dot{\mathrm{u}} \Delta_t$ and $\Delta_{\mathrm{v}} = \dot{\mathrm{v}} \Delta_t$, it is finally possible to predict the optical flow of a given pixel (u, v) given the object spatial velocity $\mathcal{V}_t$ and the depth d_{uv} as follows:

$$\begin{bmatrix} \Delta_{\mathrm{u}}(\mathcal{V}_t, d_{\mathrm{uv}}) \\ \Delta_{\mathrm{v}}(\mathcal{V}_t, d_{\mathrm{uv}}) \end{bmatrix} = \Delta_t \begin{bmatrix} J_v(d_{\mathrm{uv}}) & J_\omega \end{bmatrix} \mathcal{V}_t. \tag{3.42}$$

Given a set of pixels Ω in the image that contain the object of interest, we define the *predicted* optical flow associated to the object $\mathcal{O}$ given the object spatial velocity $\mathcal{V}_t$ and the depth image D_t as the set of displacements

$$F(\Omega, \mathcal{V}_t, D_t) := \left\{ \begin{bmatrix} \Delta_{\mathrm{u}}(\mathcal{V}_t, d_{\mathrm{uv}}) \\ \Delta_{\mathrm{v}}(\mathcal{V}_t, d_{\mathrm{uv}}) \end{bmatrix} \quad \forall (\mathrm{u}, \mathrm{v}) \in \Omega \mid d_{\mathrm{uv}} \in D_t \right\}. \tag{3.43}$$

The definition of the optical flow and the predicted optical flow of the object and their comparison makes it possible to relate the object velocity $\mathcal{V}_t$ with consecutive images I_{t-1}, I_t and is useful for the development of object pose and velocity tracking algorithms from RGB-D images.

3.4 Evaluation metrics

Object pose tracking algorithms are evaluated primarily in terms of the *error* between the tracked pose T_t and the ground truth pose T_t^{gt}. In case of object pose and velocity tracking algorithms, the error between the velocity $\mathcal{V}_t$ and the ground truth velocity $\mathcal{V}_t^{gt}$ is also considered.

Drawing from the classical literature on target tracking (Bar-Shalom et al., 2002), the Root Mean Square Error (RMSE) is one of the possible metrics that can be used to evaluate the tracking error of a given algorithm along a given test trajectory.

Recently, other metrics, as the Average Distance Distinguishable (ADD) and the Average Distance Indistinguishable (ADI), were proposed in the literature of 6D object pose estimation (Xiang et al., 2018).

In the present work, we adopt both metrics as they provide insights on different aspects of the performance of a given algorithm.

Another important evaluation criterion is the *execution time* defined as the average time required to execute a single iteration of the algorithm under examination.

In this section, I will briefly introduce and discuss the aforementioned metrics that have been used to compare the methods proposed in the present Thesis with other state-of-the-art algorithms.

3.4.1 Root Mean Square Error (RMSE)

Given a discrete time signal $\{x_t\}$, that is N steps long, and a ground truth signal $\{x_t^{gt}\}$ to be compared, the Root Mean Square Error (RMSE) associated to $\{x_t\}$ is defined as follows:

$$\mathrm{RMSE}(\{x_t\}) = \sqrt{\frac{1}{N}\sum_{t=1}^{N} e(x_t, x_t^{gt})^2}, \tag{3.44}$$

where $e(x_t, x_t^{gt}) \in \mathbb{R}$ represent the error between the estimated and ground truth signals at time t. The definition of the error e depends on the set x_t belongs to.

Positional RMSE

Given a pose $T_t \in \mathrm{SE}(3)$ and its positional component $p_t \in \mathbb{R}^3$, the positional RMSE error in *meters* is obtained by defining the error e as

$$e(p_t, p_t^{gt}) = \|p_t - p_t^{gt}\| \quad (\mathrm{m}). \tag{3.45}$$

Rotational RMSE

Given a pose $T_t \in \mathrm{SE}(3)$ and its rotational component $R_t \in \mathrm{SO}(3)$, the rotational RMSE error in *degrees* is obtained by defining the error e as

$$e(R_t, R_t^{gt}) = \|\log(R_t^{gt} R_t^T)^\vee\| \; \frac{180}{\pi} \quad (\text{deg}), \tag{3.46}$$

where

- $\log(R_t^{gt} R_t^T) \in \mathfrak{so}(3)$ is the matrix logarithm of the relative orientation $R_t^{gt} R_t^T$;
- $\log(R_t^{gt} R_t^T)^\vee \in \mathbb{R}^3$ is the vector representation of the matrix logarithm;
- $\|\log(R_t^{gt} R_t^T)^\vee\|$ represents a proper error metric on SO(3) taking values in the interval $[0, \pi]$ (Huynh, 2009).

We remark that this metric is easily evaluated as the angle of rotation of the matrix $R_t^{gt} R_t^T$ (Huynh, 2009).

Linear velocity RMSE

Given a spatial velocity $\mathcal{V} \in \mathbb{R}^6$ and its linear velocity component $v \in \mathbb{R}^3$, the linear velocity RMSE in *meters per second* is obtained by defining the error e as

$$e(v_t, v_t^{gt}) = \|v_t - v_t^{gt}\| \quad (\text{m/s}). \tag{3.47}$$

The same definition can be applied to the mixed velocity $\mathcal{V}^m$.

Angular velocity RMSE

Given a spatial velocity $\mathcal{V} \in \mathbb{R}^6$ and its angular velocity component $\omega \in \mathbb{R}^3$, the angular velocity RMSE in *degrees per second* is obtained by defining the error e as

$$e(\omega_t, \omega_t^{gt}) = \|\omega_t - \omega_t^{gt}\| \frac{180}{\pi} \quad (\text{deg/s}). \tag{3.48}$$

3.4.2 Average Distance Distinguishable (ADD)

The Average Distance Distinguishable (ADD) is a popular metric used to benchmark 6D object pose estimation algorithms (Xiang et al., 2018). Nonetheless, it has been adopted also to benchmark 6D object pose tracking algorithms (Deng et al., 2019; Wen et al., 2020a). Moreover, using this metric allows comparing with other state-of-the-art algorithms especially when their implementation is not publicly available.

Given a triangle mesh description of the object, a list of n points $\{A_j\}$, sampled on the mesh, the pose T_t and the ground truth pose T_t^{gt}, we define the two point cloud sets $C(\{A_j\}, T_t) = \{a_j\}$ and $C(\{A_j\}, T_t^{gt}) = \{a_j^{gt}\}$ according to Eq. (3.31). The ADD metric is then defined as

$$\mathrm{ADD}(T_t) = \frac{1}{n}\sum_j \|a_j^{gt} - a_j\|. \tag{3.49}$$

In the definition above, we assume that the coordinate vectors a_j and a_j^{gt} are associated to the same 3D point A_j sampled on the surface of the object $\partial\mathcal{O}$, as described in Sec. 3.3.1.

The ADD metric does not take into account the actual error between the estimated and ground truth poses. Instead, it directly compares the predicted point clouds of the object at the estimated and ground truth poses. Specifically, it considers the mean distance between the points belonging to the two point clouds.

The ADD metric is not usually used as it is, instead the following derived metrics are considered.

Percentage of ADD under a threshold

Given a trajectory $\{T_t\}$ of N object poses, the ADD $< \epsilon$ metric is defined as the percentage of poses for which the ADD metric is below a given threshold ϵ in meters:

$$\mathrm{ADD} < \epsilon = \frac{|\{t \mid \mathrm{ADD}(T_t) < \epsilon\}|}{N} \in [0, 1], \tag{3.50}$$

where $|\cdot|$ indicates the cardinality of a set.

Area under the curve of ADD given a threshold

Given a trajectory $\{T_t\}$ of N object poses, we define the ordered set of ADD distances less than a threshold ϵ as

$$\mathrm{ADD}(\{T_t\})_\epsilon \coloneqq \left\{0, \mathrm{ADD}(T_{\tilde{t}(1)}), \cdots, \mathrm{ADD}(T_{\tilde{t}(k)})\right\} \cup \{\epsilon\}, \tag{3.51}$$

where $\tilde{t}(\cdot)$ is a permutation of the instants of time $\{1, \cdots, N\}$ such that

$$0 \leq \text{ADD}(T_{\tilde{t}(1)}) \leq \cdots \leq \text{ADD}(T_{\tilde{t}(k)}) \leq \epsilon \tag{3.52}$$

and $k \leq N$. The strict equality $k < N$ holds if there are poses for which the ADD distance is greater than ϵ. A set of accuracies

$$\left\{0, \frac{1}{N\epsilon}, \cdots, \frac{k}{N\epsilon}\right\} \cup \left\{\frac{k}{N\epsilon}\right\} \tag{3.53}$$

is associated to the list of the distances. Then, we define the curve of ADD distances (ADD-C) as the curve

$$\text{ADD-C}(\{T_t\})_\epsilon \coloneqq \left\{(0,0), (\text{ADD}(T_{\tilde{t}(1)}), \frac{1}{N\epsilon}), \cdots, (\text{ADD}(T_{\tilde{t}(k)}), \frac{k}{N\epsilon}), (\epsilon, \frac{k}{N\epsilon})\right\}, \tag{3.54}$$

where each pair is a point belonging to the curve. The area under the curve (ADD-AUC) metric is then defined as the rectangular approximation of the integral of the ADD-C curve.

In the ideal situation in which all the distances are equal to zero, i.e. the algorithm under examination does not commit any error, then $k = N$ and the integral evaluates to $\epsilon\frac{k}{N\epsilon} = 1$. On the other hand, if all distances are greater than ϵ then $k = 0$ and the area evaluates to 0. In general, the ADD-AUC metric belongs to the interval $[0, \frac{k}{N}] \subseteq [0, 1]$.

Differently from the ADD $< \epsilon$ metric, which only considers the percentage of distances below a given threshold, the ADD-AUC metric takes into account how the distances are distributed in the interval $[0, \epsilon]$. As a consequence, two algorithms that have the same ADD $< \epsilon$, might have very different ADD-AUC metrics.

3.4.3 Average Distance Indistinguishable (ADI)

The Average Distance Indistinguishable (ADI) is defined similarly to the ADD metric with the difference that it can be used for objects with rotational symmetries that could cause ambiguities in the pose estimation process. The ADI metric is defined as

$$\text{ADI}(T_t) = \frac{1}{n}\sum_j \min_k \|a_j^{gt} - a_k\|, \tag{3.55}$$

where the distance between the points a_j^{gt} and a_j, as in the ADD metric, is substituted with the minimum distance between each point a_j^{gt} and each other point a_k. The ADI $< \epsilon$ and ADI-AUC metrics are similarly defined.

3.4.4 Comparison between the ADD/ADI and the RMSE metrics

The ADD and ADI-based metrics do not express the actual translational or rotational error directly. Instead, they provide statistics depending on the number of poses having an ADD or ADI distance lower than the threshold ϵ and on how it compares with the total number of poses. The actual magnitude of the distance of all the other poses does not affect these metrics, irrespective of the fact that it is slightly greater or much greater than ϵ, as might happens if the algorithm produces spikes in the output. However, the presence of spikes might negatively affect the execution of robotic tasks involving closed loop control. For this reason, we also consider the RMSE metric which is more susceptible to such spikes as it does not use any thresholding mechanism.

3.5 Classical object pose tracking

In this section, I will provide an overview of classical approaches for object pose tracking. The majority of these methods have been proposed before the recent spread of Deep Learning techniques in the computer vision and robotics communities.

In Sec. 3.5.1, I first present methods that consider the object as a standalone entity. Then, in Sec. 3.5.2, I overview methods that track the object pose during an interaction with the robot end-effector. Finally, in Sec. 3.5.3, I briefly mention methods that jointly track the shape and the pose of the object.

For the sake of comparison, Table 3.1 summarizes the main characteristics of the presented works, including the methods that are described in the following Secs. 3.6 and 3.7.

3.5.1 Object pose tracking

Bayesian filtering-based methods

The most classical approaches for object pose tracking combine Bayesian filtering with depth information in order to track the 6D pose of the object.

Table 3.1 Comparison of the main characteristics of the state-of-the-art methods presented in Secs. 3.5, 3.6 and 3.7.

Reference	Task	Methodology	Input	Initial conditions	Object modelling	Training requirements	Frame rate
Wüthrich et al. (2013)	6D pose tracking	particle filtering	depth	6D pose	3D mesh	/	30 Hz
Issac et al. (2016)	6D pose/velocity tracking	Kalman filtering	depth	6D pose/velocity	3D mesh	/	30 Hz
Choi and Christensen (2013)	6D pose tracking	particle filtering	RGB-D	6D pose	textured 3D mesh	/	20 Hz
Jongeneel et al. (2021)	6D pose/velocity tracking	particle filtering	RGB	6D pose/velocity	3D mesh	/	N.A.
Tjaden et al. (2019)	6D pose tracking	optimization	RGB	6D pose	3D mesh	/	N.A.
Li and Stueckler (2021)	6D pose/velocity tracking	optimization	RGB-D, events	6D pose	3D mesh	/	3 Hz
Bauer et al. (2020)	6D pose refinement	optimization	RGB-D	6D pose	3D mesh	/	N.A
Gao and Tedrake (2019)	6D point-set registration	expect. maxim.	RGB-D	/	3D mesh	/	77 Hz
Yang et al. (2020)	6D point-set registration	truncated least squares	RGB-D	/	3D mesh	/	N.A.
Vezzani et al. (2017)	6D pose localization	particle filtering	3D contacts	6D pose	3D mesh	/	N.A.
Koval et al. (2013)	3D pose tracking	particle filtering	3D contacts	3D pose	3D mesh	/	20 Hz
Hebert et al. (2012)	6D pose tracking	Kalman filtering	RGB-D	6D pose	3D mesh	/	8 Hz
Wen et al. (2020b)	6D point-set registration	optimization	RGB-D	/	3D mesh	/	1.6 Hz
Costanzo et al. (2020)^1	1D velocity tracking	Kalman filtering	tactile data	/	/	/	2000 Hz
Izatt et al. (2017)	6D pose/velocity tracking	optimization	depth, 3D contacts	6D pose/velocity	SDF†	/	12 Hz
Schmidt et al. (2015)	6D pose tracking	optimization	depth, tactile data	6D pose	SDF†	/	30 Hz
Yu and Rodriguez (2018a)	3D pose tracking	optimization	Vicon, FT sensing	3D pose	/	/	100 Hz
Liang et al. (2020)	6D pose/velocity tracking	particle optimization	RGB, tactile data	6D pose/velocity	3D mesh	/	30 Hz
Kumru and Özkan (2021)	6D pose/velocity tracking	Kalman filtering	depth	6D pose/velocity	GPIS‡	/	27 Hz
Grinvald et al. (2021)	6D pose tracking	point-set registration	depth	6D pose	SDF†	/	1 Hz
Joseph Tan et al. (2015)	6D pose tracking	point-set registration	depth	6D pose	3D mesh	random forest	30 Hz
Garon and Lalonde (2017)	6D pose tracking	CNN for regression	RGB-D	6D pose	textured 3D mesh	CNN (per object)	125 Hz
Leeb et al. (2019)	6D pose tracking	LSTM for regression	RGB	6D pose	textured 3D mesh*	LSTM	N.A.
Wen et al. (2020a)	6D pose tracking	CNN for regression	RGB-D	6D pose	textured 3D mesh	CNN (per object)	90 Hz
Dubeau et al. (2020)	6D pose tracking	CNN for regression	RGB-D, events	6D pose	textured 3D mesh	CNN (per object)	34 Hz
Ge and Loianno (2021)	6D pose tracking	CNN for regression	RGB, IMU	6D pose	textured 3D mesh	CNN (per object)	50 Hz
Manhardt et al. (2018)	6D pose refinement	CNN for regerssion	RGB	6D pose	textured 3D mesh*	CNN (per object)	40 Hz
Li et al. (2018)	6D pose refinement	CNN for regression	RGB	6D pose	textured 3D mesh	CNN (per dataset)	12 Hz
Wong et al. (2017)	6D pose tracking	Kalman filtering	RGB-D	/	3D mesh	CNN for segmentation	14 Hz
Deng et al. (2019)	6D pose tracking	particle filtering	RGB-D	2D bounding box	textured 3D mesh	CNN (per object)	17.5 Hz
Runz et al. (2018)	6D pose tracking	optimization	RGB-D	6D pose	surfels	CNN for segmentation	30 Hz
Zhang et al. (2020)	6D pose/velocity tracking	optimization	RGB-D	/	/	CNN for segmentation	5 Hz
Masuda et al. (2021)	6D pose/velocity tracking	optimization	events	6D pose	/	neural network	N.A.
Periyasamy et al. (2019)	6D pose refinement	optimization	RGB	6D pose	textured 3D mesh*	CNN	20 Hz
Iwase et al. (2021)	6D pose refinement	optimization	RGB	6D pose	3D mesh	CNN	92 Hz
Yu and Rodriguez (2018b)	6D pose tracking	optimization	AprilTag, FT sensing	3D pose	/	support vector machine	10 Hz
Suresh et al. (2021)	3D pose tracking	optimization	Vicon, FT sensing	3D pose	GPIS‡	Gaussian process	100 Hz
Sodhi et al. (2021a)	3D pose tracking	optimization	RGB tactile data	3D pose	/	CNN	N.A.
Sodhi et al. (2021b)	6D pose tracking	optimization	RGB tactile data	6D pose	/	CNN	N.A.

* Only required at training time

† Signed Distance Function (Osher and Fedkiw, 2003)

‡ Gaussian Process Implicit Surface (Williams and Fitzgibbon, 2006)

^{1} We refer to the part of this work dealing with the tracking of the relative rotational velocity of the object with respect to the end-effector

Wüthrich et al. (2013) use a particle filter (Ristic et al., 2003) to update the state of the object in real-time given the whole depth image of the scene. In order to dismiss parts of the image that does not contain the object, due to the limited extension of the object or occlusions, they propose to model occlusions as additional binary states for each pixel. Despite the high-dimensionality of the state, the method runs in real-time.

Issac et al. (2016) use instead an Unscented Kalman filter to track the object pose and velocity from depth images in real-time. The proposed filter handles the occlusions by modelling the depth noise using a heavy-tailed Gaussian distribution. Such a distribution allows modelling the occlusions as outliers in the depth measurements, that are rejected thanks to an enhanced formulation of the filter. The proposed filter provides smoother estimates in general and steadier estimates than the particle filter in (Wüthrich et al., 2013) in presence of total occlusion of the object.

Despite the promising results, recent works, e.g. (Wen et al., 2020a), have shown the limits of the aforementioned methods in terms of object pose accuracy. Moreover, the recent availability of object segmentation algorithms (He et al., 2017; Xiang et al., 2018), which identify the pixels in the image containing the object, could make the proposed strategies for occlusion handling overly complex nowadays. In the Chapter 4 of this Thesis, I will present an algorithm similar to (Issac et al., 2016) that does not need ad-hoc strategies for occlusion handling, rather it exploits recent advances in deep learning for object segmentation and combines them with Kalman Filtering to accomplish the object pose tracking task.

Choi and Christensen (2013) propose a similar approach as Wüthrich et al. (2013) while considering also the normals to the surface of the object and the points colors in the formulation of the measurement models. The information from the normals allow alleviating the uncertainty in the association between the measured and predicted object surfaces. Using the colours, instead, help handle rotational ambiguities that can be disambiguated using the object texture. On the other hand, this method requires a textured 3D model of the object and the estimates of the surface normals, which might be noisy. This method requires a large number of particles, in the order of thousands, to accomplish the tracking task with real sequences. The resulting frame rate, which is approximately 20 frames per second, could make the algorithm unsuitable for tracking fast object motions as those considered in the Chapter 5 of this Thesis.

In (Jongeneel et al., 2021), the authors concentrate on the task of tracking the pose of objects that are tossed on a surface. This problem is challenging due to the abrupt changes in the velocity of the object after the impact with the surface. In this respect, they design an Unscented Particle Filter (Van Der Merwe et al., 2000) which incorporates the effects of the impact and the role of the friction in a motion model. The particles are then weighted by projecting several 3D object points onto the image plane and by evaluating the fitness with the incoming RGB images using a color histograms-based likelihood function. As mentioned above, this Thesis also addresses the problem of tracking fast object motions which is closely related to the topic addressed in (Jongeneel et al., 2021). However, the latter focuses on the specific scenario of objects that are tossed on a surface, which is not considered in this Thesis.

Optimization-based methods

Tjaden et al. (2019) propose to reduce the set of features to be used for tracking to plain RGB data. They use a probabilistic formulation which divides the image in subregions each associated to a temporally consistent color histogram that is updated at each frame. The histograms contribute to local cost functions which evaluate the consistency between the predicted silhouette of the object, obtained from the 3D model, and the actual image. The method optimizes the cost function using a second order optimization method providing faster convergence rates than common gradient descent-based methods, leading to real-time performance.

Very recently, Li and Stueckler (2021) combined RGB-D data with events from an event camera. They use optimization to find the pose of the object which minimizes the error between the measured events and the predicted events, depending on the depth and on the object velocity. They also add a regularization term which minimizes the instantaneous object acceleration. The pose of the object is also evaluated on sparse keyframes, using RGB data, that are used to constraint the trajectory obtained from the events in order to fuse the two modalities. A major limitation of this method is that it cannot run in real time, making it unsuitable for real-time tracking purposes.

We remark that the above mentioned methods employ slightly different sensing modalities than those adopted in this Thesis. I.e. Tjaden et al. (2019) use RGB images only and Li and Stueckler (2021) use also image events while the methods presented in this Thesis employ RGB-D sensing. Nonetheless, they are reported for the sake of completeness.

An additional class of methods solve the so-called “object pose refinement” task. Given an initial condition, usually the output of an object pose estimator, they iteratively refine the pose of the object. In this scenario the object is typically static, i.e. it does not move. Although their aim is slightly different from tracking, the iterative nature of these methods make them comparable. The majority of these methods use deep learning techniques and will be presented in next Secs. 3.6 and 3.7. Nevertheless, there are also works which tackle the problem using numerical optimization solely.

(Bauer et al., 2020) propose a simulation-based physics-guided iterative refinement process from RGB-D images which enforces the physical plausibility of the object pose in presence of constraints such as the surface of a table or other objects. Although this method cannot run in real time, it might be useful to track objects in cluttered scenes if they move quite slowly.

Point set registration-based methods

Several methods for object pose tracking have been proposed in the context of point set registration, starting from the Iterative Closest Point (ICP) method (Besl and McKay, 1992). These methods find the pose of the object that reduces the distance error between the measured partial point cloud of the object and the predicted point cloud according to the current pose. They thus neglect the problem of extracting the partial point cloud of the object from the whole scene point cloud and concentrate on the registration problem.

Gao and Tedrake (2019) propose FilterReg, a probabilistic method for point cloud registration that is cast as a maximum likelihood estimation problem and solved using the expectation-maximization (EM) algorithm. Differently from ICP, which associates measured and predicted points using a nearest neighbours metric, FilterReg models the predicted point cloud as a Gaussian Mixture Model (GMM). This allows introducing more complex weighting mechanisms, for each point, and handle the presence of outliers. The tracking capabilities of this method are shown in a table-top robot manipulation scenario, where FilterReg tracks the pose of each link of the robot and the manipulated object using an external depth camera while treating the points from the table as outliers.

In TEASER (Yang et al., 2020), the authors dedicate even more attention to the problem of rejecting the outliers by providing a certifiable method, i.e. a method which provides easy to check conditions that indicate if the provided solution is optimal or not. This method runs in real-time which makes it feasible for tracking purposes.

We recall that registration methods minimize the distance error between the measured and the target point cloud. On the other hand, they do not provide an estimate of the object velocity, which might be crucial to close the control loop in specific robotic applications (Piga et al., 2021a; Viña et al., 2015). In the Chapter 4 of this Thesis, I will provide experimental results supporting this statement.

3.5.2 Object/end-effector pose tracking

Several works make the additional assumption that the object is within the end-effector of a robot, e.g. during a grasp, or in contact with it, e.g. during pushing or more general manipulation actions.

Bayesian filtering-based methods

Vezzani et al. (2017) propose the Memory Unscented Particle Filter (MUPF) to localize the pose of an object given a set of 3D tactile point measurements acquired on the surface of the object. The proposed method processes the measurements sequentially in small batches while taking into account the multimodality of the pose distribution due to the reduced number of points in a batch. Given that the measurements are in the form of 3D points, the algorithm could be extended to mixed sets of points from touch and vision, however this scenario has not been explored.

The MUPF updates the pose by spreading the particles in the state space according to a motion model and then weighting them according to the incoming measurements. Koval et al. (2013) propose to do the opposite. Indeed, when working with tactile measurements belonging to the surface of the object, they found that exploring the state space with a motion model can produce many pose candidates that are not in contact with the measurements. As a result, the pose distribution is not adequately represented by the particles. For this reason, they propose to sample the particles from a distribution which assigns high probability to those that are consistent with the tactile measurements, and then weight them according to the motion model. This choice also reduces the number of particles with respect to conventional particle filters thus increasing the frame rate.

In Hebert et al. (2012), the authors do not use tactile information and propose an Unscented Kalman filter for pose tracking that exploits depth, visual RGB features and the 2D silhouette to update the object pose. The same strategy is applied to parts of the end-effector of the robot, which makes possible to track its relative pose with respect to the object. This information is useful as feedback to maintain a stable contact with the object, e.g. during grasping or pushing actions, using a closed loop controller.

More recently, Wen et al. (2020b) focused on scenarios where the robot end-effector occludes the object to be tracked. The proposed approach first estimates the hand state using a particles-based approach which tries to maximize the overlap between the hand 3D model and the point cloud. Then, given the state of the hand, the point cloud of the hand is removed from that of the scene. The remaining points are used to propose object pose candidates using an hypotheses generation method (Mitash et al., 2019). Finally, false hypotheses are pruned using physical reasoning, e.g. avoiding compenetration with the hand, and the remaining poses are evaluated according to fitness with the observed point cloud. Remarkably, this work compares with some RGB-based deep neural networks for pose estimation showing how hand-object occlusions cause important drop in performance if the end-effector is not taken into account in the problem formulation. The method cannot run in real-time, however it can be useful to track slowly moving objects during grasping or post-grasping stages.

Hang et al. (2020) study a related but fundamentally different problem. They assume underactuation in the robot hand which makes difficult to sense the hand configuration and the position of the contacts between the hand and the object. Readings from the hand encoders are not available as well in the considered scenario. Instead, they measure the pose of the tip of the object using a fiducial marker and combine it with geometric grasp constraints within a particle filter. The filter weights several hand configuration hypotheses using the information of the observed object's movements and iteratively tracks the hand configuration. The proposed filter is eventually used to control the pose of the grasped object without using joints encoders nor tactile sensing.

Although some of methods presented in this section estimate the pose of both the object and the end-effector separately, e.g. (Hebert et al., 2012) and (Wen et al., 2020b), none of them explicitly consider the possibility that the object undergoes non-negligible relative motions with respect to the end-effector. This hypothesis is considered instead in Costanzo et al. (2020). The authors study in-hand object manipulation tasks under the hypothesis that the object might slide between the fingers of a parallel jaw gripper equipped with tactile sensors. They control the grip force acting on the object in order to avoid possible rotational slippage or generate in-hand object rotations that allow changing the grasp configuration without performing re-grasping. In order to close the control loop, they estimate the relative rotational velocity of the object with respect to the fingers using an Extended Kalman Filter that incorporates a dynamical model of the rotational friction acting on the object.

In the Chapter 6 of this Thesis, I will consider a scenario that is closely related to that of (Costanzo et al., 2020) as regards the *perception* of the object state. Specifically, I will address the problem of estimating the relative translational velocity of the object with respect to the fingers of a humanoid robot hand during in-hand sliding motions using tactile sensors. The estimate is obtained using an Extended Kalman Filter whose measurement model is learned from ground-truth data using the differentiable Kalman filtering paradigm (Kloss et al., 2021; Lee et al., 2020). In this regard, the presented developments do not extend the work presented in (Costanzo et al., 2020), rather they explore a different approach to addressing the object *perception* problem while focusing on translational motions instead of rotational ones. We remark that, differently from (Costanzo et al., 2020), this Thesis does not address the problem of *controlling* the object sliding motion.

Optimization-based methods

Izatt et al. (2017) use the optimization-based framework DART (Schmidt et al., 2014) for object tracking from depth-images to combine depth data with tactile point clouds. The tactile point clouds are reconstructed using the vision-based sensor GelSight (Yuan et al., 2017) while the object is manipulated by the robot end-effector. Interestingly, the proposed method explicitly models the non-penetration constraint between the hand and the object to favour physically plausible object pose estimates. This work demonstrates that the addition of contact-based geometric information significantly improves the tracking performance during contact, and provides robustness to occlusions of small objects due to the robot end-effector.

The DART framework (Schmidt et al., 2014) is similarly used in (Schmidt et al., 2015) where the vision-based sensors are substituted with more general force-torque sensors and touch sensors for contact detection.

Fusion of visual and tactile sensing is also explored in Yu and Rodriguez (2018a). The authors use the incremental smoothing and mapping tracking framework (iSAM) (Kaess et al., 2008) to track the pose of a known planar object during pushing actions. The task is accomplished by fusing a global visual prior with local contact sensing implemented using force-torque sensors. The method runs faster than real-time at 100 Hz. Differently from other works, this method considers a simple 3D tracking scenario which accounts for the 2D Cartesian position on the plane and one rotational degree of freedom. On the other hand, it is one of the few to model the object pushing constraint which is more challenging than usual in-hand object interactions where the object is constrained by the fingers of the end-effector.

We remark that the above mentioned methods track the object pose by using visual and tactile data at the same time. Instead, the methods presented in this Thesis consider the two sensing modalities separately. Nonetheless, these methods are reported for the sake of completeness.

Very recently, Liang et al. (2020) propose to use GPU-accelerated parallel simulations of the hand-object interaction in an online particle-based optimizer to track the 6D object pose of the object using tactile feedback. The considered scenario is that of in-hand object manipulation where, differently from other works, relative motions between the fingers and the object are allowed. The proposed method is initialized using a noisy visual prior. Interestingly, this is one of the few works that explicitly considers the object slippage with a specific constraint in the problem formulation. The results are obtained using simulated and real data from a robot setup. However, the slippage constraint is implemented only in simulation. In this respect, the developments from the Chapter 6 of this Thesis provide insights on a closely related problem, i.e. tracking in-hand object sliding motions from tactile data, supported by experimental results from a real robot setup.

3.5.3 Joint object reconstruction and pose tracking

We would like to mention few works that extend the problem of tracking the pose of the object to the joint tracking of the pose and reconstruction of the model of the object. Although this Thesis does not address the problem of reconstructing the model of the object, the following works are reported for the sake of completeness.

Kumru and Özkan (2021) propose a Gaussian processes-based method to jointly tracks the object position, orientation and velocity, together with the shape of the object from a partial 3D point cloud of the object. The method is implemented efficiently by casting the Gaussian process regression to an iterative Extended Kalman filtering task. Remarkably, the proposed method provides an analytical description of the 3D object shape together with confidence intervals that can be used to understand which parts of the shape estimate are more uncertain and require more information to be refined. Furthermore, given that the shape is not only estimated but tracked over time, this method potentially extends to non-rigid objects.

In (Grinvald et al., 2021), the authors use a single Truncated Signed Distance Function (TSDF) (Curless and Levoy, 1996; Osher and Fedkiw, 2003) volume to reconstruct the surface belonging to all the objects in the scene. The object pose is then tracked by using ICP on the reconstructed surfaces and the partial point cloud of each object, segmented using an external source of object masks. A major limitation of this method is that it does not reach real-time performance. Nonetheless, it is worth mentioning as it reconstructs and tracks several objects at the same time.

3.6 Deep learning for object pose tracking

In this section, I will present methods for 6D object pose tracking and refinement that use a full machine learning or deep learning formulation to accomplish the task.

3.6.1 Object pose tracking

Joseph Tan et al. (2015) use random forests (Denisko and Hoffman, 2018) to track the pose of a rigid object from depth images in real-time. The tracking problem is cast as a registration problem between the 3D points on the model of the object and the 3D points from the depth image. The errors between corresponding points are used as inputs to the random forest that is trained to predict the relative transformation that aligns the point clouds. Application of such relative transformation to the previous estimate allows tracking the object in consecutive frames. In order to track objects from different camera viewpoints, the training dataset contains the data corresponding to a fixed number of camera poses sampled around the object. The proposed method is able to provide the transformation even when using a subsets of the points from the visible object surface, which makes it robust to possible occlusions.

In (Garon and Lalonde, 2017), the authors propose to use a Convolutional Neural Network (CNN), which takes two RGB-D images as input, to directly regress the transformation in the object pose between consecutive time stamps. The first image fed to the CNN is the current observation from the camera, cropped according to the previous object pose. The second image is a synthetic rendering of the RGB and depth channels obtained using a textured 3D model of the object according to the previous pose. The training set is obtained by sampling several camera poses and generating pairs of images where the current observation from the camera is simulated by rendering the object on synthetic background and applying random background, noise and lightning. The actual tracking is obtained, as usual, by applying the regressed transformation to the previous object pose. This work compares against (Joseph Tan et al., 2015), showing better performance in terms of translational and rotational errors.

The idea of comparing rendered images with current observations is also considered in (Leeb et al., 2019). Differently from the previous work, object crops are here substituted with the output of a segmentation recurrent neural network, based on Long Short-Term Memory (LSTM) (Yu et al., 2019). The network takes as input the previous and current RGB frame and the previous mask. The actual tracking is obtained by means of two additional LSTM-based networks that take as input the previous and current RGB frames and masks and provide the object translation, scale and rotation as the output, respectively. The training set is obtained using a synthetic image generator that randomly picks object sizes, textures and initial poses. According to the authors, the tracking accuracy is maintained only for the first 30 to 60 frames, including frames with occlusions, implying that this method is not suitable for long-term tracking without re-initialization.

Recently, Wen et al. (2020a) proposed se(3)-TrackNet which shares very similar ideas to (Garon and Lalonde, 2017). Nonetheless, their implementation achieve an interesting frame rate of 90 fps. Furthermore, they provide results on the commonly adopted YCB-Video dataset (Xiang et al., 2018) which makes easier to compare to their work. The experimental results from the Chapter 4 of this Thesis emphasize some limitations of this method. Although it shows remarkable results when initialized using the ground truth pose of the object, it proves to be very sensitive to perturbations to the initial condition leading to early track loss. Similarly, experiments on tracking fast object motions, from Chapter 5, result in track loss after few frames. As the experimental results from this Thesis will prove, hybrid architectures for 6D object pose tracking, as those proposed in Chapters 4 and 5, are more robust to perturbations of the initial conditions and better suited to track fast object motions.

In (Dubeau et al., 2020), the authors extend their previous work (Garon and Lalonde, 2017), mentioned above, by considering an additional source of information in the form of events from an event camera. The integration is obtained by training an additional network which predicts the relative object pose transformation given the input events. The transformation is combined with the previous pose estimate to obtain a new prediction. Finally, the prediction is used to render a synthetic RGB-D frame of the object as required by the original pipeline in (Garon and Lalonde, 2017). This way RGB-D data and events are combined in a single pipeline.

Very recently, Ge and Loianno (2021) proposed VIPose. They adopt the same strategy as (Garon and Lalonde, 2017) and (Wen et al., 2020a). However, they do not use depth and, instead, provide inertial measurements, from an Inertial Measurement Unit (IMU) attached to the camera, to the network. This addition makes the predicted inter-frames transformation more robust to severe occlusions of the object as the information on the object motion can be recovered from the IMU data. One considerable limitation of this method is that the inertial information is consistent with the apparent motion of the object in the camera plane only if the object is not moving. In this respect, the method presented in the Chapter 5 of this Thesis, which complements the RGB data with the information on the 6D motion of the object extracted from the optical flow, can be used regardeless of whether the camera and/or the object are moving or not.

We stress the fact that most of the methods from this section, namely (Dubeau et al., 2020; Garon and Lalonde, 2017; Ge and Loianno, 2021; Wen et al., 2020a) require a separate training for each object of interest in order to learn the tracking task. In this regard, in the Chapter 4 of this Thesis I will describe a depth-based 6D object pose tracker which has much modest training requirements resolving to that of a segmentation network. The segmentation network can be trained for a set of objects in the first place, without requiring a separate training for each object. Furthermore, recent works, such as (Ceola et al., 2021), have shown that it is possible to extend the segmentation task to new objects, not considered in the original training set, online in a few seconds. To the best of our knowledge, a similar extension is not possible at the moment with deep learning architectures for 6D object pose tracking.

3.6.2 Object pose refinement

The following algorithms share several similarities with those presented in the previous section. Nonetheless, given that they are officially presented as “pose refinement” methods, they are described in a separate section.

Manhardt et al. (2018) train a deep neural network to predict the inter-frame object pose transformation from RGB data. Differently from the other methods, a new visual loss is proposed that drives the pose update by minimizing the alignment error of the object contours. This choice makes the algorithm segmentation-free, robust to occlusions and agnostic to symmetries. The method can run in real-time making it feasible for tracking purposes. The claimed accuracy is similar to that of 3D ICP methods without the need for depth data. We remark that this method requires a separate training for each object using synthetic images.

According to the authors of Li et al. (2018), the gap between Manhardt et al. (2018) and depth-based methods is nonetheless still large. For this reason they propose another pose refinement method, called DeepIM. Given an initial pose, the network iteratively refines the object pose by matching the rendered image against the observed RGB image. Interestingly, and differently from all the other learning-based methods presented thus far, the structure of DeepIM is inspired to that of networks for optical flow estimation such as FlowNet (Dosovitskiy et al., 2015). Indeed, taking the observed and rendered image and their corresponding masks as input, the convolution layers of FlowNet are used to output a feature map which is then fed to several fully connected layers to predict the relative pose. According to the authors, the obtained results show that intermediate representations related to optical flow are better suited for pose refinement rather than standard CNN features specialized for, e.g., object recognition tasks. Interestingly, the method presented in the Chapter 5 of this Thesis, named ROFT, shares similarities with this method as it uses the optical flow to estimate the 6D object velocity. The velocity is then used to update the object pose, a process which resembles the object pose refinement step of DeepIM. A substantial difference between the two methods is that DeepIM uses the features of the network FlowNet to accomplish the task, while ROFT uses the plain optical flow and a Kalman Filter without the need for additional training.

3.7 Hybrid approaches for object pose tracking

The chapter ends with an overview of state-of-the-art hybrid methods for object pose tracking and refinement. These methods combine Deep Learning techniques with Bayesian filtering or optimization.

3.7.1 Object pose tracking

Bayesian filtering-based methods

SegICP (Wong et al., 2017) incorporates off-the-shelf object segmentation networks with a multi-hypothesis point cloud registration procedure in order to estimate the pose of the object given its 3D model. The obtained pose is used as a measurement update in a Kalman filter to track the object pose and velocity. Although this work shares many similarities with the method proposed in the Chapter 4 of this Thesis, named MaskUKF, it lacks an experimental validation against a standard dataset using standard metrics. Moreover, it does not provide comparisons with other state-of-the-art methods. In this respect, the experimental evaluation of MaskUKF provides a more clear picture on how an object segmentation-aided depth-based 6D tracker compares with the state of the art.

PoseRBPF (Deng et al., 2019) formulates the object pose tracking problem as a Rao-Blackwellized particle filer (Särkkä, 2013) with decoupled rotation and translation tracking. The particles are used to represent possible hypotheses of the translational part of the state. The rotation space is instead represented using the latent space of an autoencoder (Martin et al., 2018) for image reconstruction. Rotation tracking is implemented by discretization of the rotation space into bins each associated to an embedding in the latent space. By comparing the latent vectors with the embeddings associated to the input image, the full rotational distribution of the object is tracked. Furthermore, the latent vectors allow weighting the particles of the filter. State-of-the art performances are achieved using a considerable number of particles resulting in non real-time performance. This fact represents a practical limitation when tracking fast object motions. Moreover, the fact that the state space is explored via random walks, as typically done in particle filters, also affects the performance in this scenario leading to track loss. These statements will be supported by the experimental evaluation from the Chapter 5. We remark that this method requires training the autoencoder separately for each object of interest given a textured 3D mesh model.

Optimization-based methods

Similarly to SegICP, MaskFusion (Runz et al., 2018) takes advantage of deep neural networks for object segmentation to identify parts of the RGB-D image where the object of interest is present. However, it does not require the model of the object as it is substituted with a set of surface elements, termed "surfels", that are constructed online while the RGB-D camera scans the scene. The surfels-based representation is used to track the object pose in real-time by minimizing the distance between the 3D measured points, extracted from the depth image, and the points sampled on the surfels. The cost function also includes a photometric term which matches the measured points with the sampled points according to the constancy of the brightness. Remarkably, this method is compatible with slow segmentation networks as it uses a coarser depth-based segmentation method when the output of the network is not available. In this regard, we recall that the method presented in the Chapter 5 of this Thesis similarly handles slow and delayed sources of object segmentation masks.

Differently from MaskFusion, it does not require an additional segmentation method when the masks are not available, rather it exploits the availability of high frame rate optical flow to propagate the measured masks.

Zhang et al. (2020) combine off-the-shelf object segmentation networks, partial point clouds of the object and the measured optical flow to estimate and track the pose and the linear velocity of the object. In order to estimate the pose of the object, the 3D points on the object at the previous frame are back-projected to the image plane and propagated with the measured optical flow. The same 3D points are also updated according to the pose to be optimized, back-projected and compared with the points propagated with the optical flow until their distance is minimized in least-squares sense. Remarkably, the proposed method also refines the measured optical flow such that the points distance is further minimized. On the other hand, it does not provide any strategy to handle possibly slow segmentation networks. The method presented in the Chapter 5 of this Thesis, named ROFT, is similar to this method as it uses the optical flow to reason on the motion of the object in the 2D space and update the object velocity accordingly. Differently from this method, ROFT can handle slow object segmentation sources. Furthermore, ROFT estimates the full 6D object velocity, i.e. both linear and angular, instead of the linear velocity solely. This method does not reach real-time performance and reaches a maximum frame rate of 6 fps.

Similarly to (Dubeau et al., 2020), the authors of (Masuda et al., 2021) propose to use events from an event camera to track the object pose and velocity. Differently from pure deep learning approaches, which directly regress the inter-frame relative pose using the accumulated events, the authors train a differentiable Multilayer Perceptron network (MLP) (Haykin, 1999) that compares the measured events with the object pose and velocity, producing an estimation error as output. Once the error network has been trained, an optimization problem is used to find the object pose and velocity which minimizes the error online. We report this method for the sake of completeness, given that this Thesis does not address the task of object pose tracking using event cameras.

Hybrid approaches for object pose refinement are also proposed in the literature. In (Periyasamy et al., 2019), the authors propose a pose refinement approach where the rendered and observed scene are not compared in the RGB space directly. Instead, the authors propose to map the predicted pose and the current observation to a common abstract feature space that is learned in a self-supervised manner from pixel correspondences. The path from the predicted pose to the feature space, for each pixel, is obtained using a differentiable renderer, i.e. a renderer that can be inverted. This feature makes possible to find the inter-frame pose transformation by minimizing online a simple pixel-wise error in the abstract space using a first-order gradient-based optimizer. The method reaches quasi real-time performance, i.e. 20 fps, making it feasible for tracking purposes.

Very recently, (Iwase et al., 2021) proposed a similar approach as in (Periyasamy et al., 2019) while increasing the achievable frame rate up to 92 fps. Furthermore, this method does not require the texture of the 3D models of the object at training time.

3.7.2 Object/end-effector pose tracking

In this last section, I present several recent works that use a combination of machine learning and Kalman filtering or optimization to track the pose of objects mainly during pushing and insertion tasks using vision and touch. One of the reasons for using machine learning in this context is the ever-growing diffusion of tactile sensors based on visual sensing or made from soft materials. The output of these sensors might be challenging to integrate in classical Kalman filtering and optimization pipelines because it is high-dimensional or difficult to be interpreted and modelled. Using machine learning allows extracting intermediate representations that can be integrated more easily in the tracking process.

In the Chapter 6 of this Thesis I considered a related scenario, i.e. that of tracking the sliding motion of an object between the fingers of a multi-fingered hand using soft tactile sensors. Although the kind of interaction between the object and the end-effector is different, i.e. sliding in lieu of pushing or object insertion, the reasons for using an hybrid approach still hold, i.e. extracting intermediate representations of the measured signal that are compatible with the tracking framework. To the best of our knowledge, the proposed method represents one of the first tentatives to track the sliding motion using an hybrid paradigm. On the other hand, model-based approaches which address a similar problem, e.g. (Costanzo, 2021; Costanzo et al., 2020; Liang et al., 2020), have been proposed.

Bayesian filtering-based methods

Lee et al. (2020) propose to use differentiable Kalman filters (Kloss et al., 2021) to learn how to optimally fuse different sensor modalities, such as vision and touch. Specifically, differentiable filters allow learning the dynamics and measurement models from data while preserving the algorithmic prior of the Kalman filter prediction and correction steps. The approach is tested within simulated 2D object pose tracking tasks involving planar pushing and in-hand manipulation. The proposed learned filters show the ability to adaptively select the correct sensing modality to be used, among vision and touch, when vision is available and the contact is missing or, vice versa, when the contact information is available and the object is occluded in the input images. We recall that the differentiable Kalman filtering paradigm has been used to develop the method presented in the Chapter 6. Specifically, it has been used to learn a possible measurement model of soft tactile sensors, which relates the output of the sensors with the object velocity.

Optimization-based methods

Yu and Rodriguez (2018b) propose to use the same optimization framework as in Yu and Rodriguez (2018a), which fuses visual and tactile measurements to track the pose of a pushed object, in a different scenario. Specifically, they consider the task of inserting an object picked by a suction cup into a tight space. Although the original framework is able to compensate weak visual sensing in presence of occlusions with local tactile sensing, the task of interest introduces further challenges. Indeed, the insertion task makes difficult to detect the position of the contact points between the object and the insertion space. The authors tackle this problem by training a Support Vector Machine (SVM) (Haykin, 1999) network to classify the type of contacts among a set of possible configurations. The additional information is integrated in the optimization problem by providing additional constraints which make the estimated object pose compatible with the contacts.

Recently, Suresh et al. (2021) re-proposed the framework for visuo-tactile tracking of a pushed object (Yu and Rodriguez, 2018a), mentioned above, and extended it to the task of joint pose and shape estimation. The problem is formulated as a factor graph optimization problem (Dellaert and Kaess, 2017) which associates visual and tactile information with the shape of the object that is updated by training a Gaussian process implicit surface (GPIS) (Williams and Fitzgibbon, 2006) online using tactile data.

More recent works exploit the availability of vision-based tactile sensors (Lambeta et al., 2020; Yuan et al., 2017). These sensors provide rich contact information in the form of RGB images capturing the local deformation of soft materials covering the sensor itself.

Sodhi et al. (2021a) use the Digit (Lambeta et al., 2020) vision-based tactile sensor to estimate the pose of a planar object being pushed by a manipulator. They first learn a suitable observation model that maps consecutive tactile images to the relative pose of the sensor. Then, they combine this information within a factor graph-based optimization (Dellaert and Kaess, 2017) approach for object pose tracking using tactile feedback alone.

Very recently, the same authors (Sodhi et al., 2021b) tackle the problem of tracking the 6D pose of small objects during in-hand manipulation using vision-based tactile sensors. They use again factor graph optimization to infer the object pose given measurements of the normals of the surface of the object in contact with the sensor. The normals are obtained from the tactile images using an adversarial network (Goodfellow et al., 2014) trained supervisedly using pairs of tactile and surface normal images obtained in simulation and from the real setup after manual labelling.

Chapter 4

Object segmentation-aided 6D Object Pose and Velocity Tracking

Object perception is one of the key challenges of modern robotics, representing a technology enabler to perform manipulation tasks effectively. Specifically, the estimation and tracking of the 6D pose of objects from camera images is useful to plan grasping actions while taking into account possible obstacles directly in the 3D world. Object manipulation is another important application, where the evolution of the object pose and velocity over time is fundamental for closing the control loop.

The problem of 6D object pose estimation and tracking from RGB(-D) images has been extensively addressed in the past both in the computer vision and robotics community, the latter focusing on solutions based on the use of Kalman and particle filtering techniques (Issac et al., 2016; Vezzani et al., 2017; Wüthrich et al., 2013).

Recently, deep learning techniques have been employed to solve the 6D object pose estimation problem (Li et al., 2018; Peng et al., 2019; Song et al., 2020; Wang et al., 2019; Xiang et al., 2018). The ever-growing availability of tools for synthetic data generation made possible to train these methods without the need to manually label massive amount of data as in the past. Although the proposed methods have shown impressive results on standard benchmarks, it is still not clear whether their performance is adequate enough for robotics. As an example, some of these methods have been successfully employed in one-shot grasping tasks (Wang et al., 2019). However, to the best of our knowledge, the possibility of closing the loop for tasks that require a continuous estimation of the object pose has not yet been thoroughly assessed.

Deep learning has been recently employed also in 6D object pose tracking methodologies. PoseRBPF (Deng et al., 2019) combines Rao-Blackwellized particle filtering with a deep neural network in order to capture the full object orientation distribution, even in presence of arbitrary symmetries. Its best performance is achieved only with a large amount of particles, leading to low frame rates. Moreover, the method requires a separate training for each object. se(3)-TrackNet (Wen et al., 2020a) instead departs from the hybrid paradigm and proposes a full end-to-end deep learning architecture. Specifically, it learns how to predict the change in the pose of the object between consecutive images, i.e. it tracks the pose of the object, given the current RGB-D observation and a synthetically rendered image of the object at the previous instant of time. Although the idea to completely replace the algorithmic prior given by Kalman and particle filtering with a neural network certainly deserves to be investigated, it is not clear how this method behaves if the residual pose between consecutive steps gets too large, for whatever reason, than the residuals seen at training time. Like PoseRBPF, se(3)-TrackNet needs to be trained separately for each object.

Given the above considerations, during the PhD I proposed a method (Piga et al., 2021a), called MaskUKF, that uses 2D binary masks and an Unscented Kalman Filter (UKF) (Wan and Merwe, 2000) to track the pose and the velocity of the object from depth observations. MaskUKF provides similar or superior performance to state-of-the-art 6D object pose estimation and tracking methods without the need to be trained separately for each object of interest. Indeed, the training requirements reduce to that of a deep neural network for 2D object segmentation of a set of objects. Moreover, the proposed method proves to be more robust to perturbations of the initial condition of the object pose with respect to the compared 6D object pose tracking methods.

The major contributions, discussed in the present chapter, are the following:

- The design and the implementation of a 6D object pose and velocity tracker which conveniently combines deep learning, only for the task of 2D object segmentation, with classical Kalman filtering and that does not need to be trained separately for each object of interest.

- The explanation on how to use depth information directly in an Unscented Kalman filtering setting and how to leverage recent advances in Sequential Unscented filtering to reach real-time performance.

- The rigorous benchmark of the method on the renowned YCB-Video dataset against state-of-the-art 6D object pose estimation methods and tracking methods, with particular attention to the usage of the same initial conditions for all the compared methods.

- The design of a heuristic 1-parameter procedure to identify and reject outliers in the depth observation which increases the tracking performance.
- The execution of experiments on a humanoid robot platform in simulation which shows the importance of joint pose and velocity tracking in a robotic setting where closed loop control is employed and the necessity to prefer tracking methods over pose estimation in this setting.

The proposed method has been also compared against a baseline consisting in a simpler architecture which uses an Iterative Closest Point (ICP) (Besl and McKay, 1992) registration algorithm from Rusu and Cousins (2011) and the masks from the segmentation network.

A video of the experiments is available online[1]. The code used for the experiments is made publicly available for free with an Open Source license online[2].

The remainder of this chapter is organized as follows. In Sec. 4.1, I present the proposed method in detail with the help of the theory from Chapters 2 and 3. The experimental results of the evaluation on the YCB-Video dataset are provided in Sec. 4.2. Finally, in Sec. 4.3, I discuss the results of the closed loop control experiments with a humanoid robot platform in simulation.

4.1 Description of the method

The proposed method tracks the pose T_t and the velocity $\mathcal{V}_t^m$ of an object of interest $\mathcal{O}$ using a Unscented Kalman Filter given a partial point cloud $C(M_t, D_t)$ of the object defined as in Eq. (3.29). It is recalled that D_t is the depth image at time t while M_t is a set of pixel locations in the input RGB image I_t that contain the object of interest.

In the scenario considered for this approach, it is assumed that an *object segmentation* algorithm is available in order to identify the object of interest within the image I_t and provide the set of locations M_t, in the following referred to as the object segmentation mask.

In the remainder of this section, all the components of the tracking algorithm are described.

[1]https://www.frontiersin.org/articles/10.3389/frobt.2021.594583/full#supplementary-material
[2]https://github.com/hsp-iit/mask-ukf

4.1.1 Definition of the state and the measurements

The state to be tracked x_t is defined as

$$x_t = \begin{bmatrix} p_t \\ v_t^m \\ o_t \\ \dot{o}_t \end{bmatrix} \in \mathbb{R}^{12}, \tag{4.1}$$

where

- $p_t \in \mathbb{R}^3$ is the positional part of the pose T_t;
- $o_t \in \mathbb{R}^3$ is an Euler ZYX parametrization of the rotational part R_t of the pose T_t;
- v_t^m is the linear velocity part of the mixed velocity $\mathcal{V}_t^m$;
- $\dot{o}_t := \frac{\mathrm{d}o_t}{\mathrm{d}t} \in \mathbb{R}^3$ is the vector of the angular rates.

It can be easily shown (Stevens et al., 2015) that the angular velocity part ω_t of the mixed velocity $\mathcal{V}_t^m$ can be obtained as a linear function of the angular rates

$$\omega_t = \begin{bmatrix} 0 & -s_\phi & c_\phi c_\chi \\ 0 & c_\phi & s_\phi c_\chi \\ 1 & 0 & -s_\chi \end{bmatrix} \dot{o}_t, \tag{4.2}$$

where ϕ, χ and ψ are the Euler angles composing o_t and c_α, s_α are shorthands for $\cos(\alpha)$ and $\sin(\alpha)$ for a given angle α, respectively. We remark that the matrix in Eq. (4.2) is singular if $c_\chi = 0$. Nevertheless, in the following developments we will not use the inverse map from the angular velocity to the angular rates.

The point cloud set $C(M_t, D_t)$ is used as a measurement

$$z_t = \begin{bmatrix} z_{1,t}^T & \cdots & z_{j,t}^T & \cdots & z_{L_t,t}^T \end{bmatrix}^T \in \mathbb{R}^{3L_t}, \tag{4.3}$$

where each subvector $z_{j,t} \in \mathbb{R}^3$ is the j-th vector in the set $C(M_t, D_t)$ and L_t is the cardinality of the set at time t. The cardinality changes over time as the number of points on the object surface that are visible from the camera viewpoint depends on the object pose and might vary due to occlusions. We remark that the adopted tracking framework, i.e. the Unscented Kalman filter, naturally handles time-varying measurement models.

4.1.2 Choice of the filtering algorithm

In order to track the state of the object, we adopted the Unscented Kalman Filter for the following reasons:

- the ability to handle model and measurement functions f and h even if they are not provided with an analytical expression (as in this scenario for the measurement function h, later described);
- the availability of a sequential variant of the algorithm, discussed in Sec. 2.2.3, that efficiently processes high-dimensional measurements as in the case of point clouds;
- the recognized superiority (Julier and Uhlmann, 2004; Wan and Merwe, 2000) over alternatives based on linearization, e.g. the Extended Kalman Filter, in terms of unbiasedness and consistency.

Considering the above motivations, I adopted the Algorithm 7 "Sequential Unscented Kalman filter" to accomplish the task at hand. Furthermore, given that the state x_t contains several angular components, i.e. the Euler angles, the algorithm has been modified to take them into account as in the Algorithm 8 "Unscented Kalman filtering using Euler angles".

4.1.3 Motion model

The motion model accounts for the prior knowledge about the motion of the object. We adopted the White Noise Acceleration model (WNA) (Bar-Shalom et al., 2002) from the tracking literature.

The WNA model assumes that the second derivative of the position $\ddot{p}(t) \in \mathbb{R}^3$ is driven by a white noise process $w_r(t)$ with assigned power spectral density $Q_p \in R^{3\times 3}$. The same model has been used for the rotational part of the state, with power spectral density Q_o, resulting in

$$\begin{aligned} \ddot{r}(t) &= \omega_r(t), \\ \ddot{o}(t) &= \omega_o(t). \end{aligned} \tag{4.4}$$

In order to obtain a discrete time model as in Eq. (2.7), we discretized the WNA model assuming constant linear velocity and angular Euler rates within each sampling period Δ_t. The final model is as follows

$$\begin{aligned} x_t &= f(x_{t-1}, w_{t-1}) \\ &= F x_{t-1} + w, \\ w &\sim \mathcal{N}(0, Q), \end{aligned} \tag{4.5}$$

where F is the state transition matrix, that depends on Δ_t, and Q is the process noise covariance matrix depending on Q_p, Q_o and Δ_t. The complete expression of the matrices is as follows:

$$F = \begin{bmatrix} I_3 & I_3\Delta_t & 0 & 0 \\ 0 & I_3 & 0 & 0 \\ 0 & 0 & I_3 & I_3\Delta_t \\ 0 & 0 & 0 & I_3 \end{bmatrix} \in \mathbb{R}^{12\times 12}, \tag{4.6}$$

$$Q = \begin{bmatrix} Q_p\frac{\Delta_t^3}{3} & Q_p\frac{\Delta_t^2}{2} & 0 & 0 \\ Q_p\frac{\Delta_t^2}{2} & Q_p\Delta_t & 0 & 0 \\ 0 & 0 & Q_o\frac{\Delta_t^3}{3} & Q_o\frac{\Delta_t^2}{2} \\ 0 & 0 & Q_o\frac{\Delta_t^2}{2} & Q_o\Delta_t \end{bmatrix} \in \mathbb{R}^{12\times 12}. \tag{4.7}$$

4.1.4 Measurement model

The specification of the measurement model accounts for the definition of the function h in Equation (2.8). The role of h is to establish the relationship between the state x_t and the measurements that we expect to observe when the object $\mathcal{O}$ is in configuration x_t. To this end, given a triangle mesh description of the object (see Sec. 3.2) and a fixed list of L_m points $\{A_j\}$ sampled on the mesh, the predicted point cloud of the object in configuration x_t

$$C(\{A_j\}, x_t) = \{a_{j,t}(A_j, x_t)\},$$

as defined in Eq. (3.31), is considered. For the sake of clarity, we omit the explicit dependence of the coordinates $a_{j,t}$ from the point A_j and the state x_t in the following.

In order to actually implement a Kalman filtering approach, it is required to decide how to compare the L_t measured points coordinates $z_{j,t} \in \mathbb{R}^3$ with the L_m predicted points coordinates $a_{j,t}$, i.e. to solve the problem of associating the elements belonging to the two sets. In this respect, we adopted a likelihood field-based model (Thrun et al., 2005) for which the overall point cloud vector z_t is considered as an ensemble of independent points $z_{j,t}$ each distributed according to a normal distribution

$$z_{j,t} \sim \mathcal{N}(\pi_{j,t}, \sigma_j^2 I_3), \tag{4.8}$$

where the mean $\pi_{j,t}$ is defined as the point $a_{j,t}$ having minimum distance from the real measured point $z_{j,t}$

$$\pi_{j,t}(x_t) = \arg \min_{a_{k,t}} \|z_{j,t} - a_{k,t}\| \,. \tag{4.9}$$

The above equations can be easily cast into the following measurement model

$$\begin{aligned} z_t &= h_t(x_t) + \nu_t, \\ \nu_t &\sim \mathcal{N}(0, R_t), \end{aligned} \tag{4.10}$$

where

$$\begin{aligned} h_t(x_t) &= \begin{bmatrix} \pi_{1,t}(x_t)^T & \dots & \pi_{L_t,t}(x_t)^T \end{bmatrix}^T, \\ R_t &= \mathrm{diag}(\sigma_1^2 I_3, \dots, \sigma_{L_t}^2 I_3). \end{aligned} \tag{4.11}$$

In summary, given the object in configuration x_t, the point cloud that we expect to observe is predicted by projecting the measured point cloud z_t on the surface of the object. This choice represents a possible, although not unique, solution to the problem of associating the measured point cloud with the predicted one. Nonetheless, it guarantees that the size of the sigma points of the measurements $\mathcal{Z}_t^{(i)}$, hence the size of the predicted mean of the measurement, $\mu_{z,t}$, is the same as the size of the measurement vector z_t. This assumption is required to evaluate the Kalman innovation and perform the correction step. Alternative solutions, e.g. projecting the points $\{A_j\}$ sampled on the mesh onto the measured point cloud, would possibly produce vectors of incompatible sizes thereby making impossible to execute the algorithm.

It is worth mentioning that the resulting measurement model is nonlinear with additive noise and the noise covariance matrix R_t has a diagonal block structure as required by the hypothesis of the Sequential Unscented Kalman Filter (see Algorithm 7).

Implementation of the measurement function

In order to implement the Unscented Kalman correction step using the measurement equation (4.11), it is required to solve the $L_t(2n + 1)$ optimization problems corresponding to the evaluation of the L_t projections $\pi_{t,j}$ in Eq. (4.11) for each of the $2n+1$ sigma points as per the Eq. (2.82).

To reduce the computational cost of these evaluations, we approximate the projections using a Nearest Neighbor approach. To this end, the points $\{A_j\}$ are inserted in a 3D k-d tree. Then, for each sigma point $\mathcal{X}_t^{-(i)}$, the measurements $z_{j,t}$ are transformed to the reference frame of the object in configuration $\mathcal{X}_t^{-(i)}$ and an efficient Nearest neighbour query is executed on the tree for each transformed measurement. The resulting projections are then transformed back to the camera reference frame according to the pose $\mathcal{X}_t^{-(i)}$.

We notice that the proposed measurement function is only available by means of numerical evaluations, i.e. the output of the nearest neighbour search, but it does not have an analytical form. This choice is made possible by the adoption of the Unscented Kalman filter that, differently from other alternatives such as the Extended Kalman filter, does not require the evaluation of the Jacobian of the measurement function with respect to the state which might be intractable in this case.

We remark that the proposed approximation implicitly assumes that representing the object model as a set of 3D points $\{A_j\}$ is sufficient to accurately project the measured point cloud on the surface of the object, as required by the measurement model described in Sec. 4.1.4. In order to quantify how the approximation might impact on the tracking performance, we also considered implementing the projections in Eq. (4.11) by taking advantage of the full triangle mesh description of the object model. Concretely, we adopted the Computational Geometry Algorithms Library (CGAL) (The CGAL Project, 2022). This library allows evaluating the point in the union of all the triangular primitives of the mesh, i.e. its trianglular faces, which is closest to a given query point. A comparison of the overall performance obtained using the two approaches, i.e. the k-d tree and CGAL, is discussed in Sec. 4.2.10.

4.1.5 Outliers rejection

Point clouds from modern RGB-D sensors are characterized by non Gaussian noise and outliers that violate the hypotheses of the model in Equation (4.10) which might induce biases in the estimates in Gaussian filters such as the Kalman filter (Issac et al., 2016). To tackle this, it is possible to try identifying the outliers and exclude them from the measurement vector z_t.

In this project, I proposed to identify the outliers by picking pairs of points on the point cloud z_t, $z_{i,t}$ and $z_{j,t}$, and their projections, π_i and π_j respectively, on the surface of the object in a configuration corresponding to the previous state x_{t-1}.

Under the assumption that the tracking process has already reached a steady state condition, it is expected that the point cloud at time t fits very closely the surface of the object in the estimated configuration at time $t-1$. As a result, the distance d_{ij} between the two points and the distance d^{π}_{ij} between their projections should be almost preserved if both $z_{i,t}$ and $z_{j,t}$ are not outliers, i.e. belong to the surface of the object. Then, by comparing the absolute difference between d_{ij} and d^{π}_{ij} to a threshold

$$|d_{ij} - d^{\pi}_{ij}| > \delta^{\text{outlier}}, \tag{4.12}$$

it is possible to identify the presence of outliers. In such a case, the point among $z_{i,t}$ and $z_{j,t}$ that has higher distance from its projection is marked as an outlier. I.e., if

$$\|z_{i,t} - \pi_i\| > \|z_{j,t} - \pi_j\|, \tag{4.13}$$

then $z_{i,t}$ is marked as the outlier and vice versa.

Since outliers that violate the additive Gaussian noise hypothesis are usually distributed relatively far from the actual surface of the object, we propose, as a heuristic, to choose candidate pairs of points as the combination of one point $z_{i,t}$ and its furthest point $z^{f}_{i,t}$ on the point cloud for each point on the point cloud z_t. Every time an outlier is found, it is removed from the point cloud and the procedure is repeated until all points have been visited. An efficient evaluation of the points $z^{f}_{i,t}$ is obtained using the algorithm proposed in (Curtin and Gardner, 2016).

4.1.6 Handling of missing or invalid measurements

The tracking process depends on the availability of the segmentation masks M_t that are required to extract the point cloud of the object from the depth image D_t. The tracking process is initiated using the first segmentation mask M_0. Then, it is assumed that a new mask M_t is available at each time step t. In case the mask is not available at time t, the most recent available mask is used instead.

Sometimes it might happen that the mask M_t is empty because the object is totally occluded in the scene. In this case the point cloud of the object is not available. On the other hand, it might also happen that the point cloud is available but the majority of the points are not valid. This condition usually verifies if the object of interest has semi-transparent parts or parts of the texture that are not diverse enough for the depth estimation process. In all these cases, the measurements z_t are not reliable or are too few to be informative enough. In order to actually determine if the measurements are too few, we evaluate the ratio $\frac{L_t^{\text{valid}}}{L_t}$ between the number of valid points and the total number of points at time t, and compare it against a threshold ϵ^{valid}.

Such an absence of measurements is usually counteracted by the usage of the Kalman prediction step only. However, if the object is undergoing a motion characterized by moderate to high velocities, this approach is discouraged and might lead to unbounded estimates and track loss.

Alternatively, in this project I proposed to sample a virtual point cloud on the mesh of the object in a configuration corresponding to the previous estimate x_{t-1}. The resulting measurement will be indicated as $z_t^{\text{virtual}}(x_{t-1})$. We remark that the virtual point cloud is not kept fixed, instead a new virtual cloud is sampled at each time t as long as the absence of measurements endures. The advantage of this choice is that the estimated velocities are driven to zero in case the absence of information persists for multiple frames allowing the tracking process to safely recover when the measurements are available again. We remark that this procedure does not arbitrarily alter the state of the filter which would impair the correctness of the state covariance associated with the state.

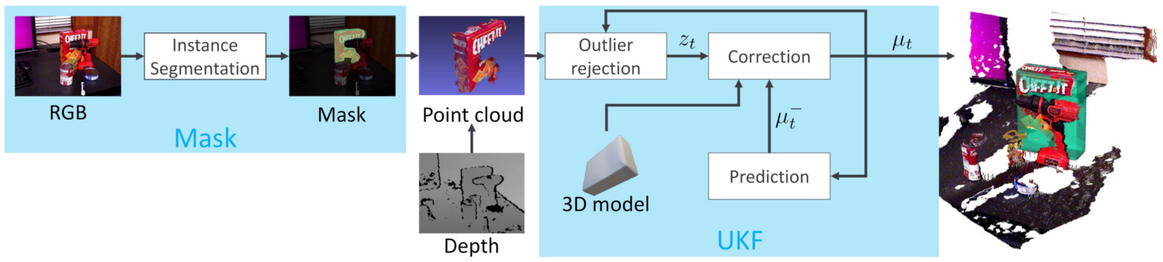


Figure 4.1 Illustration of the MaskUKF framework for 6D object pose and velocity tracking. For each RGB-D frame, an object segmentation algorithm provides the segmentation mask of the object of interest. The mask is combined with the depth frame to produce a partial point cloud of the object that is further refined to remove possible outliers. The resulting measurements are fed into an Unscented Kalman filter to update the belief of the object pose and velocity.

MaskUKF: Instance Segmentation-aided 6D Object Pose and Velocity Tracking

```
 1: procedure MaskUKF($\mu_0, P_0, \delta^{\text{outlier}}, \epsilon^{\text{valid}}$)
 2:     $\mu_{t=0} = \mu_0$
 3:     $P_{t=0} = P_0$
 4:     for each $(I_t, D_t)$ do
 5:         $M_t \leftarrow$ InstanceSegmentation($I_t$) or last mask available
 6:         $z_t \leftarrow C(M_t, D_t)$
 7:         if $M_t$ empty or $(L_t^{\text{valid}}/L_t) < \epsilon^{\text{valid}}$ then
 8:             $z_t = z_t^{\text{virtual}}(\mu_{t-1})$                         ▷ See 4.1.6
 9:         else
10:             $z_t \leftarrow$ OutlierRejection($z_t, \delta^{\text{outlier}}$)              ▷ See 4.1.5
11:         end if
12:         $(\mu_t, P_t)$ = SequentialUKF($f(\cdot), h_t(\cdot), \mu_{t-1}, P_{t-1}, z_t$)   ▷ See Algorithms 7, 8
13:                                                              ▷ See 4.1.3 for $f(\cdot)$
14:                                                              ▷ See 4.1.4 for $h_t(\cdot)$
15:     end for
16: end procedure
```

4.1.7 Resulting algorithm

The proposed algorithm for 6D object pose and velocity tracking is summarized in the table entitled “MaskUKF: Instance Segmentation-aided 6D Object Pose and Velocity Tracking”. The overall framework is also depicted in Fig. 4.1.

4.2 Experimental evaluation

This section reports on the performance achieved by the proposed method on the standard dataset YCB-Video (Xiang et al., 2018) and provides comparisons with state-of-the-art algorithms for 6D object pose tracking. Following prior works (Deng et al., 2019; Wen et al., 2020a), the comparison is also extended to 6D object pose estimation algorithms as they solve a related task, although fundamentally different as explained in Sec. 3.1.2.

The analysis includes the standard ADI-AUC metric (Xiang et al., 2018), as well as the pose RMSE (Bar-Shalom et al., 2002) tracking errors. Furthermore, the evaluation includes an ablation study where the outlier rejection mechanism in Sec. 4.1.5 is disabled in order to assess its effectiveness. Qualitative results are also provided in Figs. 4.2, 4.3, 4.4 and 4.5.

4.2.1 Description of the evaluation dataset

The YCB-Video dataset features RGB-D video sequences of 21 objects from the YCB Object and Model Set (Calli et al., 2015) under different occlusion conditions. The dataset contains 92 RGB-D video sequences recorded at 30 fps, split in a training set of 80 videos and a testing set of 12 videos from which 2949 key-frames are extracted in order to evaluate performance metrics. Every frame is annotated with segmentation masks and 6D object poses.

4.2.2 Description of the compared algorithms

Pose tracking methods

PoseRBPF combines a deep convolutional autoencoder for implicit orientation encoding (Martin et al., 2018) with a particle filter in order to track the position and the orientation of the object over time from RGB-D observations. PoseRBPF requires a textured 3D mesh model of the object both at training and test time. The autoencoder needs to be trained separately for each object of interest. In order to be initialized, the algorithm requires the 2D coordinates of the center of a bounding box enclosing the object in the image. This algorithm can be considered as a representative of the category of hybrid approaches which fuse Deep Learning and Bayesian filtering, similarly to the proposed method.

The comparison has been extended to more recent approaches based on Deep Learning solely. One of these methods is **se(3)-TrackNet**, a deep convolutional neural network that tackles the 6D object pose tracking problem end-to-end. Given a new RGB-D image and a prediction of the RGB and depth images corresponding to the previous estimate, obtained using a rendering software, the network predicts a relative pose of the object between the previous and next time steps. se(3)-TrackNet requires a textured 3D mesh model both at training and test time. It needs to be trained separately for each object of interest. As regards the initialization, it requires the full 6D pose of the object.

It is worth recalling that the proposed method, **MaskUKF**, requires an instance segmentation algorithm that can provide the segmentation masks of the object of interest. This component is typically implemented as a deep neural network to be trained once, for all the objects in a set, but not separately for each object. Furthermore, recent advances in object segmentation, such as (Ceola et al., 2021), have shown that it is possible to learn how to segment a new object, not considered in the original training set, with an online training procedure taking a few seconds. MaskUKF requires a non-textured 3D mesh model at test time. For the initialization stage, it requires the full 6D pose of the object.

Pose estimation methods

DenseFusion (Wang et al., 2019) is a deep convolutional neural network that fuses segmentation, RGB and depth information at the pixel level in order to regress the 6D pose of the object. Similarly to the other methods, it requires a 3D mesh model of the object at training time but it does not require it at testing time. It also needs the object segmentation in order to extract the relevant part of the depth image, similarly to MaskUKF, and of the RGB image. The algorithm is trained once, for all the objects of interest.

DOPE (Tremblay et al., 2018) is another example of deep convolutional neural network for 6D object pose estimation. Differently from DenseFusion, it uses the RGB information only and avoid regressing the pose directly. Instead, it predicts a set of 2D keypoints belonging to the object and whose 3D coordinates in the frame attached to the object mesh are known. The 2D-3D correspondences are then used to estimate the pose of the object resorting to the Perspective-n-Point (PnP) algorithm. The PnP algorithm is a classic algorithm that is commonly used to estimate the pose of a moving camera given a set of 3D points in the world and their corresponding 2D projections in the image. The algorithm requires a textured 3D mesh model at training time. It needs to be trained separately for each object of interest.

Both methods do not require an initial condition as they are 6D object pose estimation algorithms.

Baselines

As baseline, the classic point cloud registration algorithm ICP (Besl and McKay, 1992) from (Rusu and Cousins, 2011) has been considered. In order to have a fair comparison with the proposed method, the adopted ICP algorithm is assisted by segmentation in order to extract the relevant part of the point cloud of the entire scene. It is also equipped with its own outlier rejection procedure, similarly to MaskUKF.

Given that the ICP algorithm needs to be initialized with the 6D pose of the object, at each iteration, we use the estimate from the previous frame as the initialization. The ICP algorithm requires a non-textured 3D mesh model of the object at test time.

4.2.3 Description of the experimental setup

Source of segmentation masks

The proposed method relies on the availability of the segmentation masks M_t of the object of interest as input. As source of segmentation masks, we considered two alternatives. The first is the semantic segmentation network from the PoseCNN (Xiang et al., 2018) framework as trained by the authors on all the 200k+ frames of the YCB-Video dataset. The second alternative is the Mask R-CNN (He et al., 2017) network, a consolidated region-based approach for object detection and segmentation, trained using 50k frames from the same dataset.

The reason for using PoseCNN is that it is also adopted in DenseFusion. In order to have a fair comparison, we use the same segmentation.

The reason for using MaskR R-CNN is that, differently from the segmentation network from PoseCNN, it has not been proposed in the pose estimation and tracking communities and represents an off-the-shelf and commonly adopted solution for object segmentation.

Source of points for point cloud prediction

The algorithm relies on the availability of a list of points $\{A_j\}$ sampled on the surface of the object $\mathcal{O}$, required to make a prediction of the object point cloud given a candidate pose of the object. To this end, we used the set of uniform point clouds sampled on the mesh of the objects, that are provided within the YCB-Video dataset. Each point cloud comprises 2621 3D points.

Initialization

The initialization of the tracking process is done resorting to the ground truth pose of the object. However, given that the ground truth is not available in reality and in order to reproduce a more reasonable scenario, we perturbed the ground truth pose with a displacement of 5 cm along each Cartesian axis and a variation of 10 degrees for each Euler angle. For the sake of completeness, we also provide the results obtained when initializing with the actual ground truth, i.e. without the additional perturbation.

We remark that the same initial condition has been used for all the tracking algorithms considered in the comparison and for the baseline ICP. When required, we made use of the software implementation of the compared algorithms, as provided by their authors, to execute the experiments using different initial conditions.

In the proposed scenario, re-initialization is not allowed during the tracking process for all the algorithms. For this reason, we disabled the re-initialization mechanisms used within PoseRBPF. Furthermore, as can be seen from the software implementation of PoseRBPF, it executes the initialization stage repeatedly at time $t = 0$ and subsequent times until the rotational error, evaluated with respect to the ground truth, is below a given threshold. Given that the ground truth is usually not available in a real tracking scenario, we disabled this mechanism, i.e. the algorithm is initialized once as all the others. Nonetheless, for the sake of completeness, we report the results obtained when using the above mentioned "multistage" initialization step.

Choice of the algorithm parameters

In Table 4.1 we report the parameters of MaskUKF that we used for the experiments.

The values of the entries in the covariance matrices P_0, Q_p, Q_r and R_t were chosen empirically in order to ensure a fast response of the filter while limiting, as much as possible, the effect of the noise from the measured point cloud on the output.

Table 4.1 Parameter set for MaskUKF for the experiments on the YCB-Video dataset.

Parameter	Values
$P_{0\,(1:3,1:3)}$	$\mathrm{diag}(10^{-2}, 10^{-2}, 10^{-2})\,\mathrm{m}^2$
$P_{0\,(4:6,4:6)}$	$\mathrm{diag}(10^{-2}, 10^{-2}, 10^{-2})\,(\mathrm{m/s})^2$
$P_{0\,(7:9,7:9)}$	$\mathrm{diag}(10^{-2}, 10^{-2}, 10^{-2})\,\mathrm{rad}^2$
$P_{0\,(10:12,10:12)}$	$\mathrm{diag}(10^{-2}, 10^{-2}, 10^{-2})\,(\mathrm{rad/s})^2$
Q_p	$\mathrm{diag}(0.1, 0.1, 0.1)\,(\mathrm{m/s})^2/\mathrm{s}$
Q_r	$\mathrm{diag}(0.2, 0.2, 0.2)\,(\mathrm{rad/s})^2/\mathrm{s}$
R_t	$\mathrm{diag}(0.001, \cdots, 0.001)\,\mathrm{m}^2$
$\delta^{\mathrm{outlier}}$	$0.01\,\mathrm{m}$
$\epsilon^{\mathrm{valid}}$	0.2

As regards the outlier rejection mechanism, we set $\delta^{\mathrm{outlier}}$ to 1 cm. Ideally, this parameter should be as small as possible to limit the number of false negatives, i.e. points that are outliers but that are considered as valid. On the other hand, due the noise in the measured point cloud, a too small value might lead to false positives, i.e. points that are valid but that are erroneously classified as outliers. We found experimentally that setting the threshold to 1 cm provides a good balance between the two phenomena.

4.2.4 Description of the adopted metrics

Following prior work Wang et al. (2019), the ADI-AUC metric (see Sec. 3.4.3), with the threshold ϵ set to 10cm is reported for each object and for the union of all the objects, in order to determine the overall performance of the algorithms. Although more strict, the ADD-AUC metric has not been considered, instead, as the proposed algorithm, MaskUKF, and ICP do not use the RGB information. Hence, they cannot distinguish between different orientations of objects that would look identical according to the information provided by the depth only.

We evaluated the ADI-AUC metric on the 2949 key-frames provided by the YCB-Video dataset.

As explained in Sec. 3.4.4, it is also important to compare the algorithms using the RMSE metric from the filtering and tracking literature. In this respect, we used the positional and rotational RMSE errors, defined in Sec. 3.4.1. For objects that have a pure cylindrical shape, the part of the angular error along the symmetry axis has been neglected. In fact, algorithms that only use the depth information, as MaskUKF and ICP, cannot infer the rotation along that axis. A similar reasoning was considered for texture-less objects, whose rotation along the axes of symmetry, if any, cannot be estimated even using RGB information.

Table 4.2 Quantitative evaluation on YCB-Video dataset key-frames using the ADI-AUC metric. A bold entry indicates a better result. For the tracking algorithms, the best results are decided taking into account the experiment initialized with the perturbed ground truth.

metric	ADI-AUC, $\epsilon = 10$ cm														
method	DOPE	Dense Fusion	se(3)-TrackNet		PoseRBPF (200 particles)			ICP		MaskUKF		ICP		MaskUKF	
segm.	-	Pose CNN	-		-			PoseCNN				Mask R-CNN			
init.	-	-	pert.	gt	pert.	gt	gt (multi)	pert.	gt	pert.	gt	pert.	gt	pert.	gt
002	-	**96.4**	0.5	96.5	93.2	95.4	95.9	95.0	95.0	96.1	96.1	93.9	93.9	96.1	96.3
003	65.0	95.5	93.1	97.2	78.6	95.2	95.2	**97.0**	97.0	96.7	96.8	96.3	96.3	89.2	88.8
004	84.9	97.5	31.1	98.1	57.0	95.8	95.8	97.6	97.6	**98.1**	98.3	97.0	97.5	**98.1**	98.2
005	84.0	94.6	3.9	89.0	67.7	89.8	90.2	94.3	94.3	**96.7**	96.9	95.8	95.9	94.1	94.2
006	90.7	97.2	18.9	94.4	45.0	96.3	96.3	97.8	97.8	**98.2**	98.3	98.0	98.0	**98.2**	98.3
007	-	96.6	31.5	91.1	37.6	82.5	82.5	96.9	96.9	94.1	96.7	**97.2**	97.3	96.2	94.4
008	-	**96.5**	3.5	98.5	87.4	89.1	86.2	91.0	93.4	93.8	97.2	91.0	91.1	75.7	79.7
009	84.8	98.1	2.5	98.3	11.5	97.8	97.8	97.7	97.7	98.3	98.3	98.5	98.5	**98.7**	98.7
010	21.3	91.3	17.6	80.2	14.4	53.9	59.7	87.5	88.5	93.2	93.3	84.1	87.7	**94.0**	95.9
011	-	96.6	0.5	97.0	59.6	94.5	95.2	97.2	97.2	97.6	97.7	96.6	97.6	**97.7**	97.8
019	-	97.1	97.5	97.7	79.8	95.0	95.0	**97.7**	97.7	96.5	96.6	97.6	97.6	96.4	96.5
021	-	95.8	44.7	97.2	71.0	94.5	94.5	94.9	95.9	**97.0**	97.1	96.1	96.4	**97.0**	97.1
024	-	88.2	84.5	96.5	84.5	91.7	90.4	97.3	97.3	**97.6**	97.8	94.2	96.1	97.5	97.6
025	-	97.1	0.7	96.7	81.5	97.2	97.2	94.5	97.3	97.1	97.1	94.5	97.3	**97.4**	97.6
035	-	96.0	2.3	97.4	53.9	96.8	96.8	**97.5**	97.5	96.5	97.4	97.4	97.4	89.8	97.5
036	-	89.7	1.3	96.2	89.4	91.6	85.7	92.4	92.4	**95.0**	95.0	91.7	91.7	94.7	94.9
037	-	95.2	1.1	97.1	14.0	95.9	95.9	84.8	85.0	**96.5**	96.6	86.0	86.2	95.7	95.7
040	-	**97.5**	95.0	95.3	67.1	95.2	96.3	91.5	96.4	96.8	96.9	91.1	96.1	95.6	94.9
051	-	72.9	1.4	96.8	**92.6**	92.9	94.0	53.9	53.8	70.0	81.6	43.1	42.2	47.1	47.2
052	-	69.8	63.2	94.3	51.7	67.2	67.2	76.1	74.0	80.3	80.2	**80.8**	81.4	35.7	41.4
061	-	92.5	0.5	7.4	92.3	96.6	96.4	84.1	98.2	**97.7**	97.8	93.6	96.9	97.1	97.2
ALL	-	93.1	30.3	92.8	63.9	89.9	90.2	91.9	92.6	**94.2**	95.3	91.5	92.3	89.5	90.4

We evaluated the RMSE metric on all the frames of each considered sequence from the YCB-Video dataset, as it is common in the filtering and tracking literature.

4.2.5 Results: ADI metric

Table 4.2 shows the evaluation for all the 21 objects in the YCB-Video dataset using the ADI-AUC metric.

For the tracking algorithms, the "init." row indicates the type of initialization. Specifically, "pert." indicates the initialization with the perturbed ground truth, "gt" with the actual ground truth and "gt (multi)" indicates the PoseRBPF initialization strategy which involves multiple frames.

Comparison with pose tracking algorithms

When considering both segmentation sources and the initialization with the perturbed ground truth, MaskUKF outperforms se(3)-TrackNet and PoseRBPF with an increase in performance of approximately 300% and 150%, respectively, on average.

When considering both segmentation sources and the initialization with the actual ground truth, the performance is instead comparable but still better for MaskUKF when considering the PoseCNN segmentation.

These results suggest that MaskUKF is more robust to perturbations of the initial conditions that, instead, cause an important drop in performance for the other tracking methods considered in the comparison.

As regards the multistage initialization of PoseRBPF, it can be seen that, according to this metric, it does not provide any sensible improvement in performance.

Comparison with pose estimation algorithms

When considering the segmentation from PoseCNN, MaskUKF outperforms the DenseFusion framework that uses the same segmentation source. Even if the increase in performance is moderate, the training requirements of MaskUKF are much less demanding. Indeed, it can track the object pose given a suitable initial condition and a trained segmentation network which does not require 6D pose labels at training time and solves a simpler task than estimating the 6D pose of an object given an image.

When comparing with DOPE, for the objects for which the results are available, MaskUKF performs better for all the objects independently of the segmentation source. We remark that DOPE does not use the depth information, while all the other methods do. The results are useful to identify the improvement in performance when using RGB-D information or depth information alone, as in the case of MaskUKF and ICP.

Comparison with the baseline

When considering the segmentation from PoseCNN, MaskUKF outperforms the ICP baseline when using both the perturbed or the actual ground truth for initialization. When considering the Mask R-CNN segmentation, both algorithms experience a drop in performance that is within 5 ADD-AUC units. We verified that this is due to missing detections or completely wrong segmentation. When using Mask R-CNN, ICP performs better than MaskUKF, however the difference is no more than 2 ADD-AUC units.

In general, the results suggest that the performance might vary with the specific segmentation algorithm that is adopted. Hence, the final user might increase the performance of MaskUKF and ICP by using a better segmentation network if at disposal. The same reasoning cannot be applied to end-to-end tracking methods, such as se(3)-TrackNet, that are not as modular as MaskUKF and ICP from this perspective. Indeed, they would require to be retrained for the entire 6D tracking task and for

Table 4.3 Quantitative evaluation on all the frames of the YCB-Video dataset using the positional RMSE metric. A bold entry indicates a better result. For the tracking algorithms, the best results are decided taking into account the experiment initialized with the perturbed ground truth.

metric	Positional RMSE (cm)														
method	DOPE	Dense Fusion	se(3)-TrackNet		PoseRBPF (200 particles)			ICP		MaskUKF		ICP		MaskUKF	
segm.	-	Pose CNN	-		-			PoseCNN				Mask R-CNN			
init.	-	-	pert.	gt	pert.	gt	gt (multi)	pert.	gt	pert.	gt	pert.	gt	pert.	gt
002	-	**0.5**	41.8	0.5	0.7	0.4	0.5	0.7	0.8	0.6	0.5	1.1	1.1	0.6	0.5
003	23.6	0.9	3.6	0.3	7.4	0.7	0.7	**0.4**	0.4	0.7	0.5	0.7	0.7	3.6	3.7
004	8.5	**0.3**	27.3	0.3	8.3	0.6	0.6	0.4	0.4	0.4	0.3	0.7	0.5	0.5	0.3
005	14.7	**0.5**	24.4	7.3	7.7	4.2	4.1	1.6	1.5	1.2	1.1	1.5	1.5	3.2	3.3
006	1.9	0.4	26.4	1.5	8.4	0.3	0.3	0.4	0.4	0.5	0.2	**0.2**	0.2	0.3	0.2
007	-	**0.4**	220	1.2	10.5	3.2	3.2	0.6	0.4	1.3	0.6	0.5	0.4	0.8	1.0
008	-	**0.7**	32.3	0.2	2.4	1.6	2.6	1.4	1.3	1.5	0.6	1.9	1.8	8.9	8.5
009	2.7	**0.3**	389	0.2	16.0	0.3	0.3	0.6	0.6	0.5	0.4	**0.3**	0.3	0.4	0.2
010	27.1	7.5	33.0	39.2	15.6	11.1	10.6	4.1	4.0	**1.6**	1.6	4.9	4.8	2.2	1.4
011	-	**0.7**	218	0.4	5.8	0.9	0.7	**0.7**	0.7	0.9	0.9	1.5	0.7	0.9	0.8
019	-	0.3	0.3	0.2	8.0	0.5	0.5	**0.2**	0.2	0.9	0.5	**0.2**	0.2	0.9	0.5
021	-	1.1	16.8	0.4	6.5	0.9	0.9	1.4	1.2	0.6	0.4	0.8	0.7	**0.5**	0.4
024	-	0.7	3.8	0.6	2.6	2.0	1.9	**0.4**	0.4	0.5	0.3	1.4	1.0	0.5	0.3
025	-	0.7	173	0.5	4.7	0.4	0.4	**0.5**	0.4	0.6	0.5	**0.5**	0.4	**0.5**	0.4
035	-	0.6	28.7	0.4	9.3	0.9	0.4	**0.5**	0.5	1.1	0.6	**0.5**	0.4	3.2	0.4
036	-	1.6	30.6	0.6	2.9	2.1	3.2	1.8	1.8	**1.4**	1.4	3.0	3.0	1.6	1.5
037	-	1.6	71.8	0.5	17.0	0.5	0.5	2.8	2.7	**0.6**	0.6	3.2	3.2	1.3	1.4
040	-	**0.5**	1.1	0.3	4.5	0.6	0.6	2.8	1.1	1.0	1.0	2.8	1.1	1.8	1.7
051	-	12.4	38.3	0.7	**3.4**	3.3	1.5	15.3	15.9	13.2	11.5	17.9	18.9	16.7	16.9
052	-	5.1	7.5	1.4	14.3	14.6	14.6	5.8	6.6	5.5	5.4	**4.3**	4.5	13.5	12.1
061	-	0.9	28.8	28.1	2.7	1.6	0.8	4.3	0.4	**0.5**	0.4	2.3	1.1	0.7	0.6
ALL	-	3.2	98.4	10.0	8.4	4.3	4.3	3.7	3.8	**3.1**	2.7	4.1	4.3	4.9	4.7

each object. This requirement is more demanding than retraining a 2D segmentation network once for all the objects of interest.

Remarkably, despite its algorithmic simplicity, the ICP baseline, combined with a segmentation network, outperforms or behaves similarly to more complex object pose tracking methods. This fact suggests that the YCB-Video dataset might be not challenging enough, despite having become a popular dataset for pose estimation benchmarking, and might be unsuitable to identify the shortcomings of classical approaches as ICP when considering metrics as the ADI-AUC.

4.2.6 Results: RMSE metric

Tables 4.3 and 4.4 shows the evaluation for all the 21 objects in the YCB-Video dataset using the positional and rotational RMSE metrics.

Comparison with pose tracking algorithms

When considering both segmentation sources and the initialization with the perturbed ground truth, MaskUKF outperforms se(3)-TrackNet and PoseRBPF. On average, the positional error reduces by ≈ 95% with respect to se(3)-TrackNet, while it approximately halves with respect to PoseRBPF. Considering the angular error, the error reduces

Table 4.4 Quantitative evaluation on all the frames of the YCB-Video dataset using the rotational RMSE metric. A bold entry indicates a better result. For the tracking algorithms, the best results are decided taking into account the experiment initialized with the perturbed ground truth.

metric	Rotational RMSE (deg)														
method	DOPE	Dense Fusion	se(3)-TrackNet		PoseRBPF (200 particles)			ICP		MaskUKF		ICP		MaskUKF	
segm.	-	Pose CNN	-		-			PoseCNN				Mask R-CNN			
init.	-	-	pert.	gt	pert.	gt	gt (multi)	pert.	gt	pert.	gt	pert.	gt	pert.	gt
002	-	2.3	83.5	2.4	33.8	72.8	4.7	2.0	2.1	1.6	1.4	14.8	14.6	**1.5**	1.2
003	30.7	4.9	3.4	2.2	97.1	5.7	5.7	**2.3**	2.3	3.0	2.8	3.1	3.1	59.7	14.5
004	13.3	2.9	42.3	1.3	99.0	5.8	5.8	2.0	2.0	**1.8**	1.5	76.1	2.6	2.0	1.5
005	18.8	9.2	40.0	28.1	134	77.8	22.4	63.7	31.4	13.0	4.7	45.6	45.6	**6.3**	6.1
006	4.7	27.8	58.1	28.6	112	7.3	7.3	5.3	5.2	3.3	2.6	3.6	3.2	**3.2**	2.6
007	-	37.2	64.2	17.9	85.4	32.2	32.3	61.6	8.5	**29.5**	12.6	61.3	6.6	77.1	70.4
008	-	**7.6**	38.7	1.8	10.5	172	11.2	89.6	20.5	18.3	3.9	20.1	20.0	142	75.4
009	7.4	3.9	74.9	3.1	135	4.8	4.8	4.6	4.6	2.4	1.4	2.0	2.0	**1.8**	1.1
010	45.9	54.1	77.0	50.1	153	81.7	75.1	131	134	68.5	68.3	114	133	**5.6**	5.0
011	-	46.1	81.3	20.5	143	34.4	13.4	10.6	10.6	5.5	5.5	19.4	7.6	**5.3**	4.6
019	-	7.6	**2.1**	2.0	178	12.6	12.6	**2.1**	2.1	4.0	3.3	2.3	2.3	4.0	3.5
021	-	14.2	74.1	2.9	124	9.8	9.8	87.0	78.7	7.5	7.5	24.4	15.1	**7.0**	6.9
024	-	85.9	8.3	5.5	126	5.5	5.3	4.2	4.0	**2.7**	2.0	25.0	8.1	3.1	2.4
025	-	50.3	110	12.6	163	7.7	7.7	122	18.3	**6.0**	6.0	120	12.3	17.4	16.5
035	-	22.1	85.5	1.9	129	3.5	3.5	**3.3**	3.3	6.1	4.4	3.7	2.5	21.1	2.0
036	-	7.7	120	2.1	170	9.6	21.3	9.5	9.4	**1.6**	1.6	21.3	21.3	1.7	1.7
037	-	82.3	54.8	6.6	114	8.5	8.5	31.0	31.0	**5.1**	4.6	32.4	32.5	5.8	5.7
040	-	**4.3**	15.9	15.6	92.4	120	7.0	10.8	10.2	5.5	7.6	120	11.1	6.7	18.3
051	-	100	60.4	3.2	**18.6**	13.9	6.4	75.8	62.1	85.2	52.9	74.8	55.1	72.8	54.4
052	-	122	80.2	4.9	68.9	36.0	36.0	52.3	48.8	**22.8**	51.9	24.3	14.6	67.8	65.3
061	-	91.6	150	55.6	108	177	5.8	139	5.5	**3.5**	2.7	176	87.1	10.5	9.3
ALL	-	45.1	67.3	18.8	116	56.8	22.9	60.8	42.9	**26.0**	22.3	62.3	38.8	38.5	28.7

with respect to se(3)-TrackNet by approximately 60% and 40%, on average, when using the PoseCNN segmentation and Mask R-CNN, respectively. The reduction increases to $\approx$ 80% and $\approx$ 70% with respect to PoseRBPF.

When considering both segmentation sources and the initialization with the actual ground truth, the positional error becomes comparable on average with that obtained by PoseRBPF. On the other hand, the error produced by se(3)-TrackNet is, on average, slightly more than twice as that of MaskUKF and PoseRBPF. The analysis of the rotational error reveals that se(3)-TrackNet outperforms both PoseRBPF and MaskUKF. For example, MaskUKF with the PoseCNN segmentation obtains, on average, 26 degrees, PoseRBPF obtains 56 degrees while se(3)-TrackNet achieves 18.8 degrees.

In general, these results indicate that se(3)-TrackNet could provide better performance than PoseRBPF and MaskUKF for the orientation tracking task if the initial condition is equal to the actual ground truth. Nevertheless, when the initial condition is perturbed, MaskUKF results more robust with an increase from $\approx$ 4 to $\approx$ 10 degrees on average, reaching 26.0 and 38.5 degrees when using the PoseCNN segmentation and Mask R-CNN, respectively. For se(3)-TrackNet, instead, the error increases by more than three times, reaching 67.3 degrees, while for PoseRBPF it approximately doubles reaching 116 degrees. A similar reasoning holds for the positional error. For MaskUKF

Table 4.5 Maximum occlusion percentage among YCB-Video dataset frames for each object.

objects	%
005_tomato_soup_can	96.9
010_potted_meat_can	69.6
061_foam_brick	68.2
003_cracker_box	61.2
024_bowl	59.7
052_extra_large_clamp	59.0
008_pudding_box	51.4
035_power_drill	47.6
037_scissors	45.4
036_wood_block	44.9
002_master_chef_can	44.6
021_bleach_cleanser	43.0
011_banana	42.5
004_sugar_box	40.95
051_large_clamp	39.3
040_large_marker	22.3
025_mug	15.5
019_pitcher_base	12.8
007_tuna_fish_can	8.0
006_mustard_bottle	3.6
009_gelatin_box	0.8

it increases by approximately 0.3 cm on average, reaching 3.1 and 4.9 cm using the alternative segmentation sources. For se(3)-TrackNet it increases by 9 times reaching 98.4 cm and for PoseRBPF it doubles reaching 8.4 cm.

We notice that the considerable average positional error of se(3)-TrackNet is due to the fact that the perturbation applied to the ground truth, at initialization time, leads to early track loss for almost all objects.

We also remark that the multistage initialization procedure of PoseRBPF allows reducing the rotational error even further from 56.8 degrees to 22.9 degrees on average, when using the exact ground truth for initialization. I.e. better than MaskUKF but still not good as se(3)-TrackNet.

Comparison with pose estimation algorithms

When equipped with the PoseCNN segmentation network, MaskUKF outclasses the DenseFusion framework that uses the same segmentation source. While the increase in performance is minimal for the Cartesian error, the difference is considerable for the angular error that, in average, is almost halved. For specific objects, e.g. 037, 021, 024 and 061, the reduction is indeed substantial, e.g. for the object 061 from 91.6 degrees to 3.5. Remarkably, these objects are among those involved in moderate to severe occlusions as can be seen in Table 4.5 where we reported the maximum percentage of occlusion for each object among all YCB-Video frames.

When comparing with DOPE on the available objects, it can be seen that MaskUKF provides better estimates of the object translation independently of the considered segmentation. The errors are in the order of few centimeters while the errors produced by the DOPE network can reach values as high as 27 centimeters (see e.g. the object 010). Considering the rotational error, MaskUKF provides better performance except for the object 010 when using the PoseCNN segmentation and the object 003 when using the Mask R-CNN segmentation. Nevertheless, the performance for the same objects using the alternative segmentation reduces from ≈ 30 degrees to ≈ 3 and from 45.9 degrees to ≈ 5 degrees respectively. This fact shows again that the performance of MaskUKF depends on the adopted segmentation. It also indicates that the performance can be improved by picking a better segmentation network for the task at hand.

Comparison with the baseline

When equipped with the PoseCNN segmentation network, MaskUKF outperforms the ICP method. While the increase in performance is minimal for the Cartesian error, the difference is considerable for the angular error that, on average, is almost halved when using the ground truth for the initialization and reduced by approximately one third when using the perturbed ground truth for initialization.

Using the segmentation from Mask R-CNN, MaskUKF still outperforms ICP in terms of the angular error while the Cartesian error is comparable. As already noticed, also the RMSE metrics reveals, on average, a drop in performance when using this segmentation with respect to PoseCNN due to missing detections or completely wrong segmentation.

If both segmentations are considered, it can be seen that, for specific objects, the reduction in angular error is indeed substantial even when considering the perturbed ground-truth as the initialization. As an example, when using Mask R-CNN, the angular error for object 005 is ≈ 6 degrees for MaskUKF while, for ICP, is ≈ 45 degrees. With the same segmentation, the error for object 010 is ≈ 5 degrees while, for ICP, is 114 degrees. Moving to the PoseCNN segmentation, for the object 061 MaskUKF achieves ≈ 3 degrees while, for ICP, it reaches 139 degrees. Similar examples are those of objects 021, 024 and 037, that are among those involved in moderate-to-severe occlusions (see Table 4.5).

Table 4.6 Frame rate comparison (fps).

method	DOPE	DenseFusion	se(3)-TrackNet	PoseRBPF (200 particles)	ICP	MaskUKF
fps	4.31	30.0	90.9	7.6	91.7	52.6

4.2.7 Results: frame rate comparison

In Table 4.6 we report a comparison of the frame rates achieved by the compared algorithms.

For MaskUKF, the frame rate is evaluated as the inverse of the mean time, averaged on the total number of frames of the testing sequences, required to perform outlier rejection and the Kalman prediction and correction steps. The time required to segment the object is not considered in the computation since MaskUKF can run asynchronously with respect to the frame rate of the segmentation algorithm, if needed (see 4.1.6).

For ICP, the evaluation was done on the mean time required to perform the registration step between the source and the target point cloud.

For PoseRBPF, se(3)-TrackNet, DenseFusion and DOPE the frame rates are those reported in the associated publications.

We remark that the frame rate reported by the authors of DenseFusion, i.e. 16.7 fps, includes the time required to segment the object. In order to have a fair comparison, the segmentation time has been omitted resulting in a frame rate of 30 fps for this algorithm. MaskUKF, ICP and se(3)-TrackNet exceed the real-time performance requirement (30 fps). This fact might be useful in case a faster than real-time image source is available.

4.2.8 Qualitative results

Comparison between object pose estimation, pose tracking and ICP

In this section, I will compare qualitatively the proposed tracking method, MaskUKF, with an object pose estimation method, i.e. DenseFusion, and a registration algorithm, i.e. ICP, in order to emphasize some of the advantages of using a tracking method.

In Fig. 4.2, the estimated pose is represented as a coloured point cloud transformed in the estimated pose and projected onto the input RGB frame. Both ICP and DenseFusion fail to estimate the correct pose of the cans 010 and 005 in the leftmost

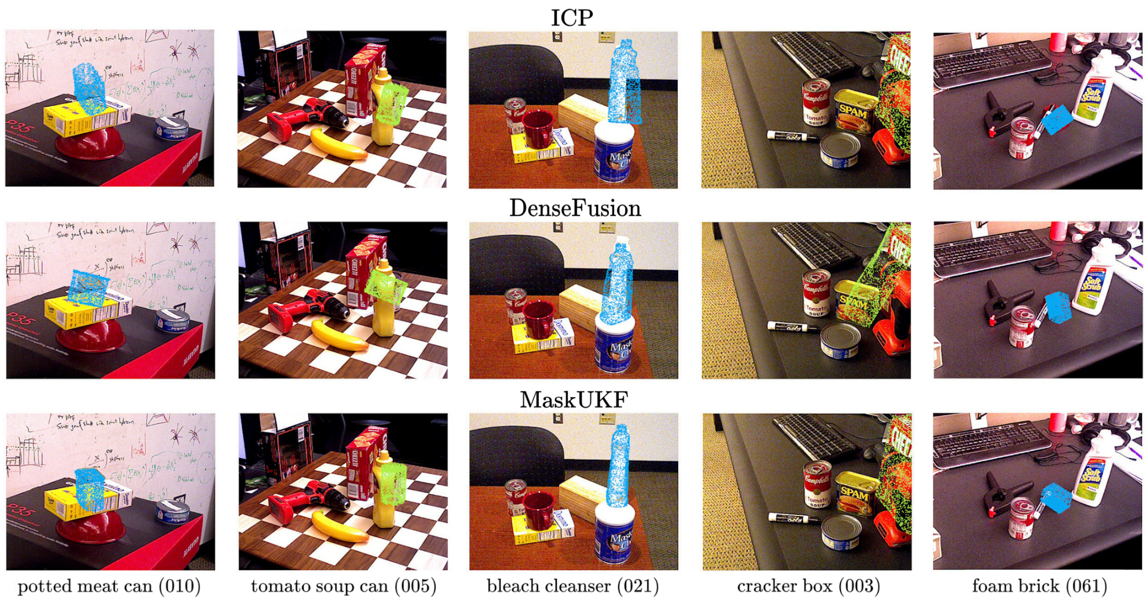


Figure 4.2 Qualitative results on the YCB-Video dataset. All the results reported here are obtained using the segmentation masks from PoseCNN. The estimated pose is represented as a coloured point cloud transformed in the estimated pose and projected onto the 2D RGB frame.

columns due to severe occlusions. In the central column, the orientation of the bleach cleanser (021) from DenseFusion and ICP is wrong. Another interesting case is that of the cracker box (003), on the right, that is only partially visible in the RGB frame. While DenseFusion struggles to estimate the correct orientation, MaskUKF and ICP are able to provide it properly. These results suggest that the algorithms based on 3D geometric correspondences between the measured point cloud and the 3D model, such as MaskUKF and ICP, better handle the presence of occlusions than the DenseFusion pose regression network. Nonetheless, if the available correspondences are few in number, as happens in presence of severe occlusions, the orientation estimated by ICP tends to be wrong.

Fig. 4.3 shows trajectory samples of the estimated pose for the bleach cleanser (021), obtained by different algorithms. In this scenario, the cleanser is visible from its shortest and texture-less side (see Fig. 4.2) hence producing partial and ambiguous measurements. The trajectories of MaskUKF are smoother than those of DenseFusion. In particular, the orientation estimate by DenseFusion is affected by considerable discontinuities. We can explain the smoothness of MaskUKF in terms of the motion model that helps regularize the estimate and reduce the effect of the noise. Considering

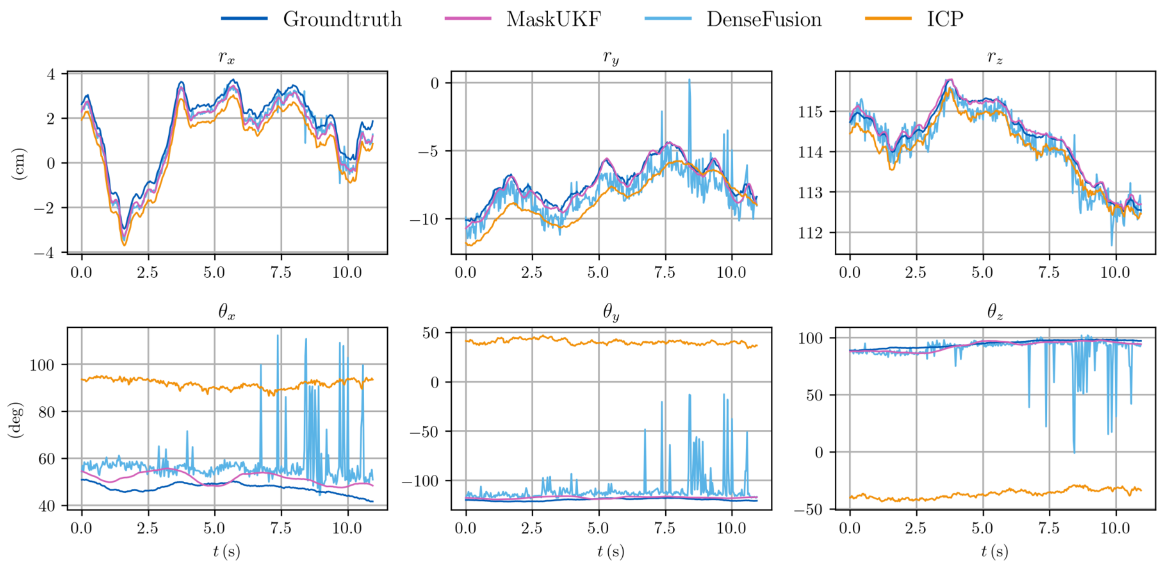


Figure 4.3 Comparison of the trajectories of several algorithms for the object bleach cleanser (021) within a subset of the sequence 0055 from the YCB-Video dataset. MaskUKF is more precise than ICP and smoother than DenseFusion which exhibits spikes and irregularities. In this figure, the orientation of the object is represented in a compact way using the Euler vector $\theta n = (\theta_x, \theta_y, \theta_z)$ obtained as the product of the axis of rotation n and the angle of rotation θ.

ICP, we observe that the level of noise in the output is comparable to that of MaskUKF. However, the error with respect to the ground truth is visibly higher, especially for the rotational part of the state. Indeed, as shown both in the plot and in Fig. 4.2, the estimate from ICP has the tendency to pivot around the visible part of the object resulting in a steady but totally wrong estimate of the orientation.

Qualitative evaluation of object velocity tracking

Differently from se(3)-TrackNet and PoseRBPF, MaskUKF also provides an estimate of the mixed velocity $\mathcal{V}^m$ of the object consisting in the linear velocity $v^m = (v^m_x, v^m_y, v^m_z)$ and the angular velocity $\omega = (\omega_x, \omega_y, \omega_z)$.

In Fig. 4.4, a qualitative example of the tracked velocities is shown. We remark that the YCB-Video dataset does not provide the ground truth velocities. Hence, they have been extracted from the ground truth poses by finite differentiation.

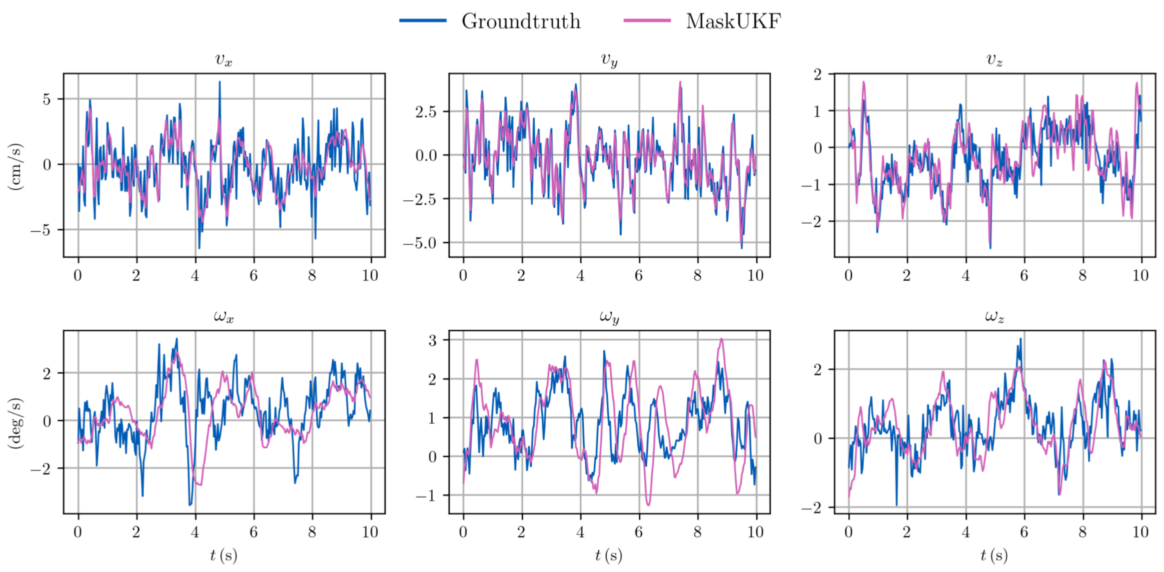


Figure 4.4 Comparison between the estimated linear velocity v^m and angular velocity ω and the ground truth velocities within a video sequence from the YCB-Video dataset. The ground truth velocities are obtained from finite differences.

4.2.9 Discussion on the robustness to perturbations of the initial conditions

One of the most interesting aspects highlighted by the experimental evaluation is the increased robustness of MaskUKF to perturbations of the initial condition of the object pose. It can be largely attributed to the fact that MaskUKF is a hybrid framework where the 6D tracking part depends on the Kalman filtering stage. Hence, the convergence properties can be modified by tuning the process and measurement covariance matrices as well as the initial condition of the state and the associated covariance. In MaskUKF, the effect of the training data is limited to the segmentation stage. This stage decides how the partial point cloud of the object is extracted from the entire scene but does not affect directly the mechanism which updates the state given the input measurements. I.e. the training data does not affect the algorithmic prior given by the Kalman filter itself. On the other hand, end-to-end approaches, such as se(3)-TrackNet, learn how to track the object pose directly from the data. However, they do not only learn where to extract interesting information given the input images but also how to update the state of the object. As a consequence, in presence of overfitting, slight changes in the initial conditions might produce pose

Table 4.7 Performance of MaskUKF, with PoseCNN segmentation, with different implementations of the measurement function.

metric	ADI-AUC	Positional RMSE (cm)	Rotational RMSE (deg)
w/ k-d tree	94.2	3.1	26.0
w/ CGAL	94.6	2.8	28.3

residuals not seen at training time and that are not well handled resulting in track loss as the presented results show.

On the other hand, PoseRBPF is a hybrid approach, similarly to MaskUKF. However, its initialization mechanism proved to work well only if several initializations are allowed within the first frames of the trajectory. Unfortunately, this mechanism requires the knowledge of the ground truth, or an external source of pose measurements, to decide whether the initialization is good enough by comparing the rotational error with a threshold. As a consequence, this strategy might be unfeasible in some contexts. Furthermore, there are no guarantees on the maximum number of initializations that are required to satisfy the condition based on the threshold.

4.2.10 Discussion on the approximation of the measurement function

Table 4.7 compares the overall tracking performance obtained when using different implementations of the measurement function in Eq. (4.9) - (4.11), as discussed in Sec. 4.1.4. Specifically, the first row refers to the implementation where the projections in (4.9) are obtained via a nearest neighbour search on a k-d tree containing approximately 2000 points sampled on the 3D mesh of the object. The second row, instead, refers to the alternative implementation where the projections are obtained via closest point queries executed on the full 3D triangle mesh using the CGAL library.

We remark that, on average, the 3D meshes from the YCB-Video dataset contains approximately 250k+ vertices and 500k+ faces, i.e. they can be considered as fairly accurate reconstructions of the shape of the actual objects. As a consequence, the alternative implementation does not allow reaching the real-time performance resulting in a up to ten time slower execution.

Despite the fact that the 3D mesh model is much more accurate than a sampled point cloud, the overall tracking performance, as can be seen from the table, is comparable. One possible explanation of the results is that most of the objects adopted in the YCB-Video dataset have shapes of boxes and cylinders or comparable to them with

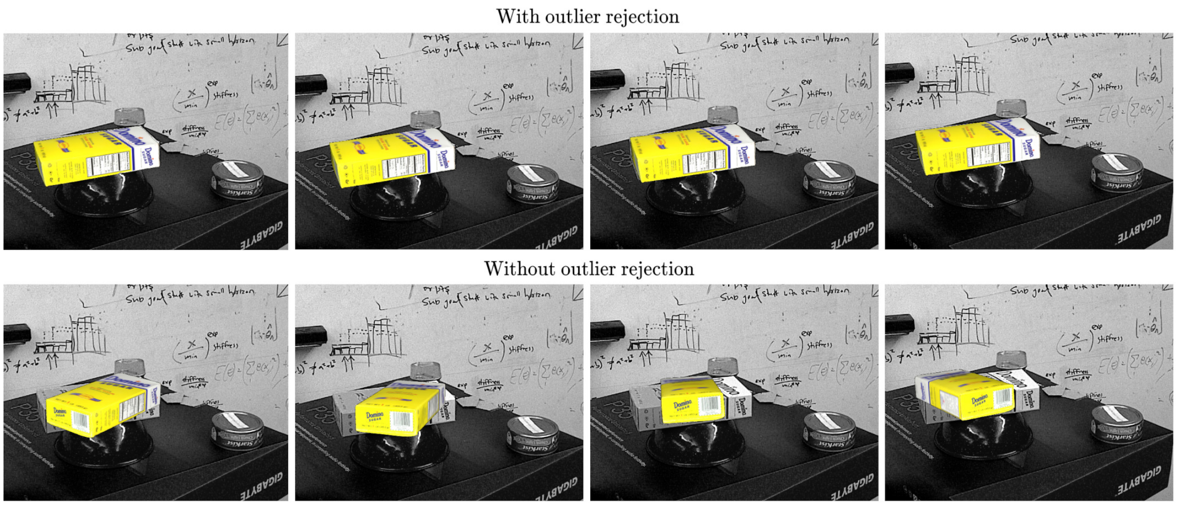


Figure 4.5 Sample estimates from MaskUKF executed on a YCB-Video sequence with and without the outlier rejection mechanism. The estimated pose is superimposed over the grayed-out input frames.

Table 4.8 Performance of MaskUKF, with PoseCNN segmentation, with and without outlier rejection.

metric	ADI-AUC	Positional RMSE (cm)	Rotational RMSE (deg)
w/ outlier rejection	94.2	3.1	26.0
w/o outlier rejection	83.3	5.05	76.9

very few exceptions, e.g., the banana (010), the scissors (025), the power drill (035) and the clamps (051 and 052). For the former shapes, it is reasonable to expect that even a simple point cloud sampled on the mesh of the object could be sufficient for the task at hand and could produce similar results.

4.2.11 Discussion on the role of the outlier rejection mechanism

Table 4.8 shows the effectiveness of the outlier rejection procedure presented in section 4.1.5. MaskUKF performs better when the outliers are taken into account, especially in terms of the reduced angular error.

To better visualize the effects of outliers in the depth measurements, in Fig. 4.5 we show sample estimates of the pose of the object sugar box (004) when the outlier rejection mechanism is enabled or disabled. As it can be seen, the outliers can substantially impact the estimate of the object orientation if not properly handled.

4.2.12 Discussion on the requirements of object manipulation tasks

We conclude this section by considering several examples of robotic manipulation tasks along with their requirements and discussing whether the latter are compatible with the pose accuracy achieved by the methods compared in the previous sections.

Object grasping

It is a known fact (Pattacini et al., 2010) that a successful grasp depends substantially on the ability of the robot to place the end-effector very near to the intended grasping position and less on the orientation (which remains nonetheless important). Indeed, if the end-effector reaches the right orientation but the positional error is in the order of even few centimeters, the grasp will fail more than if the position is correct but the orientation alignment is not fulfilled, with errors up to ten degrees.

In this respect, by looking at the individual rows of Table 4.3, and excluding the few error peaks, we can conclude that for the majority of the objects the positional error is in the order of one or two centimeters for MaskUKF, ICP and DenseFusion (when using the PoseCNN segmentation). For MaskUKF and ICP, this fact holds in the more challenging case in which the methods are initialized with the perturbed ground truth. The same cannot be said for PoseRBPF and especially for se(3)-TrackNet, unless a very precise initialization, i.e. similar to the ground truth, is available.

Considering the orientation error, as per the Table 4.4, the best performing method, i.e. MaskUKF with PoseCNN segmentation, provides angular errors below ten degrees for 15 objects out of 21, with a perturbed initialization. Using the same segmentation, ICP and DenseFusion can reach the same accuracy for 9 objects out of 21. This number reduces to 3 for se(3)-TrackNet and to 0 for PoseRBPF. Also in this case, if a very precise initialization is available, the latter methods would provide a reasonable orientation error for 13 object ouf of 21.

These considerations suggest that the compared methods are certainly suitable for grasping purposes but not for all objects and not under all circumstances, i.e. a good initialization is required for some of them. We would like to remark that these conclusions are partially confirmed by the publication associated to DenseFusion (Wang et al., 2019) where the method is tested in grasping experiments showing a 73% success rate, although the tests were limited to 5 objects.

Future improvements in the accuracy of object pose tracking and estimation algorithms might certainly benefit the reliability of robot grasping pipelines.

Pouring on an object

In order to successfully execute a pouring action, we expect that the positional error is at least compatible with the radius of the container of interest.

Among the objects that appear in the YCB-Video dataset and can be used as containers where a robot can pour on, we consider the bowl (024) and the mug (025).

The bowl has a radius of 7.5 cm which is compatible with the positional error of approximately half a centimeter achieved by MaskUKF and ICP (both initialized with a perturbed grund truth and using the PoseCNN segmentation). A similar accuracy is achieved by DenseFusion with the same segmentation. With PoseRBPF and se(3)-TrackNet (both initialized with the perturbed ground truth) the error reaches 2.6 cm and 3.8 cm, respectively, i.e. still compatible.

The mug is more challenging as the radius reduces to 4 cm. In this case, MaskUKF, ICP and DenseFusion still provide a reasonable performance. The errors of PoseRBPF and se(3)-TrackNet are not compatible instead.

We remark that the angular error also plays an important role, since a small error helps avoiding collisions with the object during the pouring action. In this respect, MaskUKF and ICP provide errors less than 5 degrees for the bowl and, for MaskUKF only, less than 6 degrees for the mug. As before, we only consider the configuration with the PoseCNN segmentation and with the perturbed initialization. The errors of DenseFusion, PoseRBPF and se(3)-TrackNet are instead considerably higher for both objects, excluding the case of the bowl where se(3)-TrackNet achieves an angular error less than 9 degrees.

Further considerations on the pouring task are presented in the Sec. 4.3 of this Chapter. In that section, an object pose tracking method, i.e. MaskUKF, is compared with object pose estimation and registration methods, i.e. DenseFusion and ICP, on a pouring task performed in simulation with a humanoid robot. The results will highlight the fact that using a tracking framework provides several benefits for the pouring task.

In-hand object manipulation

In some cases, the requirements for grasping an object that we indicated above might not be sufficient to carry out the task at hand. An example is that of in-hand object manipulation where the robot acts to change the relative pose between the end-effector and the object.

In (Costanzo et al., 2021), the authors control the relative orientation between the object and the end-effector via object or gripper pivoting actions involving rotational sliding. They report that the grasping position should be reached with an error of no more than 2 mm in order to have a successful pivoting. As the authors report, recent methods for 6D object pose estimation, such as DenseFusion, do not meet these requirements.

The results presented in this Chapter show that none of the considered methods would be suitable for this scenario. In this respect, we recognize the necessity to consider these kind of requirements in the development of future methods for object pose tracking and estimation.

4.3 Pose estimation and pose tracking for robot motion control

Thus far, the MaskUKF algorithm has been compared with other object pose tracking, estimation and registration methods on a given dataset. On the one hand, the results indicate possible advantages and disadvantages of different kind of tracking architectures, i.e. end-to-end as opposed to hybrid. On the other hand, they allowed identifying possible general advantages of pose tracking methods over pose estimation or registration ones.

In this section, I will extend the latter comparison to a different task, i.e. the control of the motion of a robot for the execution of a task depending on the pose of the object. The results will show that using a pose tracking method, such as MaskUKF, allows executing the task with safer movements and better precision than when using a pose estimation or registration algorithm. The algorithms considered for the comparison are the pose estimation network DenseFusion and the ICP registration algorithm.

In the remainder of this section, the experimental setup and the adopted metrics will be first described followed by the presentation of the experimental results.

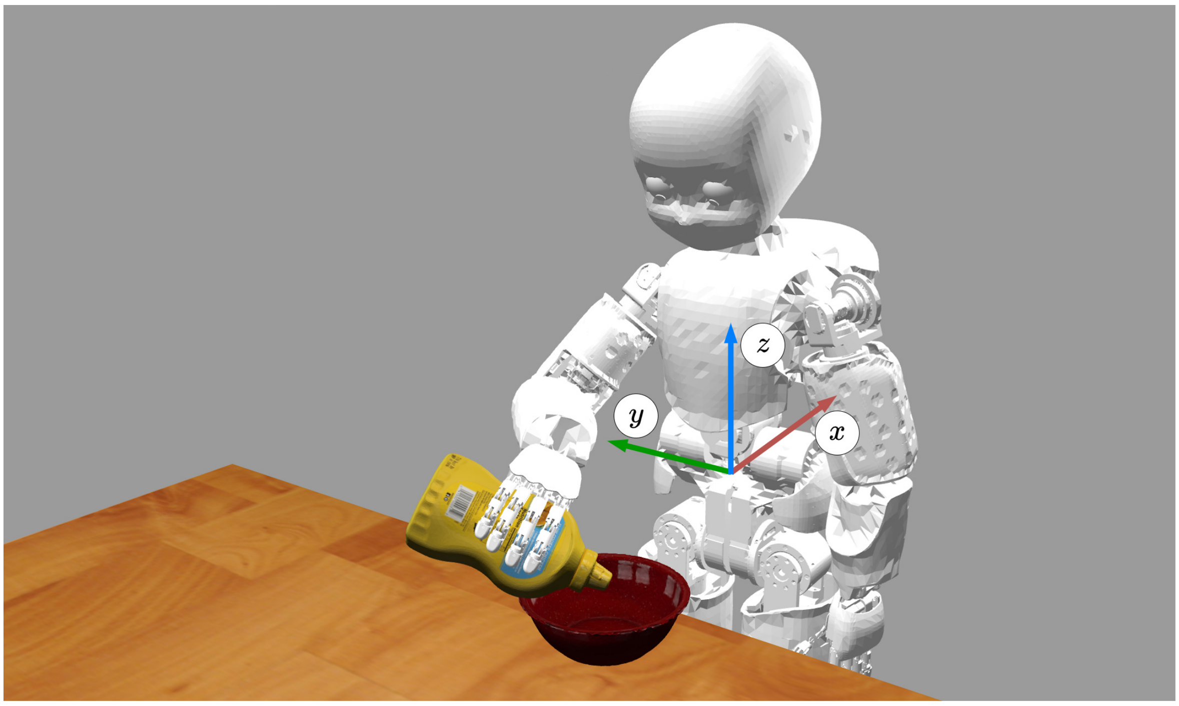


Figure 4.6 The iCub robot in the Gazebo environment. The root frame attached to the robot is also depicted.

4.3.1 Description of the experimental setup

The task at hand is concerned with the goal of following a moving container with the end-effector of a humanoid robot while the robot is holding a bottle and pouring its content inside the container. The experiment is executed in a simulated environment, namely the Gazebo (Koenig and Howard, 2004) environment, using the iCub humanoid robotic platform (Metta et al., 2010),

In order to command the robot, the output of a given object perception algorithm is fed as reference signal to a closed loop control system, later described. The task is depicted in Figs. 4.6 and 4.7.

Assumptions

We assume that the 6D pose and the velocity of the container are estimated and or tracked using RGB-D information. On the other hand, the position of the bottle is known given the robot kinematics, i.e. the focus is on tracking the state of the container only.

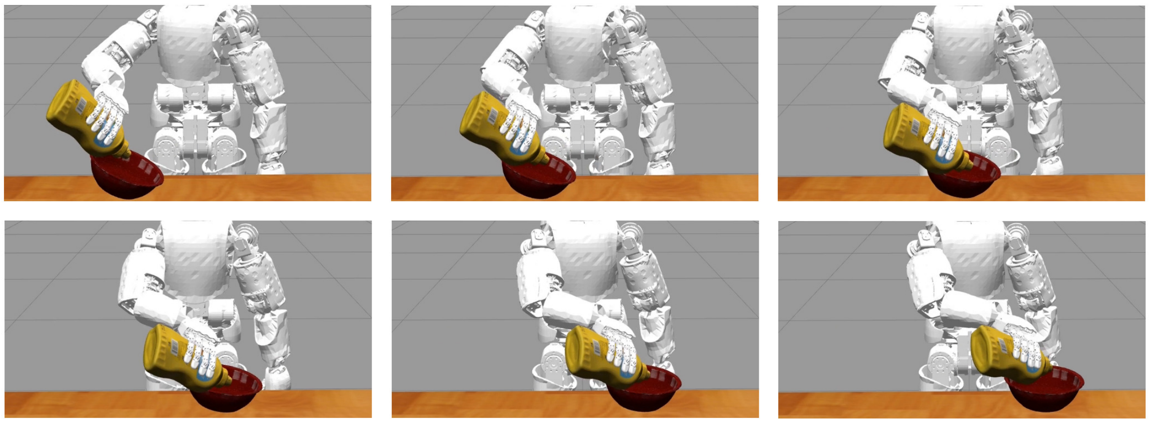


Figure 4.7 The iCub robot in the Gazebo environment while it follows a moving container, i.e. the red bowl, during a pouring task using the estimate of the pose and velocity of the container as feedback signal.

Furthermore, there is no interest in the grasping task nor in the pouring action per se but more on the possibility of exploiting the estimated signal in a closed loop fashion in order to follow the container as close as possible while avoiding contacts that might compromise the pouring action and leakages of the content of the bottle.

I adopted the YCB Object and Model Set (Calli et al., 2015) to provide the reference objects for the experiment; in particular, I chose the mustard bottle (006) as the bottle and the bowl (024) as the container (see Fig. 4.6).

The experiment is carried out in the Gazebo environment using a simulated model of the iCub robotic platform. Even though iCub features 53 DoF, only a subset of these are used in the experiment, i.e. 3 DoF in the torso, 7 DoF in the arm and 6 DoF in the head, where the stereo vision system is mounted.

It is assumed that the vision system of the robot provides the object segmentation and RGB-D images of the scene to be used to estimate and/or track the pose of the container. Additionally, the iCub gaze control system (Roncone et al., 2016) is used to track the container in the image plane by using the estimated Cartesian position of the object as a reference gazing point.

At the beginning of the experiment, the end-effector is reset to a rest configuration near the container. Then, a sinusoidal trajectory is assigned to the moving container along the y direction of the robot root frame (shown in Fig. 4.6). Similarly, a sinusoidal trajectory is assigned to the orientation of the container along the x axis of the robot root frame. Each experiment lasts 1 minute in order to test the reliability of the overall system. An excerpt of the trajectory of the moving container can be seen in Fig. 4.7.

The MaskUKF and ICP algorithms, which require an initial condition, are reset to the ground truth state provided by the simulation environment.

Description of the control system

In order to carry out the above described task, the torso (two DoF out of three) and one of the arms are considered as a 9 DoF serial chain whose dynamic behavior is described by the standard equation

$$M(q)\ddot{q} + h(q, \dot{q}) = \tau, \tag{4.14}$$

where q, $\dot{q}$ and $\ddot{q} \in \mathbb{R}^9$ are the joints angles, velocities and accelerations respectively, $M(q) \in \mathbb{R}^{9\times 9}$ is the Mass matrix of the chain, $h(q, \dot{q}) \in \mathbb{R}^9$ represents the effect of centrifugal, Coriolis and gravity terms and $\tau \in \mathbb{R}^9$ are the torques applied to the chain joints axes.

In order to command the end-effector of the robot and follow the moving container over time, a two-layer control architecture is adopted. The first layer consists in an Inverse Dynamics controller

$$\begin{aligned} \tau_{cmd} &= J_{ee}(q)^T \Lambda(q) \left(\dot{\mathcal{V}}^m_{des} - \dot{J}_{ee}(q)\dot{q}\right) + h(q, \dot{q}), \\ \Lambda(q) &= \left(J_{ee}(q) M(q)^{-1} J_{ee}(q)^T\right)^{-1}. \end{aligned} \tag{4.15}$$

Here, $J_{ee} \in \mathbb{R}^{6\times 9}$ is the Jacobian of the end-effector expressed in the robot root frame which links the joints velocities $\dot{q}$ with the end-effector linear velocity $v^m_{ee} \in \mathbb{R}^3$ and angular velocity $\omega_{ee} \in \mathbb{R}^3$

$$\begin{bmatrix} v^m_{ee} \\ \omega_{ee} \end{bmatrix} = J_{ee}(q)\dot{q}, \tag{4.16}$$

both expressed in the robot root frame. We recall that the subscript m indicates that the velocity is expressed using the mixed convention introduced in Sec. 3.1.3. The matrix $\Lambda(q) \in \mathbb{R}^{9\times 9}$ in Equation (4.15) is also called the Task Space Mass matrix. The term $\dot{\mathcal{V}}^m_{des} \in \mathbb{R}^6$ is a vector containing the desired linear acceleration of the end-effector

$\dot{v}^m_{des} \in \mathbb{R}^3$ and the desired angular acceleration $\dot{\omega}_{des} \in \mathbb{R}^3$

$$\dot{\mathcal{V}}_{des} = \begin{bmatrix} \dot{v}^m_{des} \\ \dot{\omega}_{des} \end{bmatrix}, \tag{4.17}$$

both expressed in the robot root frame.

If controlled using the torques in Equation (4.15), the system in Equation (4.14) reduces to the system of equations

$$\dot{\mathcal{V}}^m_{ee} = \begin{bmatrix} \dot{v}^m_{ee} \\ \dot{\omega}_{ee} \end{bmatrix} = \dot{\mathcal{V}}^m_{des}. \tag{4.18}$$

In essence, the first layer allows reducing the dynamics of the serial chain to a linear system having $\dot{\mathcal{V}}^m_{des}$ as input.

The second control layer consists in a Proportional Derivative (PD) controller

$$\dot{\mathcal{V}}^m_{des} = k_p e_p + k_d e_v, \tag{4.19}$$

where $e_p \in \mathbb{R}^6$ is the end-effector configuration error

$$e_p = \begin{bmatrix} p_{ee} - p_{des} \\ \log(R_{des} R^T_{ee})^\vee \end{bmatrix}, \tag{4.20}$$

and $e_v \in \mathbb{R}^6$ is the end-effector velocity error

$$e_v = \begin{bmatrix} v^m_{ee} - v^m_{des} \\ \omega_{ee} - \omega_{des} \end{bmatrix}. \tag{4.21}$$

Here, $p_{ee} \in \mathbb{R}^3$ is the Cartesian position of the end-effector, $R_{ee} \in \mathrm{SO}(3)$ is the orientation of the end-effector, $v^m_{ee} \in \mathbb{R}^3$ is the linear velocity of the end-effector and $\omega_{ee} \in \mathbb{R}^3$ is the angular velocity that are known via the robot forward kinematics and forward differential kinematics maps. Moreover, $p_{des} \in \mathbb{R}^3$ is the desired position of the end-effector, $R_{des} \in \mathrm{SO}(3)$ is the desired orientation of the end-effector, $v_{des} \in \mathbb{R}^3$ is the desired linear velocity of the end-effector and $\omega_{des} \in \mathbb{R}^3$ is the desired angular velocity. We recall that the expression $\log(R_{des} R^T_{ee})$ belongs the Lie algebra $\mathfrak{so}(3)$ while $\log(R_{des} R^T_{ee})^\vee \in \mathbb{R}^3$. The log() map and the *vee* operator $^\vee$ are defined in Eqs. (3.18) and (3.13), respectively.

In the proposed experiment, it is assumed that position of the tip of the bottle with respect to the robot end-effector is constant and known. Given this assumption, the end-effector frame *ee* in Equations (4.20) - (4.21) is considered to be attached to the tip of the bottle.

The design of the control system is finalized by setting the desired quantities as follows:

$$\begin{aligned} p_{des} &= p_t, \\ R_{des} &= R_t R_0, \\ v_{des} &= v_t^m, \\ \omega_{des} &= \omega_t, \end{aligned} \tag{4.22}$$

where p_t, R_t are the *estimated* position and orientation of the container and v_t^m, ω_t are the *estimated* Cartesian and angular velocities of the container. The rotation R_0 represents a default pouring configuration that is chosen such that the tip of the bottle points downwards. Consequently, the term $R_t R_0$ represents a perturbation of the default pouring configuration depending on the estimated orientation of the container.

Estimation of the object velocity

The specification of the desired quantities in Eq. (4.22) requires the knowledge of the Cartesian and angular velocities of the container, v_t^m and ω_t. We remark that these velocities are naturally provided by MaskUKF as a part of the state. Conversely, DenseFusion and ICP are pose estimation and registration algorithms, respectively, hence they do not provide an estimate of the object velocity. For this reason, we evaluated the velocity online by using finite differentiation.

We observe that the finite-differences approximation is typically noisy and might result in jerky motions of the robot end-effector, especially when the amount of feedback to the control system is increased. In turn, these motions might introduce noise in the stream of RGB-D images from the vision system of the robot, resulting in noisy pose estimates. For these reasons, we considered an additional scenario in which both the poses and the velocities, obtained by finite differentiation, of DenseFusion and ICP are filtered using a low-pass filter. Considering this additional scenario will allow comparing the closed-loop performance obtained with a Kalman filtering-based approach, such as MaskUKF, with that obtained with a pose estimation/registration method combined with a simpler low-pass filter.

We implemented the filtering stage using a standard single-pole filter whose time-domain description is given by the discrete recurrence

$$y_t = y_{t-1} + \alpha(u_t - y_{t-1}), \tag{4.23}$$

with $u_t \in \mathbb{R}^n$ and $y_t \in \mathbb{R}^n$ the n-dimensional input and the output of the filter, respectively. The parameter $\alpha \in [0, 1]$ is associated with the cutoff frequency f_c of the filter as follows

$$f_c = \frac{\alpha}{(1-\alpha)2\pi\Delta_t} \quad \text{(Hz)}, \tag{4.24}$$

with Δ_t the sampling time. When $\alpha = 0$ the filtering action is maximum, while if $\alpha = 1$ no filtering takes place, i.e. $y_t = u_t$. The filter, as in Eq. (4.23), has been applied on the signals p_t, v_t^m and ω_t.

As regards the rotational part of the state $R_t \in \mathrm{SO}(3)$, the equation has been modified to account for the group structure of the rotation matrices as follows:

$$\begin{aligned} \delta_R &= \log(R_t^u(R_{t-1}^y)^{-1})^\vee \in \mathbb{R}^3, \\ R_t^y &= \exp((A\delta_R)^\wedge)R_{t-1}^y \in \mathrm{SO}(3). \end{aligned} \tag{4.25}$$

We recall that the exp() and log() maps are defined in Eqs. (3.17) and (3.18), respectively. The *hat* and *vee* operators, $^\wedge$ and $^\vee$, are defined in Eqs. (3.12) and (3.13), respectively.

In Eq. (4.25), R_t^y and R_t^u are the input and output rotation matrices while δ_R is a vector representation of the error between the previous output rotation R_{t-1}^y and the input rotation R_t^u in the Lie algebra of SO(3). The matrix $A \in \mathbb{R}^{3\times3}$ is a diagonal matrix

$$A = \mathrm{diag}(\alpha_x, \alpha_y, \alpha_z), \tag{4.26}$$

where the parameters $\alpha_i \in [0, 1]$ weight the error δ_R along each of the robot root frame axes and allow deciding the amount of filtering, similarly to Eq. (4.23).

Choice of the algorithm parameters

The parameters of MaskUKF that we used in the experiments are reported in Table 4.9. They are the same that have been used for the experiments on the YCB-Video dataset, with the exclusion of the power spectral densities Q_p and Q_r. Specifically, we increased the value of the entries of these matrices by one order of magnitude in

Table 4.9 Parameter set for MaskUKF for the robot motion control experiment.

Parameter	Values
$P_{0\,(1:3,1:3)}$	$\mathrm{diag}(10^{-2}, 10^{-2}, 10^{-2})\,\mathrm{m}^2$
$P_{0\,(4:6,4:6)}$	$\mathrm{diag}(10^{-2}, 10^{-2}, 10^{-2})\,(\mathrm{m/s})^2$
$P_{0\,(7:9,7:9)}$	$\mathrm{diag}(10^{-2}, 10^{-2}, 10^{-2})\,\mathrm{rad}^2$
$P_{0\,(10:12,10:12)}$	$\mathrm{diag}(10^{-2}, 10^{-2}, 10^{-2})\,(\mathrm{rad/s})^2$
Q_p	$\mathrm{diag}(1.0, 1.0, 1.0)\,(\mathrm{m/s})^2/\mathrm{s}$
Q_r	$\mathrm{diag}(1.0, 1.0, 1.0)\,(\mathrm{rad/s})^2/\mathrm{s}$
R_t	$\mathrm{diag}(10^{-3}, \cdots, 10^{-3})\,\mathrm{m}^2$
δ^{outlier}	$0.01\,\mathrm{m}$
ϵ^{valid}	0.2

order to obtain a more responsive output of the filter. This choice is motivated by the fact that the considered scenario is more dynamic than the scenario presented in the YCB-Video sequences.

4.3.2 Description of the adopted metrics

In order to compare different algorithms on the proposed task, two errors are considered. The first is the Cartesian error along the y direction of the robot root frame, where the actual motion of the container happens, between the end effector position and the ground truth position of the object, i.e.

$$e_y = |p_{ee,y} - p^{gt}_{t,y}| \quad (\mathrm{cm}). \tag{4.27}$$

If both the control system and the object perception system work as expected, this error should be as small as possible. Indeed, if the control system works as expected p_{ee} should track $p_{des} = p_t$ almost closely and if the object perception algorithm works as expected, p_t should match p^{gt}_t almost closely. Hence, the proposed metric accounts for the overall system composed by the object perception algorithm and the control loop.

Similarly, the orientation tracking task is evaluated with the angular error

$$e_R = \|\log((R^{gt}_t R_0) R^T_{ee})^{\vee}\| \; \frac{180}{\pi} \quad (\mathrm{deg}), \tag{4.28}$$

between the orientation R_{ee} of the end-effector and the desired orientation $R^{gt}_t R_0$ evaluated using the real orientation of the object R^{gt}_t.

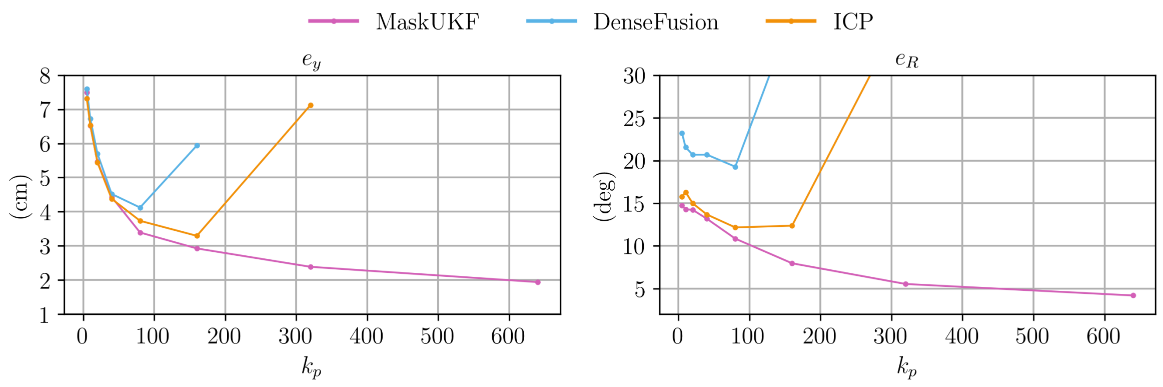


Figure 4.8 RMSE Cartesian and angular error for the object following experiment for varying proportional gains k_p. Using MaskUKF allows increasing the gain up to $k_p = 640$ reaching better tracking performance.

4.3.3 Results

In this section, we compare the performance obtained with MaskUKF with that obtained with the pose estimation and registration algorithms DenseFusion and ICP.

We first discuss the results obtained when using the plain output of DenseFusion and ICP, i.e. without applying the low-pass filtering stage.

Results without low-pass filtering

In Fig. 4.8, MaskUKF, DenseFusion and ICP are compared in terms of the Cartesian error e_y and the angular error e_R. Specifically, the Root Mean Square Error (RMSE) along 1 minute of experiment is shown for several choices of the proportional gain k_p, namely $k_p \in \{5, 10, 20, 40, 80, 160, 320, 640\}$. For each choice of the gain, the derivative gain k_d is set equal to $2\sqrt{k_p}$ as this choice assures the fastest possible closed loop dynamics for the double integrator system in Equation (4.18).

As shown in Fig. 4.8, both the errors decrease for all the algorithms when the gain increases up to $k_p \approx 80$. The error trend changes for DenseFusion in the experiment with $k_p = 160$ and for ICP in the experiment with $k_p = 320$. We could not perform the experiments for higher gains for both algorithms as they would result in jerky and erratic motions of the robot end-effector. Conversely, the tracking method MaskUKF allows increasing the gain up to $k_p = 640$ and reaching a lower tracking error.

This behavior can be explained in terms of the reduced amount of noise in the pose estimates produced by MaskUKF as shown qualitatively in Sec. 4.2.8. Furthermore,

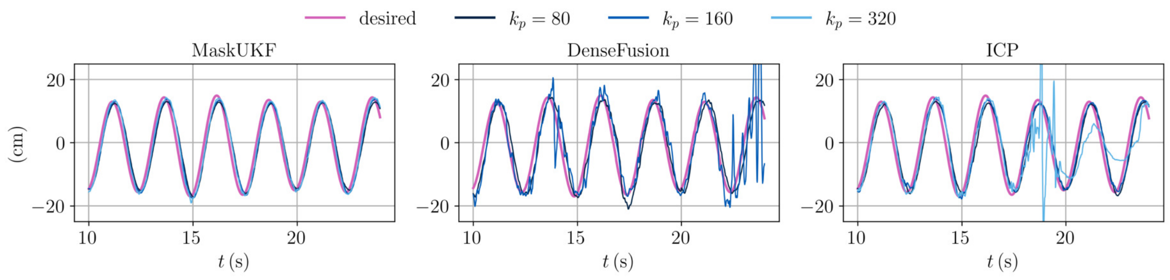


Figure 4.9 Evolution of the y coordinate of the end-effector for several algorithms and different proportional gains. The results of DenseFusion with $k_p = 320$ are not reported because that configuration is not achievable.

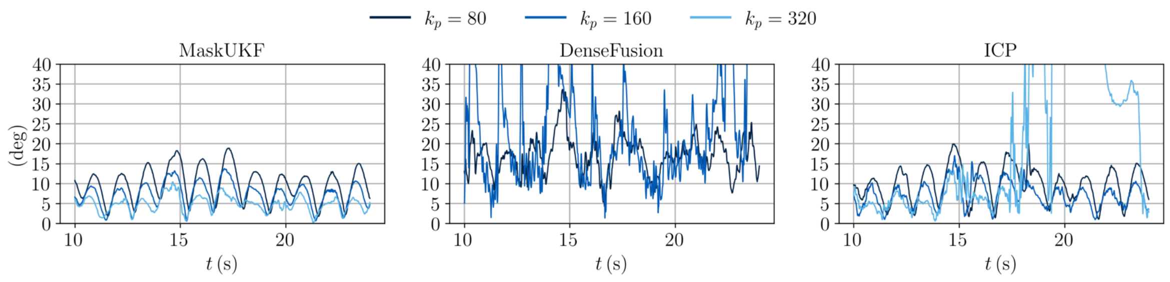


Figure 4.10 Evolution of the angular error of the end-effector for several algorithms and different proportional gains. The results of DenseFusion with $k_p = 320$ are not reported because that configuration is not achievable.

MaskUKF naturally provides a filtered estimate of the 6D velocity of the object, as shown in Sec. 4.2.8, that is required to implement the PD controller in Eq. (4.19). On the other hand, for DenseFusion and ICP the velocities are obtained by finite differentiation in this scenario. As we have discussed in Sec. 4.3.1, this kind of approximation is typically noisy, thus reducing the maximum possible amount of feedback hence the tracking performance of the closed loop system.

In the following we show samples of the actual trajectories of the end-effector for several choices of k_p, namely 80, 160 and 320.

In Fig. 4.9, we show the desired and achieved trajectory of the y coordinate for the first 25 seconds of the experiment. Using MaskUKF, the end-effector achieves a regular and smooth behavior in all cases. Using the ICP algorithm with $k_p = 320$, the end-effector fails to track the container after ≈ 18 seconds. Using DenseFusion with $k_p = 160$, the tracking is lost after ≈ 23 seconds.

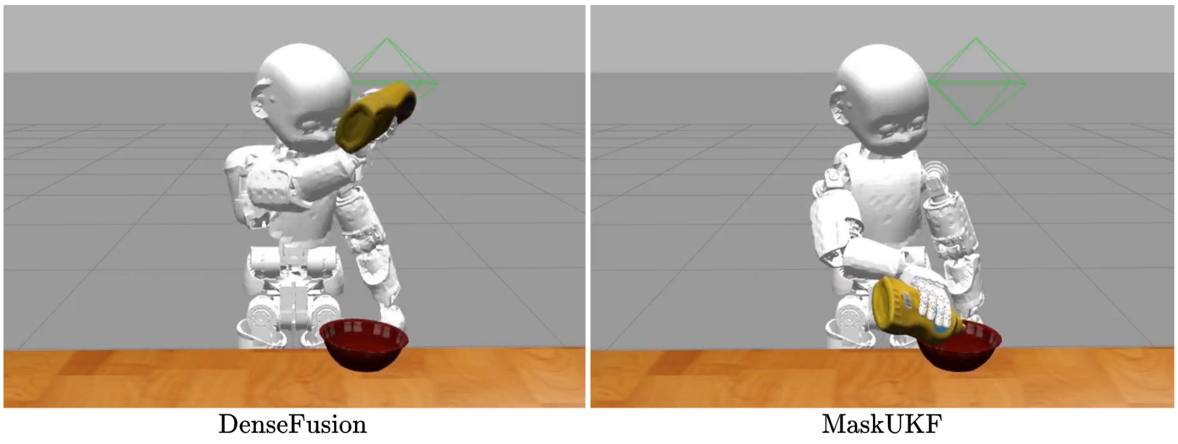


Figure 4.11 Comparison between MaskUKF and DenseFusion on the simulated task of following the red container. Using DenseFusion with $k_p = 160$ might produce unsafe motions for the robot and the surrounding environment.

In Fig. 4.10, we show the evolution of the angular error for the same choice of the proportional gains. Moving from $k_p = 80$ to $k_p = 160$ helps reducing the mean angular error when using MaskUKF and ICP. Moving to $k_p = 320$, MaskUKF reaches the minimum error. With ICP the error still reduces with respect to $k_p = 160$, however the tracking is lost after ≈ 18 seconds. When using DenseFusion with $k_p = 80$, the error is much higher than with MaskUKF and ICP. The error increases even more when moving to $k_p = 160$.

For both the Cartesian and angular errors, the results of DenseFusion with $k_p = 320$ are not reported as that configuration is not achievable.

In Fig. 4.11, we show examples of the end-effector configurations achieved when using MaskUKF and DenseFusion with $k_p = 160$. As can be seen, with DenseFusion the motion of the end-effector might be unsafe for the robot.

In summary, in this scenario MaskUKF provides better performance in terms of tracking precision and reliability when the amount of feedback given to the control system is increased. We mention that this result depends in large part on the adoption of a Kalman filtering framework which, leveraging even a rather simple motion model as in Equation (4.5), can help reducing the amount of noise in the estimates of both the pose and the velocity of the object.

Table 4.10 Parameters of the low-pass filter used to filter the output poses and velocities of DenseFusion and ICP.

Method	Signal	Values of α
DenseFusion	p_t	1.0
	v_t^m	0.5
	ω_t	0.1
	R_t	$(\alpha_x, \alpha_y, \alpha_z) = (1.0, 0.01, 1.0)$
ICP	p_t	1.0
	v_t^m	0.6
	ω_t	0.1
	R_t	$(\alpha_x, \alpha_y, \alpha_z) = (0.8, 0.05, 0.8)$

Results with low-pass filtering

In this section we discuss the results obtained when the output of DenseFusion and ICP is filtered using a low-pass filter, as discussed in Sec. 4.3.1. The parameters of the low pass-filter that we used in the experiments are reported in the Table 4.10. For the signals p_t, v_t^m and ω_t we used the same value of α for all the three channels. On the contrary, we used different values of the parameters for the rotation R_t, as we detail later.

The parameters were chosen empirically in order to remove as much as possible the noise in the estimated poses and velocities, obtained via finite differences, while at the same time avoiding to introduce too much delay and corrupt the information encoded in the signal.

For both DenseFusion and ICP, we decided not to filter the Cartesian position, i.e. $\alpha = 1.0$, as we found that the amount of noise in the signal was negligible. The linear velocity was instead moderately filtered. As regards the orientation of the object, we observed that both algorithms introduce a considerable amount of noise for the rotations about the y axis of the robot root frame. However, the container rotates about the x axis solely, as we specified in Sec. 4.3.1. For this reason, we could filter out almost all the noise of the y axis, i.e. we set $\alpha_y = 0.01$ and $\alpha_y = 0.05$ for DenseFusion and ICP, respectively. On the other hand, the amount of filtering for the x axis was limited in order to preserve the information on the actual orientation of the object. Finally, we observed that the angular velocity obtained using finite differences was considerably noisy for both algorithms, hence we had to filter it substantially, i.e. we set $\alpha = 0.1$.

In Fig. 4.12, we compare the Cartesian and angular errors obtained with MaskUKF, DenseFusion and ICP. For DenseFusion and ICP we report both results with and without low-pass filtering.

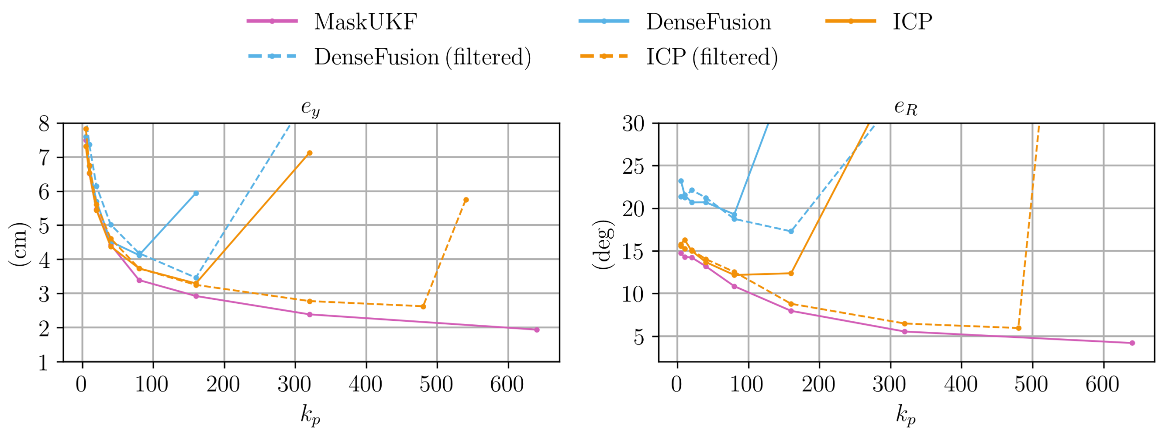


Figure 4.12 RMSE Cartesian and angular error for the object following experiment for varying proportional gains k_p. For DenseFusion and ICP the results with and without low-pass filtering are shown.

When using ICP, we can observe that the filtering stage helps reducing the tracking error. Indeed, for k_p up to ≈ 480 the error trend is very similar to that of MaskUKF. However, the absolute error is still slightly higher than that obtained with MaskUKF. Moreover, for slightly higher gains, the error increases considerably, especially the angular one.

A remarkable disadvantage of ICP is that, in our experiments, it required an ad-hoc tuning of the low-pass filter for the object rotation, i.e we had to pick different values for α_x, α_y and α_z (see Table 4.10). However, this choice was possible only because it was known that the object rotates about a specific axis. On the contrary, with MaskUKF we could obtain better results while using the same value for all the diagonal entries of the power spectral density Q_r (see Table 4.9), i.e. no prior information on the rotational motion of the object was used.

When considering DenseFusion, it can be seen that the filtering stage helps reducing the tracking error, however the reduction is much lower than that obtained with ICP, especially for the rotational components. This behavior depends on the fact that the amount of noise in the output of DenseFusion does not allow to recover the signal of interest completely. Of course, the low-pass filter helps reducing the negative effect of the noise on the control loop. In fact, it was possible to increase the proportional gain k_p up to 320, which was impossible in the non-filtered scenario. On the other hand, since the estimated poses and velocities are affected by important errors, which cannot be mitigated by the filtering stage, the tracking performance remains unsatisfactory.

In conclusion, these results show that approaches like MaskUKF, which apply a filtering framework directly to the measured point cloud, exhibit important advantages with respect to a pose estimation or pose registration method even when they are combined with an additional filtering stage afterwards.

4.3.4 Discussion on the requirements of the pouring task

We conclude the analysis of the results with a short discussion on the requirements of the pouring task and how they relate with the tracking errors achieved by the considered algorithms.

As we discussed in Sec. 4.3.1, the aim of the proposed experiment is to use the tracked object pose and velocity to close the control loop in order to follow the container and avoid possible leakages. In this respect, we deem useful to consider the percentage of frames, along the 1-minute-long experiment, for which the end-effector is closer to the center of the container than the radius r of the container itself:

$$\frac{|\{t \mid \|p_{t,ee} - p_t\| < r\}|}{T} \, 100. \tag{4.29}$$

Here, T is the total number of frames in an episode of the experiment and $|\cdot|$ indicates the cardinality of a set.

In Fig. 4.13, we plot the percentage defined in Eq. (4.29) for the radii $r \in (0, 10]$ cm and for several algorithms. For each algorithm, we considered the best configuration achieved according to the experiments presented in the previous sections. Specifically, for DenseFusion and ICP we considered the scenario where their output is filtered using a low-pass filter.

We remark that the plot was realized using the data of the experiment with the bowl container (024) from the YCB Model Set that has a radius of 7.5 cm. As a consequence, the results do not take into account the fact that, for smaller radii, the estimation error might get worse as the amount of occlusion of the object would increase. In this respect, these results represent an upper-bound of the performance that we expect to achieve.

Looking at the figure, we observe that with MaskUKF it might be possible to complete the pouring without leakages for radii as lower as ≈ 4 cm. This lower limit increases to ≈ 6 cm and ≈ 7.5 cm for ICP and DenseFusion, respectively. In the best case, i.e. 4cm, this might allow completing the pouring task for objects like a bowl, a

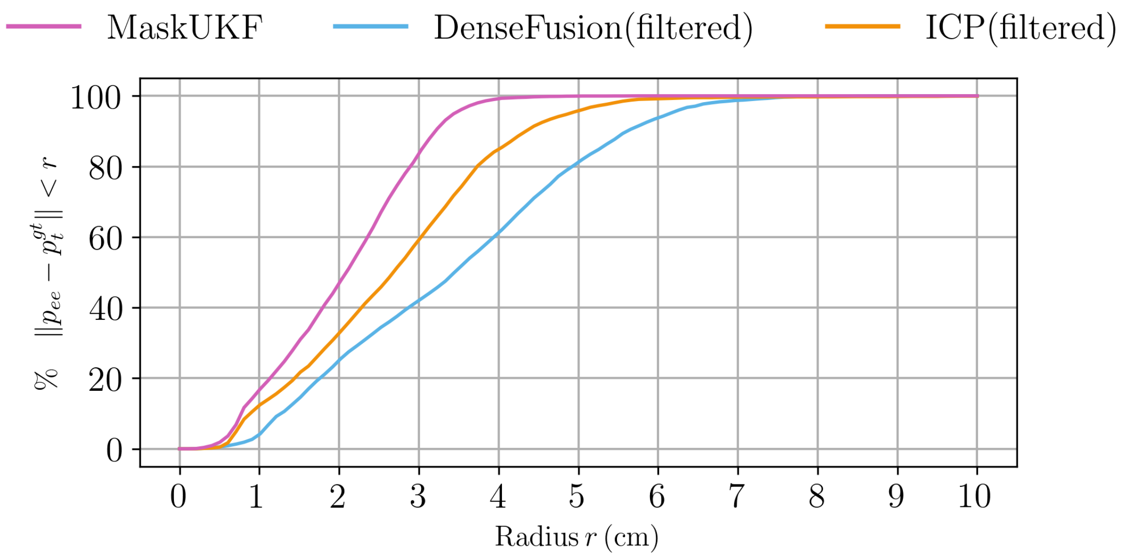


Figure 4.13 Percentage of the end-effector positions that fulfil the pouring requirements for several radii of the container and several algorithms.

mug or a glass. For smaller objects, e.g. a coffee cup, the pouring task would not be possible instead.

The above considerations, although preliminary, motivate the necessity to dedicate further research efforts in the development of algorithms for object pose tracking and estimation. This research would benefit future robots that are requested to carry out everyday tasks, such as the pouring task, even in presence of small objects.

Chapter 5

Optical flow-aided 6D Object Pose and Velocity Tracking

In the previous chapter, I described the importance of the 6D object pose and velocity tracking task and compared several pose estimation and tracking methods on the standard computer vision benchmark YCB-Video as well as on a robot motion control task. Independently from the performance achieved by each algorithm, one of the facts highlighted by the experimental evaluation is that object pose tracking methods might behave very differently when the initial conditions are perturbed with respect to the actual ground truth. In this regard, one might argue that the sensitivity to the initial conditions has to be carefully considered not only at the first time instant of the trajectory but also at any other instants. Indeed, track losses may occur not only at the initialization but also during the trajectory, especially if the object of interest is involved in dynamic motions characterized by moderate-to-fast velocities.

Unfortunately, most object pose estimation and tracking algorithms are tested on datasets, such as YCB-Video, which comprise scenes with slowly moving objects that might be inadequate to highlight the presence of track losses. Furthermore, several methods overlook the fact that, in case of limited computational resources or due to architectural complexities, large computation times can induce non-negligible delays and sparsity in the output, which negatively affect the overall tracking process. Additionally, most of the methods do not model the object motion using a specific prior or adopt a simple constant velocity model that might be inadequate in case of fast moving objects.

In this respect, several works from the past (Brox et al., 2006; Pauwels et al., 2016; Pressigout et al., 2008) proposed to integrate information on the actual pixels motion, obtained using the optical flow, to improve the tracking performance. The optical flow has been also recently combined with deep neural networks. DeepIM (Li et al., 2018) leverages the FlowNet (Dosovitskiy et al., 2015) network for optical flow prediction to iteratively refine 6D object poses. Following a different path, Zhang et al. (2020) combine object segmentation, optical flow and depth information to track the 6-DoF motion and the linear velocity of objects within automotive scenarios. Nevertheless, these scenarios are quite different from those considered in this Thesis.

Considering the aforementioned reasoning, during the PhD I dedicated considerable research efforts in designing a method that could exploit the availability of real-time algorithms for optical flow estimation in order to increase the tracking performance in scenarios with fast moving objects. Specifically, I proposed to combine real-time optical flow with *low frame rate* Convolutional Neural Networks (CNNs) for instance segmentation and 6D object pose estimation to achieve real-time 6D object pose and velocity tracking from RGB-D images. The proposed method (Piga et al., 2022) has been called "Real-time Optical Flow-aided 6D object pose and velocity Tracking" or ROFT.

Although the presented ideas are similar to those of (Zhang et al., 2020), one important difference is that ROFT explicitly accounts for delayed and low frame rate instance segmentation in case of limited computational resources.

The major contributions, discussed in the present chapter, are the following:

- The design of a mechanism that exploits real-time optical flow to synchronize *delayed* instance segmentation and 6D object pose estimation streams with a given RGB-D input stream.
- The design of an algorithm that combines optical flow and the aforementioned synchronized streams in a Kalman filtering approach that tracks both the 6D pose and the 6D velocity of a given object.
- The design of a novel a synthetic photorealistic dataset, Fast-YCB, comprising scenes with fast moving objects from the YCB Model Set (Calli et al., 2015) with linear velocities up to 63 cm/s and rotational velocities up to 266 degrees/s.

The proposed method has been compared with state-of-the-art 6D object pose trackers on the newly introduced Fast-YCB dataset and on the publicly available dataset for object and hand pose estimation HO-3D (Hampali et al., 2020).

A video of the experiments is available online[1]. The code used for the experiments is made publicly available for free with an Open Source license online[2]. The Fast-YCB dataset has been also made available online[3].

The remainder of this chapter is organized as follows. In the Secs. 5.1, 5.2, 5.3 and 5.4 the proposed method is presented in detail with the help of the theory presented in Chapters 2 and 3. The experimental results of the evaluation on the considered datasets are then provided in Sec. 5.5.

5.1 Description of the method

The proposed method tracks the pose T_t and the velocity $\mathcal{V}_t^m$ of an object of interest $\mathcal{O}$ using a cascaded filtering architecture consisting of a Linear Kalman Filter and an Unscented Kalman Filter. The inputs to the architecture are

- the RGB images I_t;
- the depth images D_t;
- the set of optical flow vectors $F_t = F_t(M_{t-1}, I_t, I_{t-1})$, as defined in Eq. (3.35), associated to the object;
- a source of object pose measurements $T_t^z \in \mathrm{SE}(3)$;
- a source of object segmentation masks M_t.

The superscript z in T_t^z indicates that this pose is measured and should not be confused with the tracked pose T_t. We notice that the definition of F_t depends on the mask at time $t-1$, instead of time t, because the optical flow vector field describes the motion of points from frame $t-1$ to frame t.

Although any source of masks M_t and poses T_t^z can be used, in this scenario it is assumed that both are only available at *low frame rate* resulting in non-synchronized and delayed streams M_t^d and T_t^d. A practical example, that we considered in this project, is that of deep learning-based networks for object segmentation and 6D pose estimation from RGB images as they could have non-negligible inference times.

[1]https://ieeexplore.ieee.org/ielx7/7083369/9568780/9568706/supp1-3119379.mp4?arnumber=9568706
[2]https://github.com/hsp-iit/roft
[3]https://github.com/hsp-iit/fast-ycb

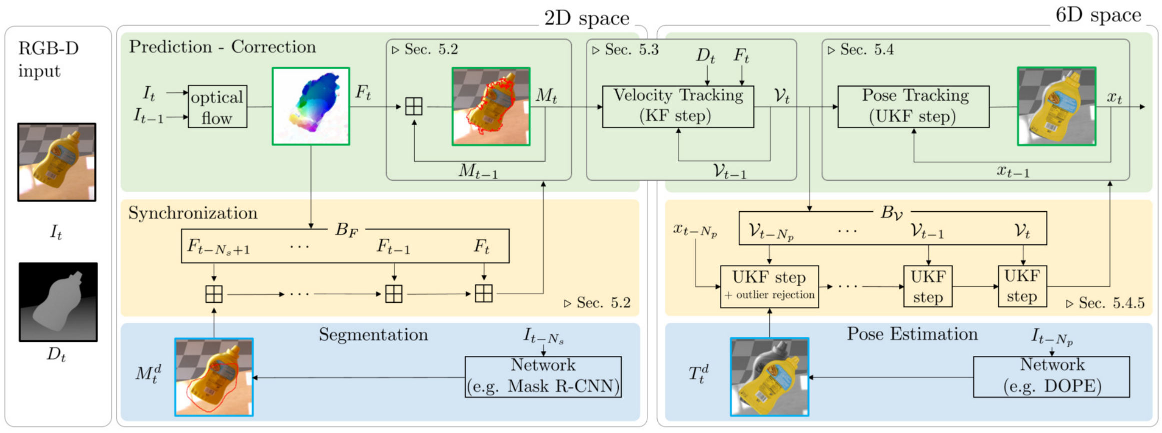


Figure 5.1 Illustration of the ROFT framework for 6D object pose and velocity tracking given the optical flow F_t and low frame rate segmentation masks M_t^d and object poses T_t^d.

In the remainder of this section, all the components of the tracking algorithm are described. Specifically, the proposed architecture consists of three stages. First the optical flow F_t is used to synchronize the stream of masks M_t^d with the input RGB stream I_t. Next, the optical flow F_t, the depth D_t and the synchronized masks M_t are combined to track the 6D object spatial velocity $\mathcal{V}_t$ within a linear Kalman Filter. In the third stage, the estimated velocities $\mathcal{V}_t$ are fused with the low frame rate pose measurements T_t^d within an Unscented Kalman Filter in order to track the 6D pose T_t of the object.

An overview of the pipeline is depicted in Fig. 5.1. The three stages are detailed in the following Secs. 5.2, 5.3 and 5.4.

5.2 Optical flow-aided object segmentation

Given the input RGB stream I_t, the optical flow frames F_t, that are synchronized with I_t, are combined with the low frame rate and non-synchronized stream of segmentation masks M_t^d in order produce a stream of masks M_t that is synchronized with I_t.

Let $M \boxplus F$ be the operator

$$M \boxplus F = \{(\mathrm{u}, \mathrm{v}) + F(\mathrm{u}, \mathrm{v}) \quad \forall (\mathrm{u}, \mathrm{v}) \in M\}, \tag{5.1}$$

which propagates the pixels coordinates in the mask M according to the flow vectors $F(\mathrm{u}, \mathrm{v})$.

The synchronized stream M_t is initialized using the first available mask M_0^d, under the assumption that the object is stationary at time $t = 0$. In order to provide a continuous and synchronized stream of masks M_t, the masks are recursively updated using the optical flow

$$M_t = M_{t-1} \boxplus F_t, \tag{5.2}$$

until a new mask M_t^d from the network is available. Meanwhile, the frames F_t are stored in a buffer B_F.

After N_s steps, it is assumed that a new mask M_t^d, associated to the RGB frame I_{t-N_s}, is available. Since the received mask is associated to a *past* RGB frame, it is iteratively propagated forward in time using the buffered flow frames from B_F up to time t

$$M_t = \big(\big(\big(M_t^d \boxplus F_{t-N_s+1}\big) \boxplus \ldots\big) \boxplus F_{t-1}\big) \boxplus F_t. \tag{5.3}$$

From here, the propagation scheme in Eq. (5.2) is reactivated until a new mask from the network is available again.

The propagation process in Eq. (5.2) is illustrated in Fig. 5.1, on the left, in the green “Prediction - Correction” block. In the same figure, on the left, we illustrate the synchronization process in Eq. (5.3) in the yellow “Synchronization” block.

We remark that the synchronized masks might be discontinuous due to the noise in the optical flow vectors. Nevertheless, the proposed approach relies more on the availability of pixel coordinates in the image I_t belonging to the surface of the object rather than the exact shape of the segmentation mask.

5.3 6D object spatial velocity tracking

Having available a stream of depth images D_t, optical flow frames F_t and synchronized segmentation masks M_t, the second stage of the architecture tracks the 6D spatial velocity of the object $\mathcal{V}_t$ from optical flow measurements using a linear Kalman Filter.

In the remainder of this section, all the components of the filter are described.

5.3.1 Description of the state and the measurements

The state to be tracked $\mathcal{V}_t$ is defined as

$$\mathcal{V}_t = \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} \in \mathbb{R}^6, \tag{5.4}$$

corresponding to the object *spatial* velocity as defined in Eq. (3.23). We recall that, according to the definition, ω_t is the angular velocity of the object and v_t is the velocity of a point, belonging to the object, which coincides with the origin of the camera instantaneously.

The optical flow $F_t = F_t(M_{t-1}, I_t, I_{t-1})$ is used as a measurement

$$z_t = \begin{bmatrix} z_{1,t}^T & \cdots & z_{j,t}^T & \cdots & z_{L_t,t}^T \end{bmatrix}^T \in \mathbb{R}^{2L_t}, \tag{5.5}$$

where each subvector $z_{j,t} \in \mathbb{R}^2$ is one of the flow vectors $F_t(\mathrm{u}, \mathrm{v})$ in the set F_t and L_t is the cardinality of the set at time t. The cardinality changes over time as the number of pixels in the mask M_{t-1} depends on the object pose and on possible object occlusions. We remark that the adopted tracking framework, i.e. the linear Kalman filter, naturally handles time-varying measurement models.

5.3.2 Motion model

We assume that the underlying dynamics of the state vector $\mathcal{V}$ is described by the simple motion model

$$\begin{aligned} \mathcal{V}_t &= f(\mathcal{V}_{t-1}, w_{t-1}), \\ &= \mathcal{V}_{t-1} + w, \\ w &\sim \mathcal{N}\left(0, \operatorname{diag}(Q_v, Q_\omega)\right), \end{aligned} \tag{5.6}$$

where the velocity increments $\mathcal{V}_t - \mathcal{V}_{t-1}$ are Gaussian with covariances $Q_v \in \mathbb{R}^{3\times 3}$ and $Q_\omega \in \mathbb{R}^{3\times 3}$ associated to the linear and angular velocity, respectively.

5.3.3 Measurement model

The specification of the measurement model accounts for the definition of the function h in Equation (2.8). The role of h is to establish the relationship between the state $\mathcal{V}_t$ and the optical flow vectors that we expect to observe. To this end, it is possible to rely on the definition of predicted optical flow given the velocity $\mathcal{V}_t$ and the depth image D_t

$$F_t(M_{t-1}, \mathcal{V}_t, D_t) = \left\{ \begin{bmatrix} \Delta_{\mathrm{u}}(\mathcal{V}_t, D_t) \\ \Delta_{\mathrm{v}}(\mathcal{V}_t, D_t) \end{bmatrix} \quad \forall (\mathrm{u}, \mathrm{v}) \in M_{t-1} \right\}, \tag{5.7}$$

provided in Eq. (3.43). By recalling the expressions of Δ_{u} and Δ_{v}, as per Eq. (3.42), and by stacking them in a single vector, the following measurement model is obtained:

$$\begin{aligned} z_t &= h_t(\mathcal{V}_t, \nu_t) \\ &= \begin{bmatrix} \dots & \begin{bmatrix} \Delta_{\mathrm{u}}(\mathcal{V}_t, D_t) \\ \Delta_{\mathrm{v}}(\mathcal{V}_t, D_t) \end{bmatrix}^T & \dots \end{bmatrix}^T + \nu_t \\ &= J(M_{t-1}, D_t)\mathcal{V}_t + \nu_t, \\ \nu_t &\sim \mathcal{N}(0, R_t), \end{aligned} \tag{5.8}$$

where

$$\begin{aligned} J(M, D) &= \begin{bmatrix} \dots \\ J_{\mathrm{uv}}(D) \\ \dots \end{bmatrix} \quad (\mathrm{u}, \mathrm{v}) \in M, \\ J_{\mathrm{uv}}(D) &= \begin{bmatrix} J_v(d) & J_\omega \end{bmatrix} \Delta_t \quad d \in D, \\ J_v(d) &= \begin{bmatrix} \frac{f_x}{d} & 0 & \frac{-(\mathrm{u}-c_x)}{d} \\ 0 & \frac{f_y}{d} & \frac{-(\mathrm{v}-c_y)}{d} \end{bmatrix}, \\ J_\omega &= \begin{bmatrix} \frac{-(\mathrm{u}-c_x)(\mathrm{v}-c_y)}{f_y} & \frac{f_x^2+(\mathrm{u}-c_x)^2}{f_x} & \frac{-(\mathrm{v}-c_y)f_x}{f_y} \\ \frac{-f_y^2-(\mathrm{v}-c_y)^2}{f_y} & \frac{(\mathrm{u}-c_x)(\mathrm{v}-c_y)}{f_x} & \frac{(\mathrm{u}-c_x)f_y}{f_x} \end{bmatrix}, \\ R_t &= \mathrm{diag}(\sigma_1^2 I_2, \dots, \sigma_{L_t}^2 I_2). \end{aligned} \tag{5.9}$$

In summary, the optical flow predictions for all the pixels belonging to the mask M_{t-1} are expressed using a *linear* measurement model with additive noise. The size of z_t is equal to the cardinality $2|M_{t-1}|$, i.e. twice the number of pixels belonging to the mask.

We mention that the possibility to express the optical flow prediction as a linear function of the state depends on the choice to represent the velocity using the *spatial* representation $\mathcal{V}$. Indeed, it can be shown that, by choosing the mixed representation $\mathcal{V}^m$, instead, the prediction would also require to include the object pose T_t in the state and the model would become nonlinear.

5.3.4 Choice of the filtering algorithm

In order to track the velocity of the object, we adopted the standard Linear Kalman Filter for the following reasons:

- the motion and measurement models are both linear;
- the availability of a sequential variant of the algorithm, discussed in Sec. 2.2.2, that efficiently processes high-dimensional measurements as in the case of the optical flow vectors.

Given the above motivations, we adopted the Algorithm 6 "Sequential Kalman filter" to accomplish this task.

5.4 6D object pose tracking

In this section, the last stage of the architecture is presented. It fuses non-synchronized pose measurements T_t^d, available at low frame rate, with the estimated velocities $\mathcal{V}_t$, available for all frames, using an Unscented Kalman Filter in order to track the 6D pose of the object. In the remainder of this section, all the components of the filter are described.

5.4.1 Description of the state and the measurements

The state to be tracked x_t is defined as

$$x_t = \begin{bmatrix} p_t \\ v_t^m \\ q_t \\ \omega_t \end{bmatrix} \tag{5.10}$$

which comprises the Cartesian position $p_t \in \mathbb{R}^3$, a unitary quaternion $q_t \in \mathbb{H}$ for the 3D orientation, the linear mixed velocity $v^m \in \mathbb{R}^3$ and the angular velocity $\omega \in \mathbb{R}^3$.

In order to track the state, we use the spatial velocity $\mathcal{V}_t$, estimated as discussed in Sec. 5.3, and the poses T_t^d as measurements:

$$z_t = \begin{bmatrix} p(T_t^d) \\ q(T_t^d) \\ v(\mathcal{V}_t) \\ \omega(\mathcal{V}_t) \end{bmatrix}. \tag{5.11}$$

Here, $p(T_t^d)$ and $q(T_t^d)$ are the Cartesian position and the quaternion components of the measured pose T_t^d, while $v(\mathcal{V}_t)$ and $\omega(\mathcal{V}_t)$ are the linear and angular velocity components of the measured spatial velocity $\mathcal{V}_t$.

As explained in the introduction, the pose measurements are available at low frame rate. Specifically, it is assumed that they are available every N_p steps. When they are not available, the measurements reduce to the velocity only

$$z_t = \begin{bmatrix} v(\mathcal{V}_t) \\ \omega(\mathcal{V}_t) \end{bmatrix} \in \mathbb{R}^6. \tag{5.12}$$

5.4.2 Motion model

We assume that the state x_t evolves according to the following model

$$\begin{aligned} x_t &= f(x_{t-1}, w_{t-1}) \\ &= F_p x_{t-1} + F_q(\omega_{t-1}) x_{t-1} + \begin{bmatrix} w_v \\ 0_4 \\ w_\omega \end{bmatrix}, \\ w_v &\sim \mathcal{N}(Q_v), \\ w_\omega &\sim \mathcal{N}(Q_\omega). \end{aligned} \tag{5.13}$$

For the positional part, a White Noise Acceleration model (WNA) is used with F_p the state transition matrix

$$F_p = \begin{bmatrix} I_3 & I_3\Delta_t & 0 \\ 0 & I_3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{13\times 13}. \tag{5.14}$$

The vector $w_v \in \mathbb{R}^6$ is the noise associated to the positional part with noise covariance matrix

$$Q_v = \begin{bmatrix} Q_p\frac{\Delta_t^3}{3} & Q_p\frac{\Delta_t^2}{2} \\ Q_p\frac{\Delta_t^2}{2} & Q_p\Delta_t \end{bmatrix} \in \mathbb{R}^{6\times 6}, \tag{5.15}$$

where Q_p is the power spectral density of the noise.

For the rotational part, a standard quaternion kinematics transition matrix (Chiella et al., 2019) is adopted

$$F_q(\omega) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \cos(\|\omega\|\frac{\Delta_t}{2})I_4 + \frac{\sin(\|\omega\|\frac{\Delta_t}{2})}{\|\omega\|}\Omega(\omega) & 0 \\ 0 & 0 & I_3 \end{bmatrix} \in \mathbb{R}^{13\times 13}, \tag{5.16}$$

where $\Omega(\omega)$ is as follows

$$\Omega(\omega) = \begin{bmatrix} -\omega_x & -\omega_y & -\omega_z & 0 \\ \omega_x & 0 & -\omega_z & \omega_y \\ \omega_y & \omega_z & 0 & -\omega_x \\ \omega_z & -\omega_y & \omega_x & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4}. \tag{5.17}$$

The vector $w_\omega \in \mathbb{R}^3$ is the noise associated to the angular velocity with noise covariance matrix Q_ω. This noise contribution affects the quaternion indirectly via the transition matrix F_q.

5.4.3 Measurement model

The measurement model relating the state x_t with the measurements z_t is time-variant as it depends on the availability of the pose measurements T_t^d, available every N_p steps:

$$\begin{aligned} z_t &= h_t(x_t, \nu_t) \\ &= \begin{cases} \begin{bmatrix} p_t \\ q_t \\ v_t^m + \omega_t \times (-p_t) \\ \omega_t \end{bmatrix} \oplus \begin{bmatrix} \nu_p \\ \nu_q \\ \nu_v \\ \nu_\omega \end{bmatrix}, & \text{if } t = kN_p \quad k \in \mathbb{N} \\ \begin{bmatrix} v_t^m + \omega_t \times (-p_t) \\ \omega_t \end{bmatrix} + \begin{bmatrix} \nu_v \\ \nu_\omega \end{bmatrix}, & \text{otherwise} \end{cases}, \\ \nu_* &\sim \mathcal{N}(0, R_*), \end{aligned} \tag{5.18}$$

where ν_* are the noise contributions for each component of the measurement with R_* the associated covariance matrices. It is recalled that $\oplus$ is the addition operator, defined in (2.114), that takes into account the quaternion arithmetic. In this regard, the noise $\nu_q \in \mathbb{R}^3$ is to be considered as a perturbation vector in the Lie algebra of the quaternions.

We remark that the extra term $\omega_t \times (-p_t)$ makes the measured *spatial* velocity $v(\mathcal{V}_t)$ comparable with the *mixed* velocity v_t^m belonging to the state x_t.

5.4.4 Choice of the filtering algorithm

The actual tracking of the state x_k is performed using an Unscented Kalman Filter. This choice is dictated by the nonlinearity of the motion and measurement models. Hence, we adopted the Algorithm 5 “Unscented Kalman filter” to address this task. Furthermore, given that the state x_t and the measurement z_t contain quaternion components, the algorithm has been modified to take them into account as in the Algorithm 9 “Unscented Kalman filtering using unit quaternions”.

5.4.5 Synchronization of pose measurements

The pose measurements T_t^d are available at low frame rate. We assume that a new pose is available each every N_p steps, hence it is associated to the RGB frame I_{t-N_p} and could not be used directly as measurement at time t. For this reason, when a new pose measurement T_t^d becomes available, the filter is reset to the previous state x_{t-N_p} and a single UKF update is executed using the pose T_t^d and the velocity $\mathcal{V}_{t-N_p}$ as measurements. It is assumed that previous velocity measurements are stored in a buffer $B_{\mathcal{V}}$. After that, additional UKF update steps are executed using the remaining buffered velocities in $B_{\mathcal{V}}$ from $\mathcal{V}_{t-N_p+1}$ to $\mathcal{V}_t$ in order to obtain a synchronized estimate x_t. This process is illustrated in Fig 5.1, on the right, in the yellow "Synchronization" block.

5.4.6 Rejection of pose measurements outliers

The pose measurements T_t^d can be affected by outliers which might induce biases in the estimates in Gaussian filters such as the Kalman filter (Issac et al., 2016). A practical example is that of deep learning-based networks for 6D pose estimation whose output might contain spikes, as shown in Sec. 4.2.8 and in Figs. 1.1 and 1.2.

To overcome this issue, when using T_t^d in the synchronization process described in Sec. 5.4.5, two UKF updates are actually executed resulting in two hypotheses. One of the two considers both pose and velocity measurements, the other considers only the velocity. Assuming that a 3D triangle mesh of the object (see Sec. 3.2) is available, two synthetic depth maps, $D^{\mathcal{V},T}$ and $D^{\mathcal{V}}$ respectively, are rendered for both hypotheses and compared to the measured depth D_{t-N_p}. The comparison is quantified in terms of two errors $e^{\mathcal{V},T} = e(D^{\mathcal{V},T})$ and $e^{\mathcal{V}} = e(D^{\mathcal{V}})$ where

$$e(D) = \sum_{(\mathrm{u},\mathrm{v}) \in M_{t-N_p}} |D_{t-N_p}(\mathrm{u},\mathrm{v}) - D(\mathrm{u},\mathrm{v})|. \tag{5.19}$$

If the two errors differ considerably, according to a threshold γ,

$$e^{\mathcal{V},T} > \gamma e^{\mathcal{V}}, \tag{5.20}$$

the pose T_t^d is marked as outlier and skipped. This process is illustrated in Fig. 5.2.

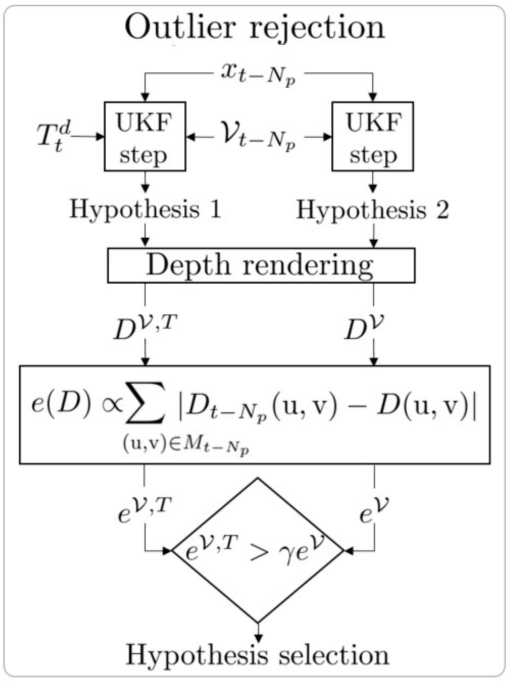


Figure 5.2 Illustration of the outlier rejection mechanism employed in ROFT.

5.4.7 Resulting algorithm

The proposed algorithm for 6D object pose and velocity tracking is summarized in the table entitled “ROFT: Real-time Optical Flow-aided 6D Object Pose and Velocity Tracking”. The overall framework is also depicted in Fig. 5.1.

ROFT: Real-time Optical Flow-aided 6D Object Pose and Velocity Tracking

```
 1: procedure ROFT($\mu_0, P_0, \mathcal{V}_0, P_{\mathcal{V},0}, M_0, N_s, N_p, \gamma$)
 2:     $\mu_{t=0} = \mu_0$                                    ▷ Initialization of the pose tracker
 3:     $P_{t=0} = P_0$
 4:     $\mathcal{V}_{t=0} = \mathcal{V}_0$                    ▷ Initialization of the spatial velocity tracker
 5:     $P_{\mathcal{V},t=0} = P_{\mathcal{V},0}$
 6:     $M_{t=0} = M_0$                                        ▷ Initialization of the mask stream
 7:     for each $(I_t, D_t[, M^d_t, T^d_t])$ do
 8:         $F_t = F_t(M_{t-1}, I_t, I_{t-1})$                 ▷ Optical flow evaluation
```

Stage 1 - Optical flow-aided object segmentation

```
 9:         if $M^d_t$ available then
10:             $\{F_{t-N_s+1}, \cdots, F_t\} \leftarrow B_F$
11:             $M_t = \Big(\big(\big(M^d_t \boxplus F_{t-N_s+1}\big) \boxplus \ldots\big) \boxplus F_{t-1}\Big) \boxplus F_t$
12:         else
13:             $M_t = M_{t-1} \boxplus F_t$
14:             $B_F = B_F \cup \{F_t\}$
15:         end if
```

Stage 2 - 6D object spatial velocity tracking

```
16:         $z_t \leftarrow F_t(M_{t-1}, I_t, I_{t-1})$
17:         $(\mathcal{V}_t, P_{\mathcal{V},t}) = \text{SequentialKF}(f(\cdot), h(\cdot, D_t), \mathcal{V}_{t-1}, P_{\mathcal{V},t-1}, z_t)$
18:                                                        ▷ See Algorithm 6 for SequentialKF($\cdot$)
19:                                                        ▷ See 5.3.2 for $f(\cdot)$
20:                                                        ▷ See 5.3.3 for $h(\cdot)$
```

Stage 3 - 6D object pose tracking

```
21:         if $T^d_t$ available then
22:             $\mathcal{V}_{t-N_p} \leftarrow B_\mathcal{V}$
23:             $z \leftarrow \{T^d_t, \mathcal{V}_{t-N_p}\}$
24:             $(\mu_{t-N_p+1}, P_{t-N_p+1}) \leftarrow \text{OutlierRejection}(\mu_{t-N_p}, z, D_{t-N_p}, M_{t-N_p}, \gamma)$
25:                                                        ▷ See 5.4.6
26:             for each $\mathcal{V}_j \in B_\mathcal{V},\ j \in [t - N_p + 1, t]$ do    ▷ See 5.4.5
27:                 $z \leftarrow \mathcal{V}_j$
28:                 $(\mu_j, P_j) = \text{UKF}(f(\cdot), h(\cdot), \mu_{j-1}, P_{j-1}, z)$
29:             end for
30:         else
31:             $z_t \leftarrow \mathcal{V}_t$
32:             $(\mu_t, P_t) = \text{UKF}(f(\cdot), h(\cdot), \mu_{t-1}, P_{t-1}, z_t)$
33:             $B_\mathcal{V} = B_\mathcal{V} \cup \{\mathcal{V}_t\}$
34:         end if
35:                                                        ▷ See Algorithms 5 and 9 for UKF($\cdot$)
36:                                                        ▷ See 5.4.2 for $f(\cdot)$
37:                                                        ▷ See 5.4.3 for $h(\cdot)$
38:     end for
39: end procedure
```

5.5 Experimental evaluation

This section reports on the performance achieved by the proposed method on the Fast-YCB (Piga et al., 2022) and HO-3D (Hampali et al., 2020) datasets and provides comparisons with state-of-the-art algorithms for 6D object pose tracking.

The analysis includes the standard ADD-AUC metric (Xiang et al., 2018), as well as the pose RMSE (Bar-Shalom et al., 2002) tracking errors. Velocity RMSE tracking errors are also discussed for the Fast-YCB dataset for which ground-truth velocities are available. The evaluation also includes the results of several ablation studies where all the key components of the pipeline are selectively disabled in order to assess their individual importance. We also provide qualitative results on the adopted datasets and on object gaze-tracking and grasping experiments executed on the iCub humanoid robot platform (Metta et al., 2010).

5.5.1 Description of the evaluation datasets

Several standard datasets for 6D object pose estimation have been proposed in the literature such as T-LESS (Hodan et al., 2017), LineMOD (Hinterstoisser et al., 2012) and YCB-Video (Xiang et al., 2018). Among them, YCB-Video has also been adopted to benchmark 6D object pose trackers such as PoseRBPF (Deng et al., 2019) and se(3)-TrackNet (Wen et al., 2020a). However, the provided sequences are characterized by slowly varying poses and low velocities, hence not necessarily ideal to benchmark 6D object pose tracking algorithms.

Fast-YCB

Given the aforementioned motivations, I proposed a new dataset, called Fast-YCB (Piga et al., 2022), consisting of six photorealistic synthetic sequences, each comprising fast motions of a single object taken from the YCB Model Set (Calli et al., 2015) in a tabletop scenario. The six objects have been selected in order to consider a variety of shapes (box, cylinders or irregular), sizes (small, medium and large) and textures (see Fig. 5.3). Each sequence is rendered with bright static light conditions and provided with 30 frames per second (fps) 1280x720 RGB-D frames and exact ground-truth 6D object poses and velocities. Optical flow frames are also provided as part of the dataset. We remark that object trajectories, used in the rendering process, were captured from real-world hand-object manipulation using a marker board attached to the manipulated object.

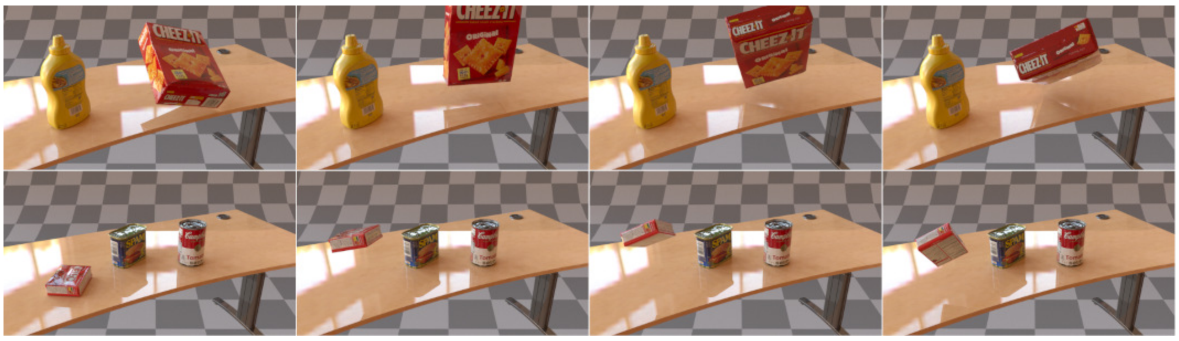


Figure 5.3 Sample frames taken from the Fast-YCB dataset. The time length of the reported frames is approximately 1 second.

HO-3D

We extended the analysis on real-world data by considering the public HO-3D (Hampali et al., 2020) dataset, comprising real images of hand-held YCB objects that are involved in challenging rotational motions. Furthermore, this dataset presents more variability in terms of background, object occlusion and lightning conditions. Each sequence is provided with 30 fps 640x480 RGB-D frames. This dataset offers ground-truth 6D object pose labels for benchmarking while it lacks annotations of the 6D object velocity.

For the evaluation, 18 among all the sequences of the dataset have been selected, excluding those containing discontinuities in the pose trajectory, hence not suitable for tracking purposes.

Additional sequences

For further qualitative evaluation, some real unlabeled sequences acquired with an Intel RealSense D415 camera, representing a scenario similar to that of Fast-YCB, were also collected.

5.5.2 Description of the compared algorithms

The proposed method is compared against two state-of-the-art object pose trackers from RGB-D images: PoseRBPF (Deng et al., 2019) and se(3)-TrackNet (Wen et al., 2020a). These algorithms have been already presented in the previous chapter. The reader is referred to the Sec. 4.2.2 for further information.

5.5.3 Description of the experimental setup

Source of segmentation masks

As source of segmentation masks M_t^d, we used the state-of-the-art Mask R-CNN instance segmentation network (He et al., 2017), pre-trained on the COCO dataset (Lin et al., 2014). In order to obtain a segmentation model for the objects of interest from the YCB Model Set (Calli et al., 2015), we fine tuned the network on a synthetic dataset generated using the BOP BlenderProc rendering engine (Denninger et al., 2020).

Source of pose measurements

Object pose measurements T_t^d were obtained using the 6D object pose estimation network DOPE (Tremblay et al., 2018), originally trained on purely synthetic data. This algorithm has been presented in the previous Chapter. The reader is referred to the Sec. 4.2.2 for further information.

Retraining of the compared algorithms

Since all the aforementioned methods were originally trained on purely synthetic images comprising the objects of interest, DOPE, PoseRBPF and se(3)-TrackNet have not been retrained in order to execute the experiments, instead the original weights provided by the authors were used.

Considered frame rates, initialization and re-initialization

All the tracking methods involved in the comparison were initialized using the first available pose T_0^d from the DOPE pose estimation network.

Although the authors of se(3)-TrackNet suggest that the algorithm can be executed without periodic re-initialization, it was experimentally observed that it looses track on both the considered datasets. For this reason, and in order to have a fair comparison with ROFT, the predictions from DOPE, delayed and at low frame rate, were used to re-initialize se(3)-TrackNet periodically. It is worth mentioning that the re-initialization stage does not affect the real-time performance.

Differently from se(3)-TrackNet, PoseRBPF cannot run in real-time and it is reported to achieve the highest frame rate of 17.5 fps in the configuration with 50 particles (Deng et al., 2021). In the adopted setup, this configuration could not achieve

Table 5.1 Parameter set for ROFT for the experiments on the Fast-YCB and HO-3D datasets.

Pose tracker parameters	
Parameter	Values
$P_{0\,(1:3,1:3)}$	$\mathrm{diag}(10^{-3}, 10^{-3}, 10^{-3})\,\mathrm{m}^2$
$P_{0\,(4:6,4:6)}$	$\mathrm{diag}(10^{-3}, 10^{-3}, 10^{-3})\,(\mathrm{m/s})^2$
$P_{0\,(7:9,7:9)}$	$\mathrm{diag}(10^{-3}, 10^{-3}, 10^{-3})\,(\mathrm{rad/s})^2$
$P_{0\,(10:12,10:12)}$	$\mathrm{diag}(10^{-3}, 10^{-3}, 10^{-3})\,(\mathrm{rad/s})^2$
Q_p	$\mathrm{diag}(1.0, 1.0, 1.0)\,(\mathrm{m/s})^2/\mathrm{s}$
Q_ω	$\mathrm{diag}(1.0, 1.0, 1.0)\,(\mathrm{rad/s})^2$
R_p	$\mathrm{diag}(10^{-3}, 10^{-3}, 10^{-3})\,\mathrm{m}^2$
R_q	$\mathrm{diag}(10^{-4}, 10^{-4}, 10^{-4})\,(\mathrm{rad/s})^2$
R_v	$\mathrm{diag}(10^{-1}, 10^{-1}, 10^{-1})\,(\mathrm{m/s})^2$
R_ω	$\mathrm{diag}(10^{-4}, 10^{-4}, 10^{-4})\,(\mathrm{rad/s})^2$
γ	2.0

Velocity tracker parameters	
Parameter	Values
$\mathcal{V}_{0\,(1:3)}$	$\mathrm{diag}(0.0, 0.0, 0.0)\,\mathrm{m/s}$
$\mathcal{V}_{0\,(4:6)}$	$\mathrm{diag}(0.0, 0.0, 0.0)\,\mathrm{rad/s}$
$P_{\mathcal{V},0\,(1:3,1:3)}$	$\mathrm{diag}(10^{-3}, 10^{-3}, 10^{-3})\,(\mathrm{m/s})^2$
$P_{\mathcal{V},0\,(4:6,4:6)}$	$\mathrm{diag}(10^{-3}, 10^{-3}, 10^{-3})\,(\mathrm{rad/s})^2$
Q_v	$\mathrm{diag}(10^{-1}, 10^{-1}, 10^{-1})\,(\mathrm{m/s})^2$
Q_ω	$\mathrm{diag}(10^{-1}, 10^{-1}, 10^{-1})\,(\mathrm{rad/s})^2$
R_t	$\mathrm{diag}(1.0, \cdots, 1.0)\,\mathrm{px}^2$

more than 13.5 fps, which implies to execute a filtering step every 2.2 frames. As this is not feasible due to the discrete nature of a dataset sampled at 30 fps, PoseRBPF was used by executing a filtering step every 2 frames. The resulting configuration corresponds to an equivalent frame rate of 15 fps.

Similarly to se(3)-TrackNet, it was observed that PoseRBPF can lose the track. However, the provided mechanism for re-initialization requires more than 2 seconds on average, due to the need for exploring the state space, making it incompatible with the claimed frame rate. For this reason the re-initialization mechanism of PoseRBPF was disabled during the experiments.

Additional implementation details

It is assumed that the RGB-D input stream I_t is available at 30 fps. In order to obtain the optical flow frames F_t, the readily available NVIDIA Optical Flow SDK (NVIDIA, 2021) was adopted as it provides real-time performance with small CUDA cores utilization.

Given that the proposed approach considers a scenario where the output of the segmentation and pose estimation networks is only available at low frame rate, Mask R-CNN and DOPE were executed at 5 fps. This configuration corresponds to a delay of 200 ms or, equivalently, to a value of $N_s = N_p = 6$.

The actual values of the covariance matrices Q_* and R_*, reported in Table 5.1, were chosen empirically in order to ensure a fast response with minimal overshoot, residual oscillations and noise in the output. Remarkably, the same set of parameters has been used in all the experiments independently of the considered dataset or object.

Table 5.2 Results on the Fast-YCB dataset: ADD-AUC and RMSE positional and angular errors for several methods.

metric	ADD-AUC, $\epsilon = 10$ cm				Positional RMSE [cm]				Rotational RMSE [deg]			
method	DOPE	ROFT	Pose RBPF	se(3)-TrackNet	DOPE	ROFT	Pose RBPF	se(3)-TrackNet	DOPE	ROFT	Pose RBPF	se(3)-TrackNet
003	54.92	**78.50**	68.94	63.02	5.1	**2.5**	2.6	7.9	28.325	**7.545**	38.455	31.729
004	60.01	81.15	**82.78**	73.70	5.1	2.4	**1.9**	4.4	34.636	**8.391**	17.073	31.623
005	64.14	79.00	75.93	**80.82**	4.7	2.9	**1.1**	3.3	29.556	**16.571**	63.811	26.282
006	57.20	73.10	**82.92**	74.83	8.6	3.1	**2.0**	11.9	36.132	**13.292**	18.418	35.480
009	60.01	**74.26**	11.32	69.00	34.4	**4.7**	13.3	35.5	27.936	**16.368**	65.857	32.311
010	57.03	73.87	**87.29**	71.30	6.1	3.1	**1.2**	19.2	31.444	**10.838**	23.758	44.712
ALL	58.83	**76.59**	68.10	72.06	15.1	**3.2**	5.7	17.6	31.491	**12.675**	42.979	34.155

5.5.4 Description of the adopted metrics

Following prior works (Deng et al., 2019; Wen et al., 2020a), the ADD-AUC metric (see Sec. 3.4.2) with the threshold ϵ set to 10 cm is reported for each object and for the union of all the objects, in order to determine the overall performance of the algorithms. The ADI-AUC metric for symmetric objects is not considered given that all the considered objects are not symmetric and all the algorithms have access to the RGB information.

Similarly to the experimental evaluation in Chapter 5, the evaluation also include the positional and rotational RMSE metrics (see Sec. 3.4.1). With regard to the Fast-YCB dataset, which also provides the ground-truth velocity of the object for all the frames, we also considered the linear and angular velocity RMSE errors (see Sec. 3.4.1).

We evaluated the above metrics by considering all the frames of each considered sequence.

5.5.5 Results on Fast-YCB

Performance of 6D object pose tracking

In Table 5.2, ROFT is compared against the state of the art in 6D object pose tracking from RGB-D images. The comparison is extended to the estimation network DOPE (also used as pose measurement T_t^d in ROFT) running at 5 fps as baseline. To this end, the ground-truth signal at each t is compared with the last available output from the network.

Considering all the objects on average, ROFT achieves the best performance according to all the metrics and gets an average Cartesian error in the order of few centimeters and an average angular error of approximately 13 degrees. Remarkably, it outperforms the other methods in terms of angular error both on average and for each

Table 5.3 Results on the Fast-YCB dataset: RMSE linear and angular velocity errors.

metric	Linear velocity RMSE (cm/s))		Angular veocity RMSE (deg/s)	
method	ROFT	ROFT w/o segm. sync.	ROFT	ROFT w/o segm. sync.
003	**5.237**	8.469	**18.250**	38.163
004	**8.500**	15.996	**24.949**	65.321
005	**9.592**	21.033	**41.212**	78.006
006	**7.227**	16.409	**26.553**	65.494
009	**22.277**	32.428	**46.374**	94.324
010	**6.388**	23.194	**26.083**	67.406
ALL	**11.409**	20.931	**32.119**	70.164

object separately. The angular error is reduced on average by approximately 71%, 63% and 60% with respect to PoseRBPF, se(3)-TrackNet and DOPE, respectively.

We remark that the experiments with se(3)-TrackNet involve re-initialization from the DOPE predictions once they are available (at 5 fps). Without re-initialization, we found that se(3)-TrackNet would lose the track of the object pose in almost all the experiments leading to a ADD-AUC percentage of 50.55% (i.e. 21.5 points less than the performance achievable with re-initialization).

Performance of 6D object velocity tracking

In Table 5.3, the RMSE errors for the linear and angular velocities $\mathcal{V}_t$ are reported. We compare the performance in the case in which the optical flow-aided segmentation mechanism, presented in Sec. 5.2, is enabled or disabled.

In terms of absolute errors, ROFT gets approximately 11 cm/s and 32 deg/s errors on average. These results are judged as fairly accurate considering that the objects motion reaches velocities up to 63 cm/s and 266 deg/s. With respect to the scenario in which the non-synchronized masks are used (indicated as “w/o segm. sync.” in the table), the masks refined with the optical flow help reducing the tracking error considerably (on average, 45% and 54% less error for the linear and angular parts, respectively).

5.5.6 Results on HO-3D

In Table 5.4 ROFT is compared against the baseline (DOPE) and the state of the art on the HO-3D dataset. Given the considerable amount of occlusions due to the interaction between objects and the human hand, this dataset poses more challenges to deep neural networks for object segmentation, pose estimation and tracking. As a

Table 5.4 Results on the HO-3D dataset: ADD-AUC and RMSE positional and angular errors for several methods.

metric	ADD-AUC, $\epsilon = 10$ cm				Positional RMSE (cm)				Rotational RMSE (deg)			
method	DOPE	ROFT	Pose RBPF	se(3)-TrackNet	DOPE	ROFT	Pose RBPF	se(3)-TrackNet	DOPE	ROFT	Pose RBPF	se(3)-TrackNet
003	48.81	**70.52**	54.21	52.07	9.8	**3.3**	3.8	6.8	45.918	**23.085**	67.665	42.743
004	60.28	**73.80**	60.87	67.20	10.5	3.1	**2.8**	8.9	47.169	**27.243**	71.168	45.790
006	36.32	56.29	**59.09**	51.89	17.0	6.6	**2.9**	16.3	78.987	**40.107**	86.246	76.446
010	43.27	46.23	**53.87**	53.67	13.1	9.0	**3.2**	14.1	57.547	**41.618**	107.586	54.454
ALL	49.34	**63.31**	57.82	58.36	12.7	5.7	**3.1**	11.9	57.761	**33.437**	83.318	55.462

Table 5.5 ADD-AUC and RMSE errors averaged on all the selected HO-3D sequences when ground-truth inputs are used.

variant	ADD-AUC (%)	RMSE e_t (cm)	RMSE e_a (deg)
ROFT	63.31	5.7	33.437
ROFT w/ GT segm.	62.54	5.5	34.693
ROFT w/ GT pose	82.00	2.3	15.820
ROFT w/ GT segm. and pose	81.86	2.1	16.238

result, the quality of the PoseRBPF and se(3)-TrackNet predictions and of the inputs to ROFT, i.e. masks from Mask R-CNN and poses from DOPE, reduces considerably.

According to the ADD-AUC metric, even the best performance, reached by ROFT, stops at approximately 64%. The best performance in terms of RMSE Cartesian error is instead obtained by PoseRBPF. On the other hand, ROFT still achieves the best performance in terms of RMSE angular error for each object, with average reductions in the error by approximately 60%, 44% and 40% compared to PoseRBPF, DOPE and se(3)-TrackNet respectively.

Remarkably, as shown in Table 5.5, the performance of ROFT improves by increasing the quality of the inputs. E.g., using ground-truth poses leads to an increase of more than 18 points of the ADD-AUC metric and a reduction of both the RMSE errors to less than half the original error.

5.5.7 Results: frame rate comparison

ROFT achieves 96.2 fps compared to 142.4 fps obtained by se(3)-TrackNet and 13.5 fps achieved by PoseRBPF (with 50 particles). The frame rate has been evaluated as the inverse of the mean iteration time averaged on all the sequences of both Fast-YCB and HO-3D datasets. All the experiments have been executed on the same hardware, i.e. a mobile computer equipped with an Intel Core i9-10980HK CPU and an NVIDIA RTX 2080S GPU. Our method and se(3)-TrackNet exceed the typical real-time requirement of 30 fps.

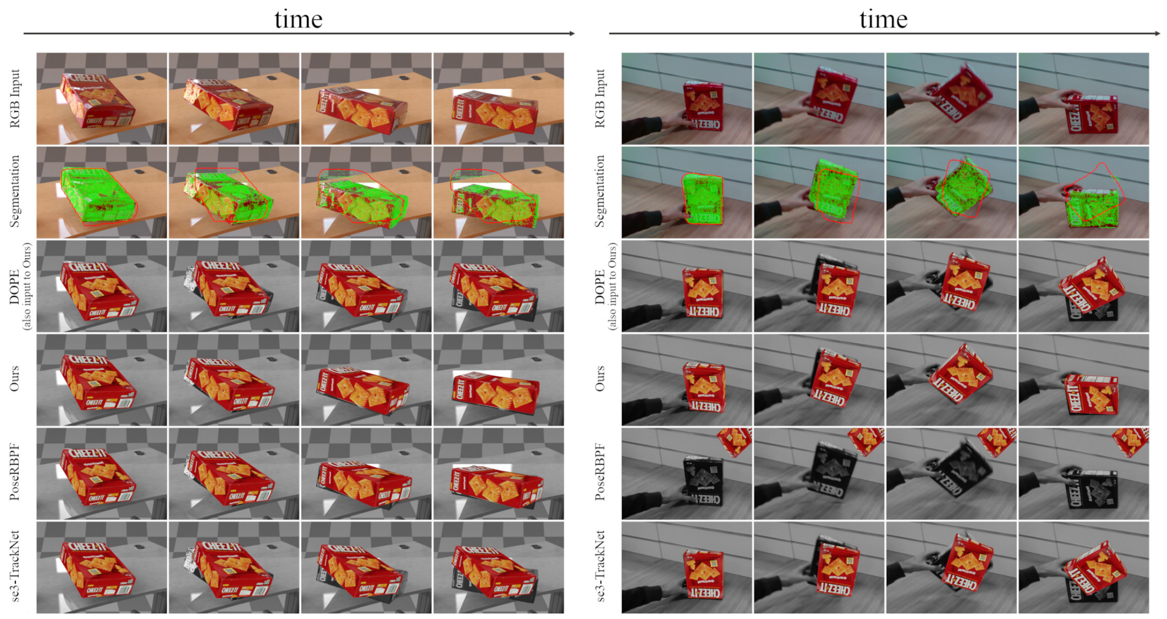


Figure 5.4 Qualitative results for the object cracker box (003). On the left: frames from Fast-YCB. On the right: frames acquired with a RealSense D415 camera. The time length of the object motion is approximately 1 second. The red contour in the Segmentation row represents the stream of delayed masks M_t^d, the green dots represent the masks M_t propagated using the optical flow.

5.5.8 Qualitative results

In this section I compare qualitatively the proposed tracking method, ROFT, with the network DOPE and the tracking methods PoseRBPF and se(3)-TrackNet on a synthetic sequence extracted from the Fast-YCB dataset and on a real sequence acquired in a similar scenario with a RealSense RGB-D camera.

In Fig. 5.4, the estimates are represented by superimposing the textured mesh of the object cracker box (003), transformed according to the estimated pose, on the input frames. ROFT provides well-synchronized estimates in case of fast motions from both the Fast-YCB dataset and the real sequences. The estimates from DOPE (used as pose measurements in ROFT) and se(3)-TrackNet show lack of synchronization especially with regard to the orientation of the object. The performance of PoseRBPF is similar to that of ROFT on the synthetic sequence, but it looses track on the real sequence.

These results can be largely attributed to the usage of the optical flow via the object velocity $\mathcal{V}$. The velocity conveys the 6D direction of motion to the Unscented Kalman filter that is valuable in presence of fast motions. Methods based on particle filtering, as PoseRBPF, usually do not employ this kind of information and explore the

Table 5.6 Performance of ROFT on the Fast-YCB dataset when key components are selectively disabled.

metric	ADD-AUC	Positional RMSE (cm)	Rotational RMSE (deg)
ROFT w/ GT segm. and pose	84.51	2.5	10.591
ROFT w/ GT pose	83.97	2.6	11.507
ROFT w/ GT segm.	77.08	3.1	11.901
ROFT	**76.59**	3.2	**12.675**
ROFT w/o pose sync.	74.83	**3.0**	17.060
ROFT w/o outlier rej.	74.79	10.8	14.248
ROFT w/o segm. sync.	64.69	5.4	27.497
ROFT w/o velocity	54.23	19.8	36.492
ROFT w/o pose	13.27	15.9	80.012
DOPE (ideal)	84.94	8.7	16.277
DOPE	58.83	15.1	31.491

state space via random walks. Conversely, end-to-end approaches for object tracking, as se(3)-TrackNet, predict the relative 6D pose between consecutive time steps directly from images. This task is rather complex to be accomplished and might result in underestimation of the actual motion or tracking loss. ROFT, instead, uses optical flow which solves the simpler task of estimating the motion of pixels in 2D space. This information is later converted to the 6D domain using a filtering based approach that does not require training.

In the second row of the same figure the stream of non-synchronized masks M_t^d from Mask R-CNN (red silhouette) is shown and compared with the output of the optical flow-aided mask synchronization mechanism as described in 5.2 (green dots). We notice that, although the obtained masks are discontinuous, they provide a reasonable source of pixel coordinates belonging to the object as required by the 6D velocity tracking stage described in Sec. 5.3.

5.5.9 Ablation studies

In this section I will provide the results of several ablation studies whose aim is to assess the effectiveness of the many components that constitute the ROFT pipeline. Furthermore, I will provide the outcome of additional experiments where the sources of segmentation masks and pose estimates are replaced with ground-truth segmentation and/or pose measurements.

The results are provided in the Table 5.6. Here, "w/ GT segm" indicates using the ground-truth segmentation, "w/ GT pose" indicates using the ground-truth pose, "w/o pose sync." indicates disabling the synchronization of pose measurements as per Sec. 5.4.5, "w/o outlier rej." indicates disabling the outlier rejection mechanism as per Sec. 5.4.6, "w/o velocity" indicates that the velocity measurements are discarded in Eq.

(5.18) and "w/o pose" indicates that the pose measurements are discarded in the same equation. The table also include a comparison with the delayed DOPE predictions at 5 fps, indicated as "DOPE", and DOPE predictions on all the frames at 30 fps without delay, indicated as "DOPE (ideal)".

Excluding DOPE (ideal) and experiments with ground-truth inputs, the full ROFT pipeline reaches the best performance according to the ADD-AUC (76.59%) and RMSE angular error metrics (12.675 deg). While the pose synchronization mechanism increases the RMSE Cartesian error by only 0.2 cm, it reduces the average angular error by approximately 5 degrees. On the other hand, the outlier rejection mechanism mostly affects the Cartesian error, reducing it by approximately 7 cm. Remarkably, ROFT achieves better RMSE errors even when compared with the DOPE (ideal) scenario. This phenomenon can be explained by the presence of the outlier rejection mechanism.

If velocity measurements $\mathcal{V}_t$ are not used, the performance degrades considerably because the filtering process performs corrections based on a new pose every N_p steps only. Also, using DOPE alone at 5 fps would give better results in this scenario. This result confirms the importance of using the optical flow via the object velocity $\mathcal{V}_t$. Conversely, when the velocities are used without pose measurements T_t^d, the performance of ROFT is the lowest due to the noise in the velocities, which gets integrated over time.

Overall, these results show that each component of the pipeline contributes to the overall performance. Remarkably, only a combination of pose and velocity measurements justifies the necessity to use a Kalman filtering approach for information fusion.

Experiments with ground-truth inputs show that the performance of ROFT can be further increased. This suggests that pipelines with a certain degree of modularity, as the one proposed, where each component plays a clear and explainable role, can be more conveniently improved by possibly replacing some of their components with better ones. It is recalled that a similar modularity is not achievable with end-to-end approaches for tracking such as se(3)-TrackNet.

5.5.10 Tracking and grasping experiments on a humanoid robot

In order to assess the effectiveness of the proposed tracker for robotics applications, I developed an online application with the iCub humanoid robotic platform (Metta et al., 2010). The gaze control system of the robot (Roncone et al., 2016) is used to track the object of interest in the image plane by using the estimated position $p_t \in \mathbb{R}^3$ as a reference gazing point (see Fig. 5.5).

We remark that, in this scenario, both the camera, mounted on the head of the robot, and the object move. Ego-motion is implicitly taken into account by the fact that the algorithm estimates the relative motion of the object with respect to the camera. Our experiments confirm this observation, demonstrating that ROFT retains good performance in presence of camera motions.

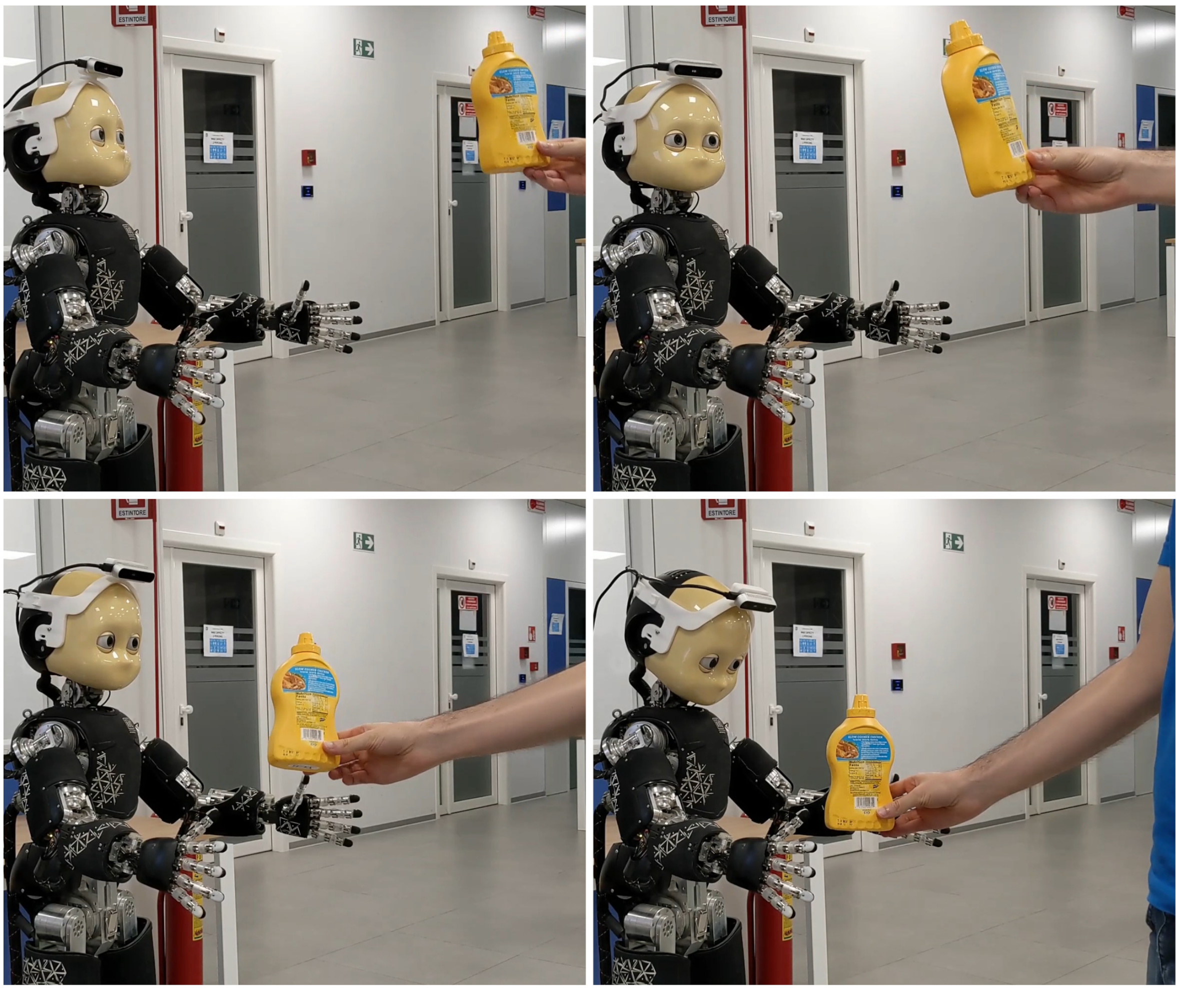


Figure 5.5 The iCub humanoid robot while it gaze-tracks the mustard bottle from the YCB Model Set using the output of ROFT as feedback.

The proposed experiment also aims at evaluating whether the tracked poses are accurate enough to enable robot grasping. To this end, we designed a simple grasping application. When the object is within the reachable space of the robot and the estimated velocity is almost zero, the pose tracked by ROFT is used to plan a grasping action. Specifically, we consider the 3D model of the object at the estimated pose and we find possible grasping candidates using the planning approach proposed in (Nguyen et al., 2018). The best grasping candidate is finally executed using the Cartesian controller (Pattacini et al., 2010) of the robot. Sample grasping sequences are shown in Figs. 5.6 and 5.7.

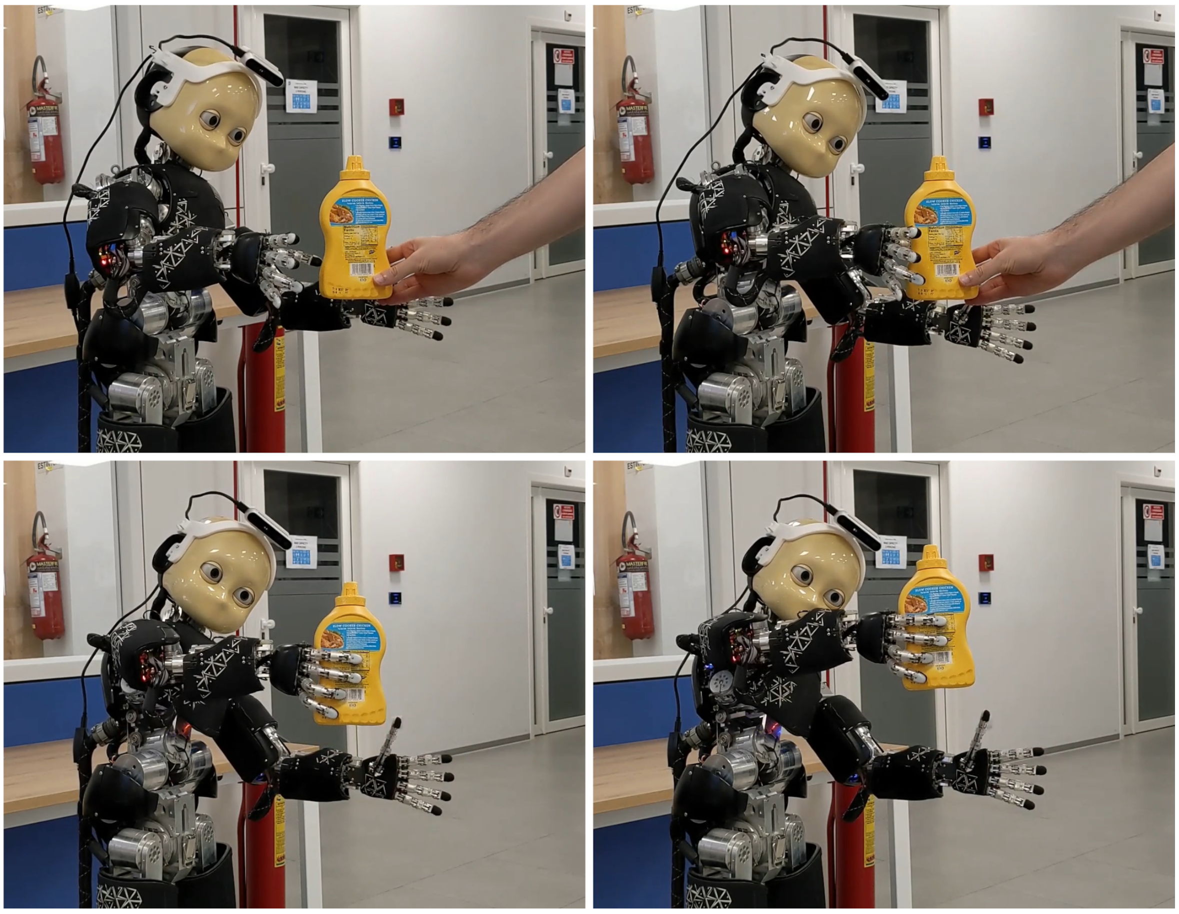


Figure 5.6 The iCub humanoid robot while it grasps the mustard bottle from the YCB Model Set. The output pose from ROFT is used to plan the grasping action.

Remarkably, by tracking the pose of the object over time, we were also able to implement a simple adaptive strategy which interrupts the grasping action if the object moves away from the planned grasp pose.

5.5.11 Limitations

This section ends the experimental evaluation by discussing some limitations of the method.

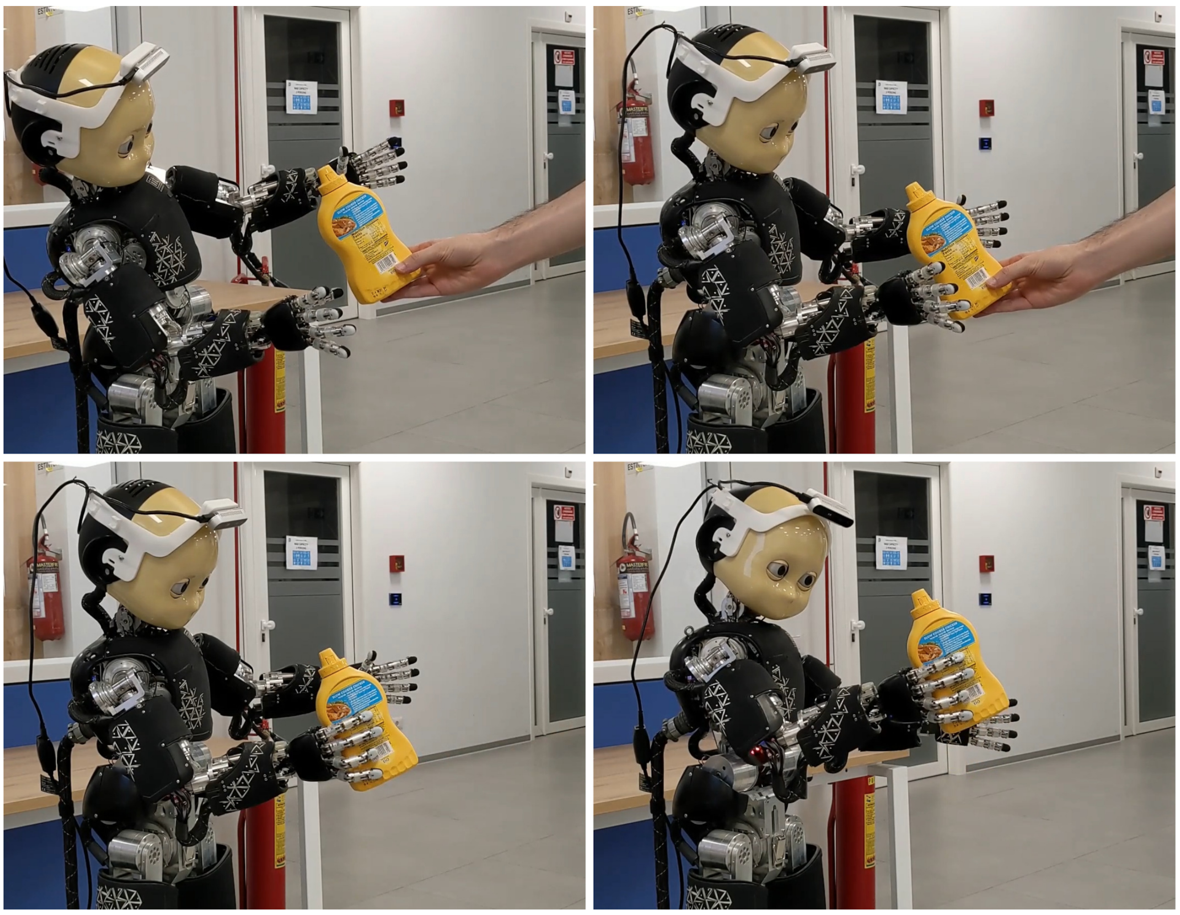
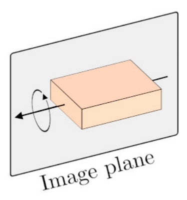


Figure 5.7 Another example of iCub humanoid robot grasping the mustard bottle from the YCB Model Set. The output pose from ROFT is used to plan the grasping action.

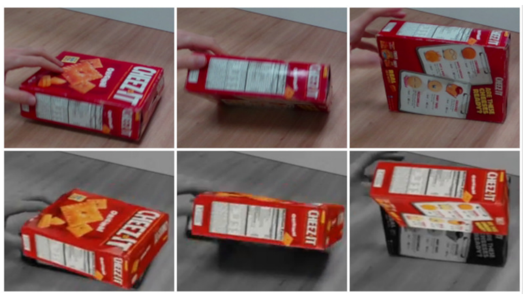
Tracking in presence of ego-motion

The proposed tracking framework has been designed to track the relative pose and the relative velocity between the camera reference frame $\{I\}$ and the reference frame $\{B\}$ attached to the object of interest (see Sec. 3.1.1). This choice implies that it can also be used in the presence of a moving camera, as the robotic experiments in Sec. 5.5.10 demonstrates qualitatively.

Nonetheless, the numerical experiments have been conducted on the Fast-YCB and HO-3D datasets, which have been acquired using a stationary camera and do not include camera ego-motion. I plan to further investigate quantitatively the effect of ego-motion on the proposed pipeline in future research.



(a)



(b)

Figure 5.8 (a) Example of out-of-plane object rotation. (b) Top: input images. Bottom: example of imprecise estimates in presence of out-of-plane object rotations.

Tracking out-of-plane object rotations

The experiments revealed that out-of-plane object rotations (see Fig. 5.8a) can sometimes result in the underestimation of the optical flow motion vectors F_t. The difficulty to track the pixels motion comes from object self-occlusions and causes less precise tracking of the angular velocity $\omega_t(\mathcal{V}_t)$ and, therefore, of the object pose. An example of the phenomenon is shown in Fig. 5.8b. Nevertheless, we observed that this condition mainly affects the magnitude of the angular velocity rather than its direction, as the final orientation of the object in Fig.5.8b shows.

It can be expected that an online adaptation of the covariance matrices R_* in Eq. (5.18) could help improve the tracking performance by giving more importance to the velocity measurements when required. I plan to investigate the design and the validation of such adaptation schemes in future research.

Chapter 6

Differentiable Filtering for 1D Object Sliding Tracking

In the previous chapters, I considered the problem of tracking the 6D pose and the velocity of a given object from RGB-D images. However, there are scenarios in which a robot might not be able to effectively use the visual sensing modality for several reasons, e.g., due to unusual lighting conditions or because the object is mostly occluded by other objects or by the end-effector of the robot during in-hand object interactions. For these reasons, it is also important to develop object pose estimation and tracking algorithms that consider other sensing modalities, such as tactile sensing.

State-of-the-art methods addressing specifically the problem of *in-hand* object tracking either use visual information and concentrates on achieving robustness to occlusions (Wen et al., 2020b) or focus on providing rich and efficient tactile contact modelling (Liang et al., 2020) that can help explaining complex within-hand object motions in real-time.

Among the most typical in-hand object motions, object slipping and sliding are particularly challenging to be perceived and controlled. For this reason, they have been extensively explored and studied within the literature on tactile-based perception and control. In this respect, several works (Dong et al., 2019; Meier et al., 2016; Veiga et al., 2018) propose methods, often learning-based, for slip detection and prediction and how to utilize them for grip stabilization. On the other hand, most works dealing with in-hand object pose estimation and tracking do not address the problem, as in Wen et al. (2020b). An exception is Liang et al. (2020), where a physical engine is used to model the tactile interaction with the object and is integrated in an object pose tracker that explicitly takes into account slippage. More recently, Costanzo (2021) proposed a nonlinear observer which combines a model of the Coulomb and viscous friction with tactile measurements in order to estimate the relative orientation between the object and the fingers of a parallel jaw gripper. They show how to use feedback from the observer in order to reject the possible rotational slippage of the object or regulate the orientation of the object to a desired reference.

During the PhD, I explored the problem of tracking the position and the velocity of an object subjected to in-hand sliding motion using tactile observations (Piga et al., 2021b). The tracking algorithm is implemented as a differentiable Kalman filter (Kloss et al., 2021). This paradigm enables tracking the state of systems that might not be easily modeled analytically, as in the case of in-hand object manipulation involving tactile sensing. This is made possible by learning the measurement model of the sensors from ground-truth data obtained with visual feedback.

The major contributions presented in this chapter are the following:

- The modelling of tactile sensing modalities in the context of differentiable Kalman filtering for state tracking without the need to manually write mathematically challenging motion and measurement models;
- The development of strategies for collecting labelled data using Kalman smoothers starting from noisy ground-truth data;
- The execution of experiments on a real humanoid anthropomorphic hand equipped with soft tactile sensors.

We remark that Liang et al. (2020) provide experiments both in simulation and using real-world data. However, the results regarding the slippage are not provided for the real-world scenario and analyzed only in simulation. Conversely, this work also provides results on sliding tracking using real-world tactile data. As regards the method proposed in (Costanzo, 2021), we observe that it focuses on tracking relative rotational motions between the object and the end-effector in presence of rotational slippage. Conversely, this work concentrates on translational sliding motions. Furthermore, (Costanzo, 2021) employs a model-based approach, while this work addresses the task at hand using an hybrid approach.

A video of the experiments is available online[1]. The code used for the experiments is made publicly available for free with an Open Source license online[2].

[1]https://www.frontiersin.org/articles/10.3389/frobt.2021.686447/full#supplementary-material
[2]https://github.com/hsp-iit/dekf-tactile-filtering

The remainder of this chapter is organized as follows. In Sec. 6.1, I present the proposed method in detail with the help of the theory from Chapters 3 and 4. Then, Sec. 6.2 explains the working principle of the adopted tactile sensors and describes the procedures for data collection and training of the differentiable filter. Finally, in Sec. 6.3, I discuss the experimental results and possible limitations of the method.

6.1 Description of the method

The proposed method tracks the *one-dimensional* position $p_t \in \mathbb{R}$ and the associated velocity $v_t \in \mathbb{R}$ of an object of interest $\mathcal{O}$ while it *slides* between the fingertips of a multifingered robotic hand using noisy tactile data τ. A differentiable Extended Kalman filter (see Sec. 2.5) is used to accomplish the task.

We assume that the tactile data is represented using an Euclidean vector $\tau \in \mathbb{R}^{ML}$ where M indicates the number of sensorized fingers and L indicates the number of channels associated to each sensor. In the literature, there exist several implementations of soft tactile sensors, e.g. (Tomo et al., 2016; Wettels et al., 2014), which deform in the 3D space during the interaction with the object. For these sensors, the number of channels is usually in the order of 3 or more, i.e. $L \geq 3$.

Differently from the other chapters, here we will assume that the inertial frame $\{I\}$, introduced in Sec. 3.1.1, is not attached to the camera, as this work deals only with tactile data. Conversely, the frame will be attached directly to the root frame of the robot that manipulates the object. Consequently, p_t will represent the position of the object with respect to the origin of the robot frame and expressed in the robot frame.

6.1.1 Assumptions on the considered manipulation task

The developments presented in this section assume that the object of interest slides between the finger of the robot hand. More specifically, we assume that

- the object can only slide along a single translational direction;
- the object does not rotate during the sliding motion;
- the object is only subjected to the gravity force, i.e. no external disturbances act on it;
- the motion of the object is caused by a periodic reduction of the grasp force resulting in a sequence of stick-slip-like motions.

We remark that the above assumptions do not include all the possible scenarios in which an object might start sliding. As an example, an increasing shear load that eventually reaches and exceeds the maximum static friction force. Moreover, the assumption that the sliding motion occurs along a single direction might not be always verified when using a general multi-fingered hand. Nonetheless, these scenarios have not been considered in this project and the experiments have been conducted under controlled conditions in order to fulfill the aforementioned assumptions.

In the remainder of this section, all the components of the tracking algorithm are described.

6.1.2 Definition of the state and the measurements

The state to be tracked x_t is defined as

$$x_t = \begin{bmatrix} p_t \\ v_t \end{bmatrix} \in \mathbb{R}^2, \tag{6.1}$$

where p_t and v_t are the position and the velocity of the object. Without loss of generality, we let $p_0 = 0$.

In order to track the state x_t, the tactile data $\tau \in \mathbb{R}^{ML}$ is used as a measurement. However, as we will prove in the experimental evaluation, using the derivative of the tactile signal $\dot{\tau}_t$, instead of the plain signal τ_t, is crucial in order to obtain a meaningful estimation of the state under the hypotheses stated in 6.1.1. For this reason, we will assume that the measurements take the form

$$z_t = \dot{\tau}_t \in \mathbb{R}^{ML}. \tag{6.2}$$

6.1.3 Motion model

Motion and measurement models are usually *first principles models*, i.e. they are obtained form the combination of suitable physical laws that describe the system of interest. This way of proceeding was indeed considered in Chapters 5 and 6. However, writing the mathematical expression of these models is not always possible if the system under investigation is considerably complex or the working principles are not completely known. This is the case for the majority of the tactile sensors adopted in the literature. For this reason, the differentiable Kalman filtering paradigm, presented in Sec. 2.5, has been adopted.

The differentiable Kalman filtering paradigm allows injecting neural network modules within the standard motion and measurement model in Eqs. (2.7) and (2.8) such that the models, or part of them, can be learned from experimental data.

In the specific case of this project, we adopted an hybrid motion model. It combines the physical notions of position and velocity with a neural network. The model is as follows:

$$\begin{aligned} x_t &= f_{\theta_1}(x_{t-1}, w_{t-1}), \\ &= \begin{pmatrix} p_{t-1} + v_{t-1}\Delta_t \\ v_{t-1} + NN_{\theta_1}(x_{t-1}) \end{pmatrix} + w, \\ w &\sim \mathcal{N}(0, Q). \end{aligned} \tag{6.3}$$

The positional part p_t is updated using a constant velocity model (Bar-Shalom et al., 2002) where the elapsed time Δ_t is assumed equal to the sampling period of the algorithm. Instead, the change in velocity between consecutive instant of times $v_t - v_{t-1}$ is modelled as the output of a neural network NN_{θ_1} with parameters θ_1 given the state at the previous instant of time x_{t-1}:

$$v_t - v_{t-1} = NN_{\theta_1}(x_{t-1}) \in \mathbb{R}. \tag{6.4}$$

The process noise covariance matrix $Q \in \mathbb{R}^{2\times 2}$ is also considered as a parameter to be learned.

In conclusion, we know that the position is the integral of the velocity, thus we combine this information within the model without the need to learn it from data. Conversely, the change in velocity could be modelled by reasoning on the forces acting on the object. However, given that extracting the forces from non-calibrated tactile sensors might be challenging, then deriving a model for the velocity from first principles might be difficult. For these reasons, we found convenient to describe the model using a neural network.

6.1.4 Measurement model

Given the complexity of the measurement process, depending on the specific nature of the tactile sensors, we propose to design the measurement function $h(x)$ using a neural network NN_{θ_2} with parameters θ_2. The resulting measurement model is:

$$\begin{aligned} z_t &= h_{\theta_2}(x_t, \nu_t) \\ &= NN_{\theta_2}(x_t) + \nu, \\ \nu &\sim \mathcal{N}(0, R), \end{aligned} \tag{6.5}$$

where $NN_{\theta_2}(x_t) \in \mathbb{R}$ maps the state x_t to a one-dimensional feature. In order to make the learned feature comparable with the actual measurement $\dot{\tau} \in \mathbb{R}^{ML}$, we employ a secondary network NN_{θ_3} with parameters θ_3 such that the actual measurement become

$$z_t = NN_{\theta_3}(\dot{\tau}_t) \in \mathbb{R}. \tag{6.6}$$

In summary, the adopted measurement model uses two neural networks in order to i) map the actual measurement $\dot{\tau}$ to a 1-dimensional measured feature $NN_{\theta_3}(\dot{\tau}_t)$ and ii) map the state x_t to a 1-dimensional feature $NN_{\theta_2}(x_t)$. The two networks are not trained separately as the target feature and its characteristics are not known a priori. Instead, they are trained jointly such that the best intermediate feature for the tracking task can be learned from the data.

We adopted a one-dimensional feature in lieu of a multi-dimensional one as we assume that the sliding motion can occur only along a single translational direction. The position associated to the motion is easily recovered by integrating the velocity once $p_0 = 0$ is fixed, as Eq (6.3) models. Hence, the filter should be mainly responsible for correcting the velocity part of the state $v_t \in \mathbb{R}$, for which a one-dimensional measurement should be sufficient.

Similarly to the motion model, the measurement noise covariance matrix $R \in \mathbb{R}$ is considered as a parameter to be learned.

6.1.5 Choice of the filtering algorithm

The proposed motion and measurement models both contain neural network modules that make the motion and measurement functions f and h nonlinear. Although the Unscented Kalman filter could be used to track the state, one of its main advantages (i.e. no need for evaluating the Jacobians) no longer holds. Indeed, almost every machine learning software framework allows evaluating the Jacobians analytically using the so-called “automatic differentiation”. For this reason, we adopted an Extended Kalman filter in order to accomplish the task.

6.1.6 Specification of the neural networks

In this section, I will provide a concise description of the inner architecture of the networks NN_{θ_1}, NN_{θ_2} and NN_{θ_3}. Furthermore, I will explain how to handle the trainable noise covariance matrices Q and R.

Following prior work (Lee et al., 2020), the network NN_{θ_1} in Equation (6.3) is designed such that the input x_{t-1} is first fed through a multi-layer encoder with the following structure:

- 1 linear layer with 64 units;
- 1 ReLU activation;
- 1 linear residual layer.

The extracted feature is then passed to a shared stage composed by:

- 1 linear layer with 64 units;
- 3 linear residual layers;
- 1 linear layer with 64 units with output size equal to 1.

The above mentioned linear residual layer is composed as follows:

- 1 linear layer with 64 units;
- 1 ReLU activation;
- 1 linear layer with 64 units;
- 1 summation junction with the input to the residual layer;
- 1 ReLU activation.

We mention that the consecutive linear layers in the shared stage cannot be unified in a single linear layer because the residual linear layer contains nonlinear activation functions in the output. We also notice that the output of the last layer of the shared stage has size equal to 1 since it needs to be summed up to the previous velocity v_{t-1} as per Equation (6.3).

The reason for using a shared stage is that, in case an input u_t to the system is available, a secondary multi-layer encoder can be used to extract features from u_t (Lee

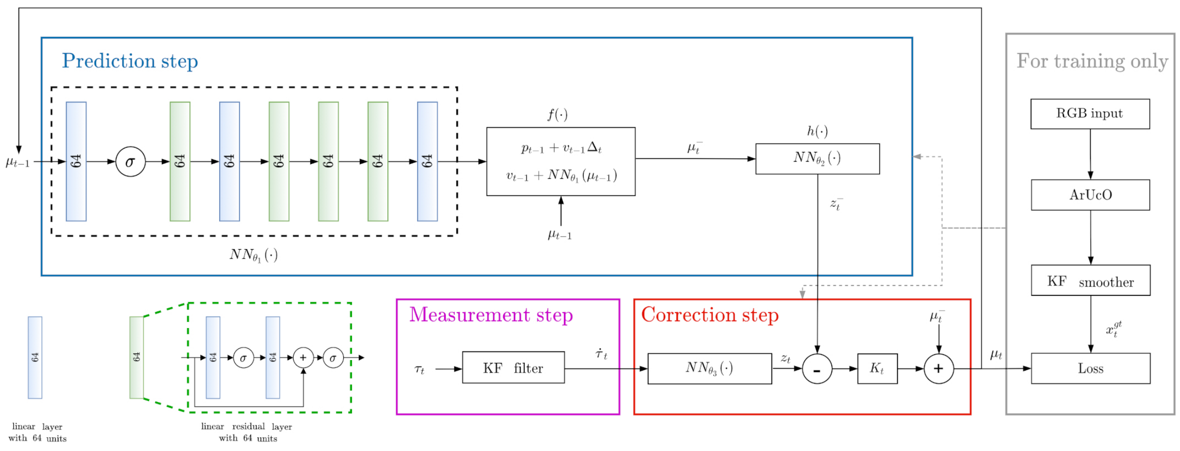


Figure 6.1 Overview of the proposed differentiable filtering architecture for object sliding tracking. The letter σ indicates the ReLU activation function. The linear layers of the network are represented as blue blocks. Instead, the linear *residual* layers are represented as green blocks.

et al., 2020). Then, the features extracted from x_{t-1} and u_t could be concatenated and then passed to the shared stage (Lee et al., 2020). Given that in this work there are no available inputs, the state features are directly fed to the shared stage.

The same structure as above has been used for the networks NN_{θ_2} and NN_{θ_3}. Also for these networks, the output size of the last layer equals to 1 as it is the size chosen for the measurement feature in Eq. (6.5).

The structure of the network $NN_{\theta_!}$ is summarized in Fig. 6.1 together with the interconnections with the other networks NN_{θ_2} and NN_{θ_3} throughout the filtering architecture. The inner structure of the linear residual layer is also represented in the bottom left part of the same figure.

Handling of the noise covariance matrices

Following prior work (Lee et al., 2020), the training procedure does not directly learn the matrices Q and R. Instead, their Cholesky decomposition $L_Q \in \mathbb{R}^{2\times 2}$ and $L_R \in \mathbb{R}^1$ is considered. In order to take into account this choice, the actual implementation of the Extended Kalman prediction and correction steps evaluates the covariance matrix Q as $L_Q L_Q^T$ and the covariance matrix R as $L_R L_R^T$, as per the definition of the Cholesky decomposition.

dEKF for Object Sliding Tracking

```
1: procedure dEKF($\mu_0, P_0, P_{\tau,0}, \theta_1, \theta_2, \theta_3$)
2:     $\mu_{t=0} = \mu_0$                          ▷ Initialization of the object sliding tracker
3:     $P_{t=0} = P_0$
4:     $\tau_{t=0} = \tau_0, \quad \dot{\tau}_{t=0} = 0$          ▷ Initialization of the filter of the tactile signal
5:     $P_{\tau,t=0} = P_{\tau,0}$
6:     for each ($\tau_t$) do
Filtering of the derivative of the tactile signal
7:         Definition of the functions $f$ and $h$:
               $\begin{bmatrix}\tau_t\\ \dot{\tau}_t\end{bmatrix} = f\left(\begin{bmatrix}\tau_{t-1}\\ \dot{\tau}_{t-1}\end{bmatrix}\right) = \begin{bmatrix}\tau_{t-1} + \dot{\tau}_{t-1}\Delta_t\\ \dot{\tau}_{t-1}\end{bmatrix},$
               $z_t = h\left(\begin{bmatrix}\tau_t\\ \dot{\tau}_t\end{bmatrix}\right) = \tau_t.$
8:         $z_t \leftarrow \tau_t$
9:         $(\tau_t, \dot{\tau}_t, P_{\tau,t}) = \mathrm{KF}(f(\cdot), h(\cdot), \tau_{t-1}, \dot{\tau}_{t-1}, P_{\tau,t-1}, z_t)$     ▷ See Algorithm 1
Object sliding tracking
10:        $z_t \leftarrow NN_{\theta_3}(\dot{\tau}_t)$
11:        $(\mu_t, P_t) = \mathrm{EKF}(f_{\theta_1}(\cdot), h_{\theta_2}(\cdot), \mu_{t-1}, P_{t-1}, z_t)$     ▷ See Algorithm 3
12:                                                   ▷ See 6.1.3 for $f_{\theta_1}(\cdot)$
13:                                     ▷ See 6.1.4 for $h_{\theta_2}(\cdot)$ and $NN_{\theta_3}(\cdot)$
14:    end for
15: end procedure
```

6.1.7 Resulting algorithm

The proposed algorithm for object sliding tracking is summarized in the table entitled "dEKF for Object Sliding Tracking". The overall framework is also depicted in Fig. 6.1.

As specified in Sec. 6.1.2, the first derivative of the tactile signal $\dot{\tau}_t$ is required to actually implement the filtering architecture. Given that tactile sensors usually do not provide the derivative of their own output, a linear Kalman filter is used to estimate the derivative $\dot{\tau}_t$. Further details can be found in the above mentioned table.

6.2 Data collection and training

The physical setup used for data collection and training comprises the iCub humanoid platform (Metta et al., 2010) and a special version of the uSkin soft tactile sensors (Tomo et al., 2016) that has been adapted (Holgado et al., 2019) for the iCub robot.

The remainder of this section is organized as follows. After a brief description of the sensors in Sec. 6.2.1 and of the iCub humanoid platform in Sec. 6.2.2, Sec. 6.2.3 explains how the training and testing data were collected via experiments of controlled sliding of several objects and post-processed. Finally, in Sec. 6.2.4, the adopted training protocol is briefly discussed.

6.2.1 Description of the tactile sensors

The adopted tactile sensors are *magnetic* soft tactile sensors. The sensing principle is the following: when external forces deform the sensor, due to the interaction with the object, the deformation is sensed by an electronic circuit such that the output signal is correlated with the interaction between the sensor and the object. In the case of magnetic sensors, a magnet is embedded on the surface of the sensor, and a Hall effect sensor detects changes in the magnetic field.

The specific sensor that has been used is the uSkin sensor (Tomo et al., 2016) and in particular a special version of the uSkin sensor that has been adapted (Holgado et al., 2019) for the anthropomorphic hands of the iCub robot.

The main sensing module of size $6 \times 6 \times 3.8$ mm, shown in Fig. 6.2-A, consists of (from the bottom):

- a PCB board hosting the Hall effect sensor chip;
- a single soft silicone skin cover;
- a neodymium magnet;
- a flexible textile cover (realized with a high friction grip tape).

Integration of the above sensing module on the fingertips of the iCub hands is achieved with a suitable fingertip adapter (Holgado et al., 2019) show in Fig. 6.2-B. The adapter hosts two sensing modules that follow the original curvature of the fingertip assembly. A top layer of grip tape is finally placed over the whole assembly in order to provide good friction properties and protect the underlying sensors. The final assembly is shown in Fig. 6.2-C.

Once mounted on the fingertips of the iCub robot, the sensors allows measuring the interaction between the fingertips and the object. Specifically, the output of the sensor consists of 3 channels, i.e. $L = 3$, that are proportional to the sensed magnetic

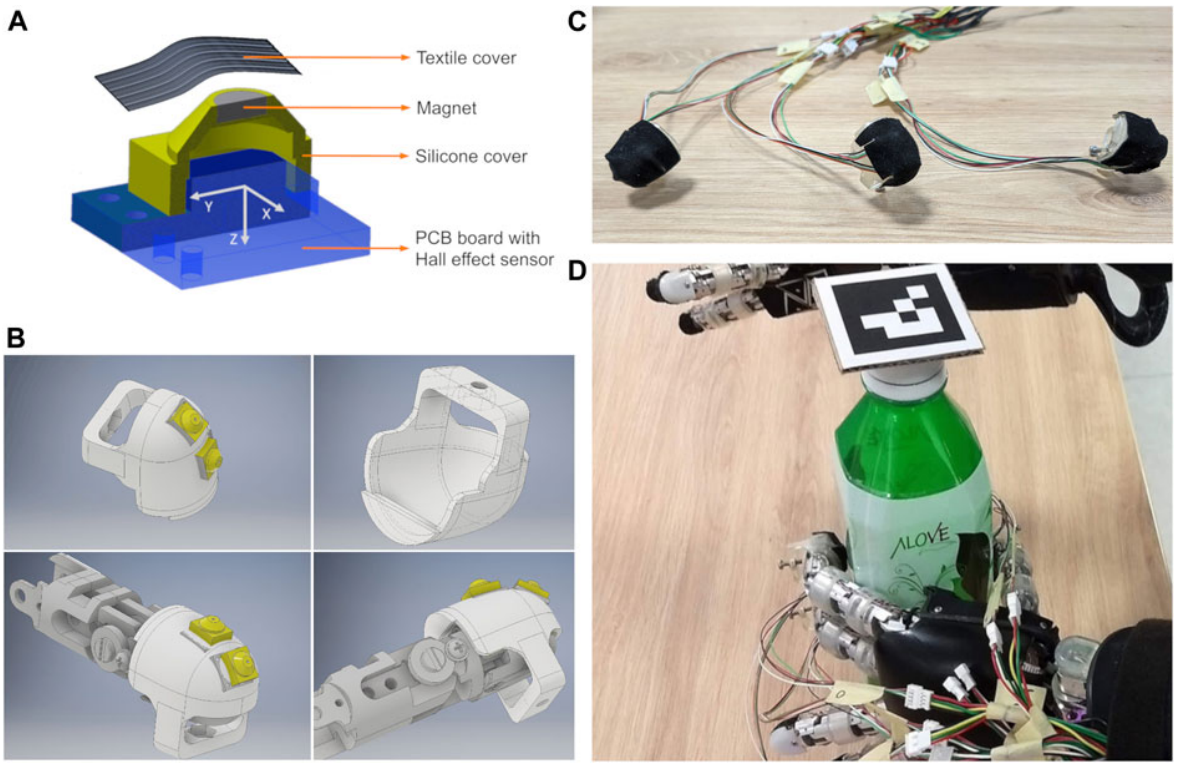


Figure 6.2 In (A), the exploded view of the sensing module. In (B), the CAD model of the fingertip adapter for the iCub finger hosting two sensing modules. In (C), the final assembled fingertip adapters with the grip tape cover. In (D), the iCub humanoid robot left hand equipped with uSkin deformable soft tactile sensors.

field. The resulting vector of 3 coordinates, for each finger f, is indicated as

$$\tau^f = \begin{bmatrix} \tau_x^f \\ \tau_y^f \\ \tau_z^f \end{bmatrix} \in \mathbb{R}^3, \tag{6.7}$$

where the x, y and z axes are represented in Fig. 6.2-A. The 3D nature of the output signal allows measuring the following types of interactions between the object and the fingertip:

- normal interactions, such as normal forces exerted by the robot when grasping an object;
- lateral interactions, such as lateral and shear forces occurring when the object slides between the fingers of the robot.

Given that only the thumb, index and middle fingers of the robot were used in the experiments, the resulting vector τ is composed as

$$\tau = \begin{pmatrix} \tau^{thumb} \\ \tau^{index} \\ \tau^{middle} \end{pmatrix} \in \mathbb{R}^9. \tag{6.8}$$

6.2.2 Description of the iCub humanoid robot

The robot platform adopted in this work is the iCub humanoid platform (Metta et al., 2010). The hands of the iCub are endowed with 9 joints that can be controlled using several control modes in order to decide the position of the fingertips. In this work, we only control the position of the thumb, index and middle fingers using voltage inputs. Each finger is equipped with two uSkin sensing elements, as previously described. Furthermore, a RGB camera system is mounted on the head of the robot. The camera system is used to track the actual position of the object, using ArUco markers (Garrido-Jurado et al., 2014), and collect the data required for training and testing purposes.

6.2.3 Data collection and post-processing

In this section the procedures required to collect the training and testing data will be described. According to the differentiable Kalman filtering paradigm, described in 2.5, it is required to collect a set X of ground-truth trajectories of the state $\{x_{t,i}^{gt}\}$ and the associated tactile signals $\{\dot{\tau}_{t,i}\}$.

To this end, a repeatable controlled sliding experiment was designed. In this experiment, the object of interest is manually placed on a table in front of the robot in a fixed starting pose. The arm of the robot is moved near the object, in a fixed pose, such that the hand can grasp the object. Once the object has been grasped, the robot arm moves up and then the thumb, index and middle fingers are controlled in a way such that the grasp force is reduced periodically resulting in a sequence of stick-slip-like motions (see Fig. 6.3).

During each experiment, the position of the object $p_t^A \in \mathbb{R}$, is acquired at 30 Hz using an ArUco marker placed on the top of the object such that it is visible from the RGB camera system mounted on the head of the robot. The signal from the marker is then post-processed in order to obtain the ground-truth state.

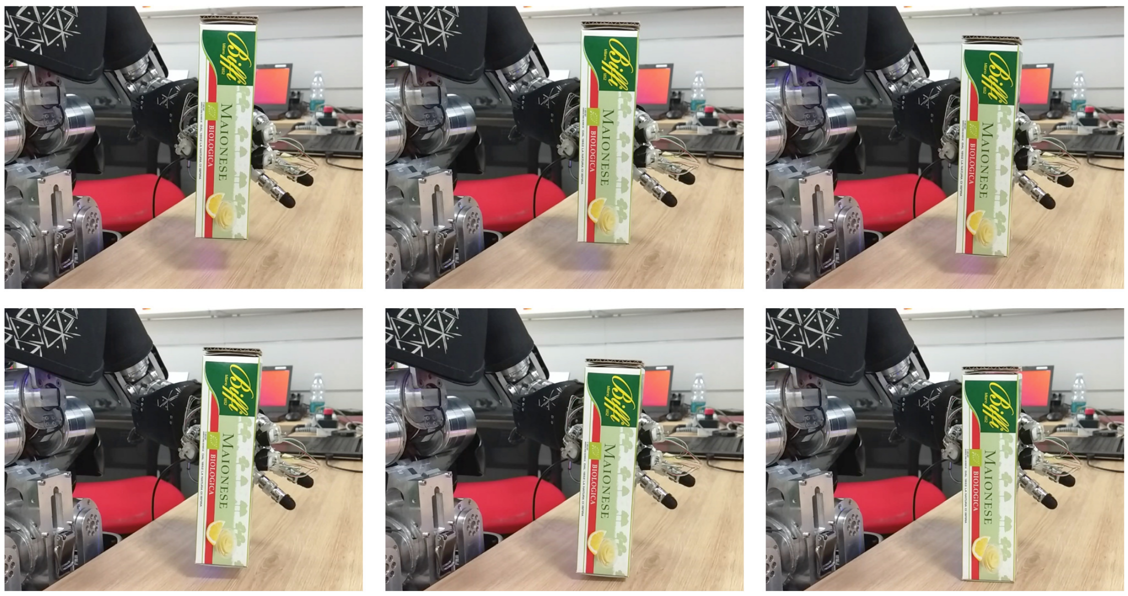


Figure 6.3 Outcome of the controlled sliding experiment with a box-shaped object.

At the same time, all the relevant tactile signals coming from the setup are also recorded at the same frequency. Only one of the two sensors mounted on each finger was recorded as it was noticed that the second sensing unit was not firmly in contact with the object. As previously explained in Sec. 6.1.7, the derivative of the tactile signal is obtained by running a linear Kalman filter using the raw tactile data τ_t.

Description of the controlled sliding experiment

Taking into account the kinematics of the robot hand, the placements of the sensing units on each finger and the reference frame attached to each sensor, it was found experimentally that the object sliding could be controlled by regulating the z channel of one of the two sensors mounted on each finger. These channels will be indicated as τ_z^{thumb}, τ_z^{index} and τ_z^{middle}. Indeed, they are proportional to the displacement of the sensor membrane in a direction that is approximately orthogonal to the surface of the object. The regulation of these signals allows deciding the grasp quality and, eventually, to start and stop the object sliding.

The actual control is made by a PI controller running at 100 Hz. Specifically, the output of the controller is used to decide the voltage V_f of the proximal joint of the finger as follows:

$$V_f = k_{p,f}(\tau^f_{z,des} - \tau^f_z) + k_{i,f} \int (\tau^f_{z,des} - \tau^f_z)\, \mathrm{d}t, \tag{6.9}$$

where $f \in \{\text{index}, \text{middle}, \text{thumb}\}$, $k_{p,f}$ is the proportional gain for the finger f, $k_{i,f}$ is the integral gain for the finger f, τ^f_z is the z tactile channel for the finger f and $\tau^f_{z,des}$ is the desired value for the finger f. The distal joints of each finger were instead regulated to suitable configurations using the iCub built-in joint position control loops.

The desired values $\tau^f_{z,des}$ were chosen by taking inspiration from a former work on hierarchical grasp control using tactile feedback (Regoli et al., 2016). Given a desired grasp strength g_{des} in Newton, in this work the forces F^f_{des} exerted by the fingers on the object are partitioned as follows:

$$F^{index}_{des} = F^{middle}_{des} = \frac{g_{des}}{2}, \quad F^{thumb}_{des} = g_{des}. \tag{6.10}$$

A similar reasoning was replicated by defining a generalized grasp strength in the space of the sensors output, $\tau_{g,des}$, and assigning the following desired values:

$$\tau^{index}_{z,des} = \tau^{middle}_{z,des} = \frac{\tau_{g,des}}{2}, \quad \tau^{thumb}_{z,des} = \tau_{g,des}. \tag{6.11}$$

The design of the desired references is concluded by assigning a periodic smooth trajectory to the desired grasp strength $\tau_{g,des}$ such that it moves alternatively between a maximum value $\tau_{g,max}$ and a minimum value $\tau_{g,min}$. The two values were chosen, for each object, such that when the grasp strength is regulated to the maximum value the object does not move. Conversely, when it is regulated to the minimum value the object slides noticeably between the fingers of the robot hand. An example of the commanded and achieved trajectory for the index and middle fingers is shown in Fig. 6.4. Fig. 6.3 shows the outcome of the controlled sliding experiment using a box-shaped object.

Description of the post-processing procedures

The position of the object p^A_t, acquired using the ArUco marker, cannot be used as ground truth state x^{gt} due to the noise in the marker estimation and the absence of the velocity v_t in its output. In order to provide an almost noise-free ground-truth signal, a linear Kalman Smoother was adopted (see Algorithm 10 in Sec. 2.4.3).

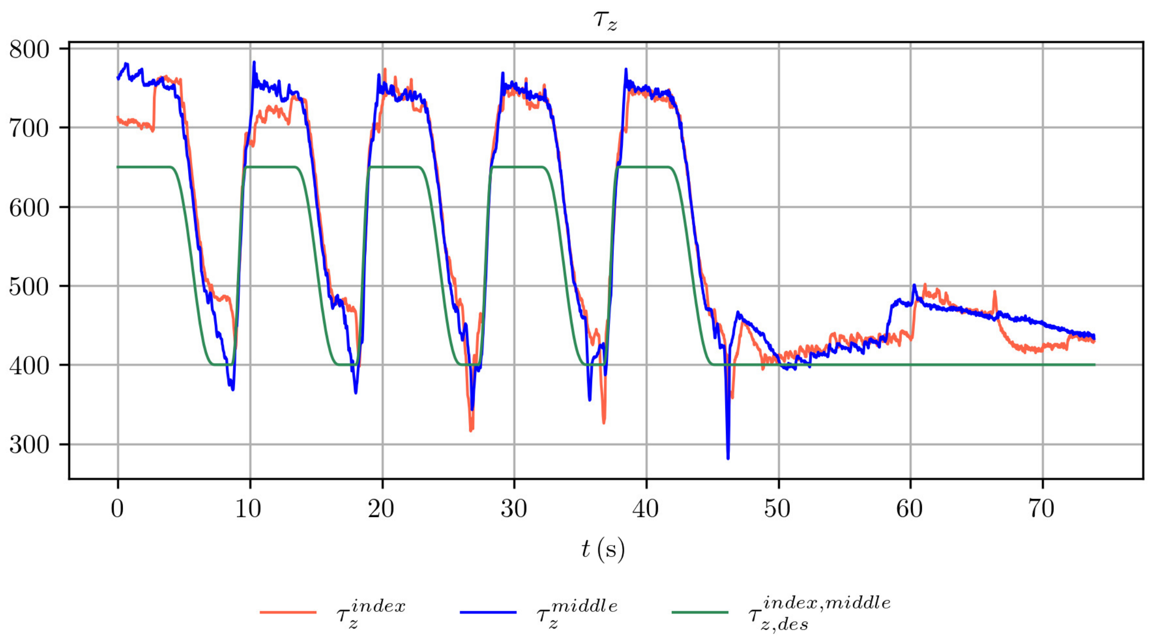


Figure 6.4 Comparison between the desired grasp strength and the achieved grasp strength for the index and middle fingers during the execution of a controlled sliding experiment.

In the specific scenario, the state to be smoothed x_t^{sm} is set equal to the state of interest x_t:

$$x_t^{sm} = \begin{pmatrix} p_t^{sm} \\ v_t^{sm} \end{pmatrix} \in \mathbb{R}^2. \tag{6.12}$$

It is assumed that the state changes according to a simple constant velocity model with Gaussian noise (Bar-Shalom et al., 2002):

$$\begin{aligned} x_t^{sm} &= \begin{pmatrix} p_t^{sm} \\ v_t^{sm} \end{pmatrix} = \begin{pmatrix} p_{t-1}^{sm} + v_{t-1}^{sm}\Delta_t \\ v_{t-1}^{sm} \end{pmatrix} + w^{sm}, \\ w^{sm} &\sim \mathcal{N}(0, Q^{sm}), \end{aligned} \tag{6.13}$$

where Δ_t is the sampling time of the algorithm and $Q^{sm} \in \mathbb{R}^{2\times 2}$ is the process noise covariance matrix.

The Kalman smoother is then fed with the ArUco estimate p_t^A as the measurement z_t^{sm} (after removing the first sample from the signal such that the position of the object

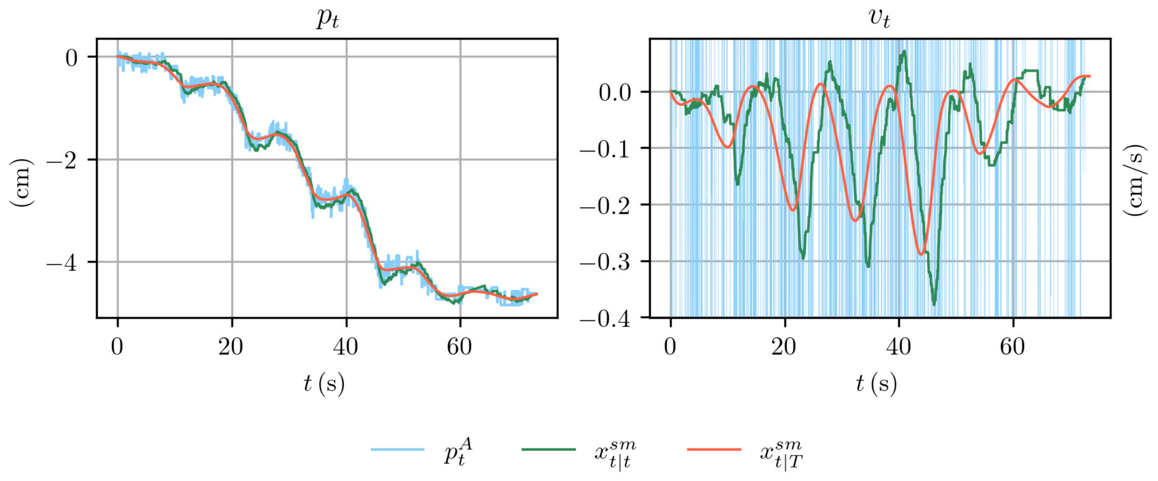


Figure 6.5 On the left, a comparison between the ground-truth position from the ArUco marker detection system p_t^A, its filtered version $x_{t|t}^{sm}$ and its smoothed version $x_{t|T}^{sm}$ obtained using a Kalman filter and smoother respectively. On the right, comparison between the ArUco velocity signal obtained using finite differences, filtered and smoothed version.

starts from zero). The measurement process is finally easily modelled as:

$$\begin{aligned} z_t^{sm} &= p_t^{sm} + \nu^{sm}, \\ \nu^{sm} &\sim \mathcal{N}(0, R^{sm}), \end{aligned} \tag{6.14}$$

where $R^{sm} \in \mathbb{R}$ is the measurement noise covariance. Given the smoothed state $x_{t,i}^{sm}$ for the i-th experiment, we define the ground-truth trajectory $x_{t,i}^{gt}$ as

$$x_{t,i}^{gt} := x_{t,i}^{sm}. \tag{6.15}$$

In all the experiments, we set $Q^{sm} = \text{diag}(0.01, 0.01)$ and $R^{sm} = 10.0$ as this configuration proved to be a good compromise between the smoothing action and the reduction of the error between p_t^{sm} and p_t^A.

An example of the outcome of the smoothing procedure is shown in Fig. 6.5 where the ArUco estimate is compared with the smoothed estimate $x_{t|T}^{sm}$ and the estimate $x_{t|t}^{sm}$ that could have been obtained using a linear Kalman filter. For the velocity component, an additional comparison is made with the signal obtained by finite differentiation of p_t^A. As it can be seen, the smoothed estimate contain less noise than the ArUco estimate and the filtered estimate. Furthermore, the smoothed velocity $v_{t|T}^{sm}$ has less

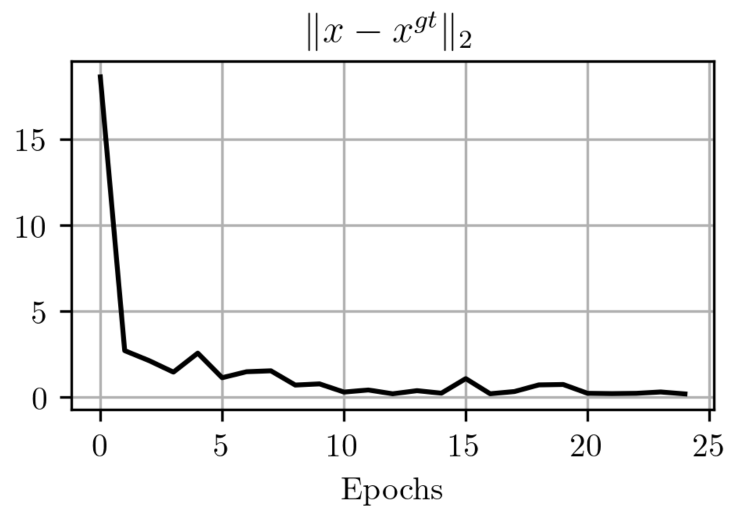


Figure 6.6 Example of the trend of the state RMSE, averaged on all the training trajectories, as a function of the number of epochs.

delay than the filtered counterpart and is significantly more reliable than using finite differences.

6.2.4 Training protocol

The whole filtering architecture is learned end-to-end using backpropagation through time, as described in Sec. 2.5.3, over subsequences of the trajectories of progressively increasing length T_s. Specifically, 5 epochs were executed for each subsequence T_s and the subsequences were chosen in the set $\{2, 4, 8, 16, 32\}$. In Fig. 6.6 we show an example of the trend of the state RMSE averaged on all the training trajectories along the training procedure.

We remark that the overall architecture, comprising the networks NN_{θ_1}, NN_{θ_2} and NN_{θ_3}, is trained end-to-end, i.e. the networks are not trained separately. Furthermore, the overall architecture needs to be trained separately for each considered object. Nonetheless, in the experimental evaluation we also tested training the architecture using the data from all the objects of interest.

The overall data collection and training pipeline is schematized in Fig. 6.1, on the right.

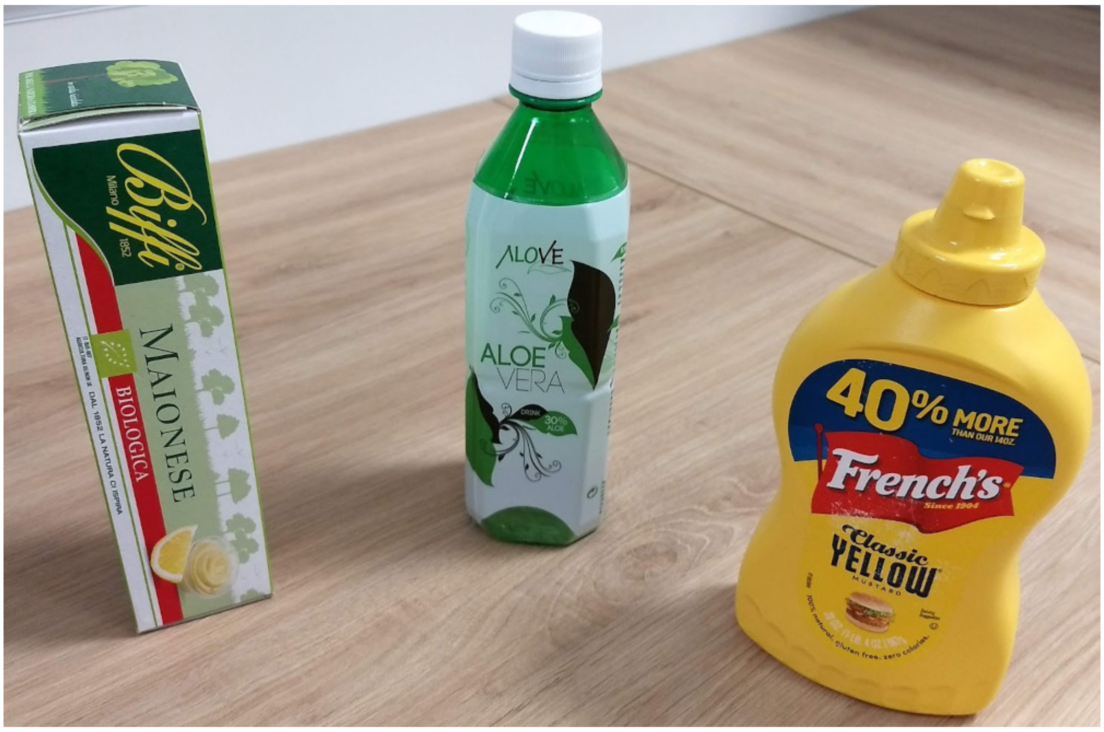


Figure 6.7 Picture of the objects used in the experiments. From the left, a box-shaped object, a water bottle and a mustard bottle.

6.3 Experimental evaluation

In this section, I will present the results of several experiments aimed at evaluating the performance of the proposed method.

Controlled sliding experiments, as described in Sec. 6.2.3, were executed using three objects of different shape and materials. The objects are shown in Fig. 6.7: from the left a box-shaped object made of paper, a water bottle made of plastic and a mustard bottle made of rigid plastic. Data has been collected during the experiments and then post-processed in order to train the filtering architecture.

The performance are evaluated on both the training and testing sequences in terms of the positional RMSE and linear velocity RMSE, defined as per Sec. 3.4.1, averaged on the training and testing trajectories respectively. The evaluation also includes several ablation studies aimed at assessing:

- changes in performance when using a subset of the tactile channels instead of the full set;

Table 6.1 Position and velocity RMSE and maximum training and testing errors using the full (xyz) set of the tactile channels for all the fingers.

data set	Training data using xyz tactile channels				Testing data using xyz tactile channels			
error	Positional error (cm)		Velocity error (cm/s)		Positional error (cm)		Velocity error (cm/s)	
metric	RMSE	max	RMSE	max	RMSE	max	RMSE	max
bottle	0.200	0.413	0.024	0.086	0.264	0.548	0.028	0.110
mustard	0.715	1.419	0.048	0.212	0.977	1.882	0.063	0.286
box	0.422	0.702	0.025	0.109	0.460	0.811	0.034	0.142
MEAN	0.446	0.845	0.032	0.136	0.567	1.080	0.042	0.179

- the necessity of using the derivative of the tactile measurements $\dot{\tau}$ instead of the plain measurements τ as required in Sec. 6.1.2;
- the generalization capabilities of the algorithm when trained using data of one object in the group and tested on the remaining ones;
- the relevance of the object weight in the training procedure.

Qualitative results on position and velocity tracking performance and considerations on the training and online inference times are also provided. The section is concluded with a discussion of the limitations of the proposed method.

In all the experiments, we set $p_0 = 0$, $v_0 = 0$ and $P_0 = \text{diag}(10^{-1}, 10^{-1})$. We recall that the process and measurement noise covariances Q and R, as per Eqs. (6.3) and (6.5), are learned during the training process.

6.3.1 Description of the collected data

One hundred [3] 1 minute-long experiments were executed for each object. Half of the sequences for each object were used as training data and the remaining sequences for testing. Sample signals from the ArUco marker detection system and of the tactile sensors output are shown in Fig. 6.8. As can be seen in the top-left plot, the trajectory of the object is approximately periodic and composed by a sequence of sliding motions, caused by the reduction in the grasp force, alternated with stationary phases. We remark that all the experiments follow a similar pattern.

6.3.2 Results on position and velocity tracking

In Table 6.1, we report the RMSE and max errors on position and velocity tracking for each object considering the training and testing sets. For each object the filter was trained using the data belonging to the training set of that object only.

[3]The number of experiments for the object “mustard bottle” amounts to 75.

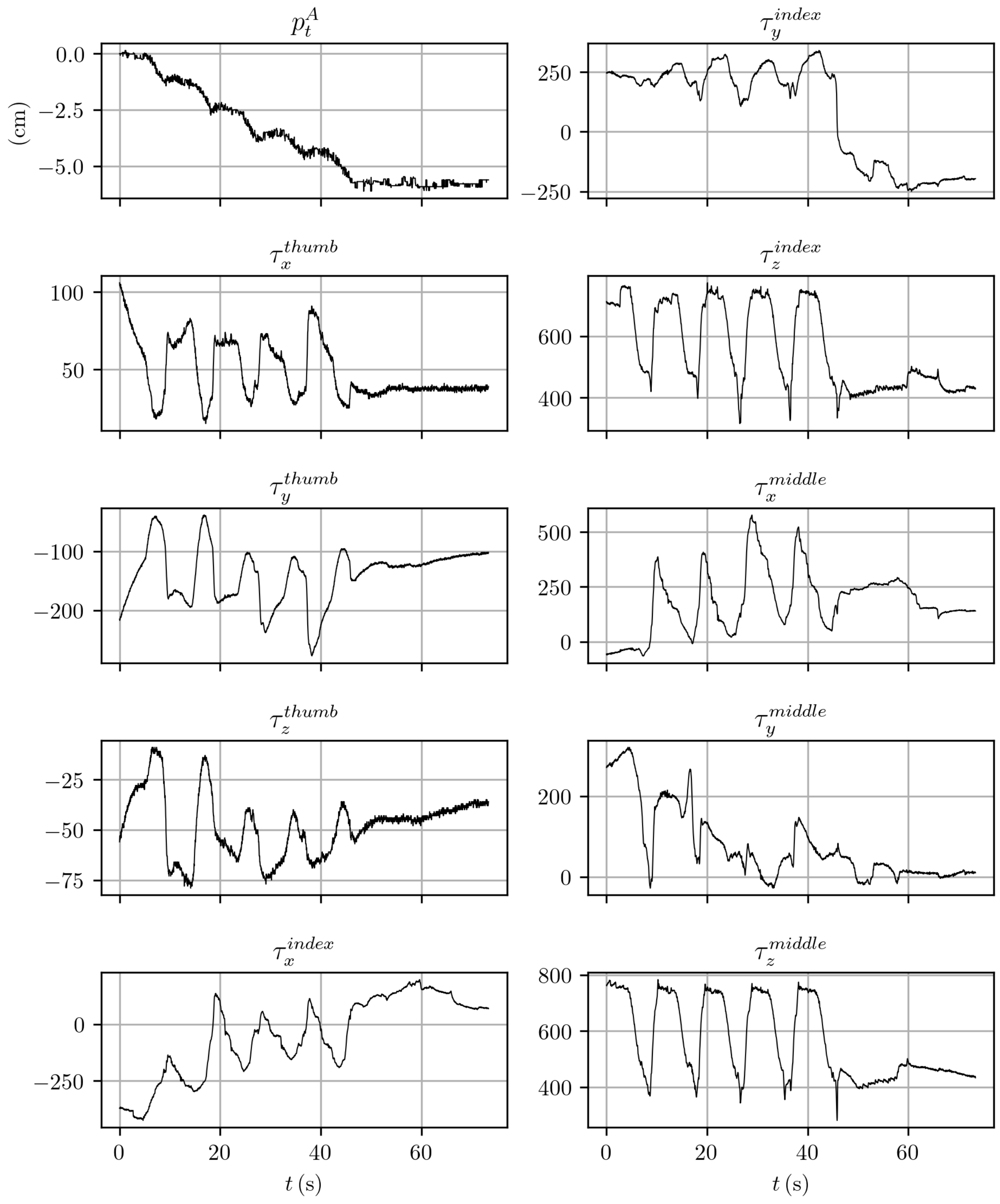


Figure 6.8 Data traces for one of the data collection experiment performed using the mustard bottle. The tactile signals correspond to the raw sensor reading using arbitrary units.

Table 6.2 Position and velocity RMSE and maximum testing errors using several configurations of the tactile channels (xyz, xy and z).

data set	Testing data											
error	Positional error (cm)						Velocity error (cm/s)					
metric	RMSE			max			RMSE			max		
channels	xyz	xy	z	xyz	xy	z	xyz	xy	z	xyz	xy	z
bottle	**0.264**	0.284	0.674	**0.548**	0.549	1.320	**0.028**	0.030	0.056	**0.110**	0.117	0.152
mustard	0.977	**0.706**	0.754	1.882	**1.423**	1.613	**0.063**	0.067	0.084	**0.286**	0.301	0.303
box	**0.460**	0.492	0.442	0.811	0.811	**0.810**	**0.034**	0.038	0.042	**0.142**	0.147	0.148
MEAN	0.567	**0.494**	0.623	1.080	**0.928**	1.248	**0.042**	0.045	0.061	**0.179**	0.188	0.201

The numerical results show that the training RMSE error in position is below 0.5 cm on average with a maximum average error of 0.845 cm. The performance degrades only slightly on the testing set: the testing RMSE error is below 0.6 cm on average, that is deemed as fairly accurate, with a maximum average error of 1.080 cm.

The training RMSE error in velocity is below 0.04 cm/s with a maximum average error of 0.136 cm/s. The performance slightly reduces on the testing set with RMSE errors below 0.05 cm/s and a maximum average error of 0.179 cm/s.

The best performance is achieved with the water bottle object, while for the mustard bottle it degrades more than other objects. We found experimentally that this condition depends on the fact that the sliding experiment has a larger variability over the experiments trials for this object than the others.

6.3.3 Relevance of the tactile channels

In Table 6.2, we consider the outcome of several training experiments where only a subset of the tactile channel set $\{\tau_x^f, \tau_y^f, \tau_z^f\}$ is considered for each finger f instead of the full set. Given the choice of the reference frame attached to each tactile sensor (see Fig. 6.2), we expect that the actual information on the sliding motion is stored in the x and y channels. On the other hand, the information stored in the z channel, while still useful in general, should not be required to estimate the sliding motion.

As it can be seen from the results on the testing set, the performance achieved using the xy configuration is similar to that achieved using the xyz configuration but slightly better in terms of positional error. On the other hand, the velocity error is lower for the xyz configuration but the difference is not substantial. If only the z channel is fed to the filter, the performance degrades such that for two objects out of three the maximum error in position is more than 1 cm.

The fact that the filter is able to produce a reasonable estimate even when using only the z channel depends on the fact that the evolution of this channel is constrained

Table 6.3 Comparison between position and velocity RMSE and maximum testing errors when using plain tactile measurements or their time derivative.

data set	Testing data using xy tactile channels							
error	Positional error (cm)				Velocity error (cm/s)			
metric	RMSE		max		RMSE		max	
measurement	τ	$\dot{\tau}$	τ	$\dot{\tau}$	τ	$\dot{\tau}$	τ	$\dot{\tau}$
bottle	0.381	**0.284**	0.648	**0.549**	0.033	**0.030**	**0.109**	0.117
mustard	2.698	**0.706**	4.045	**1.423**	0.107	**0.067**	0.401	**0.301**
box	0.699	**0.492**	1.199	**0.811**	0.049	**0.038**	0.158	**0.147**
MEAN	1.259	**0.494**	1.964	**0.928**	0.063	**0.045**	0.223	**0.188**

by the closed-loop controller, as we explained in Sec. 6.2.3. Indeed, in our experiments, when the maximum generalized grasp strength is commanded, i.e. $\tau_{g,des} = \tau_{g,max}$, the object is expected not to slide. Conversely, when the commanded signal shifts towards $\tau_{g,min}$ the object is expected to start sliding. Hence, the network could learn how to associate the z channel with the object velocity in a meaningful manner. However, we remark that this behavior is to be discouraged. It might happen that even when the minimum grasp strength is commanded, the object does not slide or barely slides. Possible reasons are non-idealities in the control loop or the fact that the object touches parts of the hand, or the table, thus preventing the sliding motion. In these cases, using the information stored in the z channel only, at training time, would make the training procedure ill-posed. This fact might explain why the achieved performance is worst than the other configurations. In Fig. 6.9, we provide qualitative results on position and velocity tracking for one of the experiments. In the plot labelled as "Estimate (z, mod.mustard)", we can actually see that using the z channel only might result in the estimation of non-zero velocities even if the object is not moving, which is undesirable.

6.3.4 Relevance of the derivative of the tactile measurements

In Table 6.3, we report the outcome of several training experiments where the filter is fed with the raw tactile measurements τ or with their derivative $\dot{\tau}$.

The numerical results demonstrate the necessity to adopt the derivative of the tactile measurements instead of the plain measurement. The RMSE positional error is reduced by $\approx 60\%$ on average and the maximum error by $\approx 50\%$.

Although the performance degrades considerably, it seems from the numerical results that it is still possible to train the network using the plain tactile measurements. However, as shown in Fig. 6.9, in the plot "Estimate (xy, no derivative)", the actual shape of the estimated signal is different from the ground truth in certain key aspects. At the beginning of the experiment, when the object is still not sliding, the slope of the

Table 6.4 Position and velocity RMSE testing errors using different combinations of training and testing sets. The model name "all" indicates a training set consisting in the union of the training sets of all the objects.

data set	Testing data using xy tactile channels							
metric	Positional RMSE (cm)				Velocity RMSE (cm/s)			
trained on	bottle	mustard	box	all	bottle	mustard	box	all
bottle	**0.284**	0.600	2.040	0.519	**0.030**	0.080	0.092	0.041
mustard	0.978	0.706	2.930	**0.664**	0.077	0.067	0.114	**0.066**
box	0.800	0.821	**0.492**	0.676	0.062	0.067	**0.038**	0.051
MEAN	0.687	0.709	1.821	**0.620**	0.056	0.071	0.081	**0.053**

estimated position is wrong and it reaches a non zero position almost instantaneously in a neighbourhood of $t = 0$. This is confirmed by the estimated velocity where, at $t = 0$, an initial spike in the velocity is reported. Furthermore, the shape of the estimated velocity does not follow the actual profile of the ground-truth velocity.

6.3.5 Generalization capabilities

In Table 6.4, we consider the outcome of several experiments where the learned filter is tested on a given object using the model trained on a different one.

Using the model trained on the data of objects bottle and mustard, the average RMSE positional error is in the order of 0.7 cm. By comparing with the performance in the ideal case, where each object is tested using the correct model, as in Table 6.2, it can be seen that the RMSE positional error approximately increased by an amount of 0.2 cm which can be considered a good compromise.

Conversely, if the model trained on the box is used, the performance on the other two objects degrades consistently with per object RMSE errors above 2 cm and average RMSE error above 1.8 cm. This outcome can be explained by the fact that this object is of a different material, i.e. paper instead of plastic, hence the frictional properties of the object are different resulting in a possible different response of the tactile sensors.

Nevertheless, the results from the same table show that the filter can be trained using the data from all the objects of interest (indicated as "all" in the table) and achieve the best performance. While this configuration requires the collection of training data for all the objects of interest, it allows using a single model online without the necessity to know the identity of the object being manipulated by the robot.

In Table 6.5, we report the outcome of another experiment aimed at understanding the relevance of the object weight in the training procedure. Specifically, we consider the scenario in which the object has weight w_{target} and the training data is available for two neighbouring weights w_1 and w_2, such that $w_1 < w_{target} < w_2$. In this experiment,

Table 6.5 Position and velocity RMSE testing errors for the box-shaped object with a weight of 125 grams when training using data corresponding to the target weight (125 grams) or the combination of weights (100 grams and 150 grams).

data set	Testing data using xy tactile channels			
error	Positional error (cm)		Velocity error (cm/s)	
metric	RMSE	max	RMSE	max
box (trained w/ 125 grams)	0.394	0.774	0.046	0.155
box (trained w/ 100 grams and 150 grams)	0.615	1.121	0.074	0.368

the network is trained using data corresponding to the weights w_1 and w_2 and the performance is then assessed using the testing data corresponding to the target weight w_{target}. The performance is compared with the baseline configuration in which the filter is trained using the data corresponding to the target weight w_{target}.

The aforementioned experiment was executed using the box-shaped object using $w_1 = 100$ grams, $w_2 = 150$ grams and $w_{target} = 125$ grams as the weights. We collected 50 experiments, 10 of which for testing purposes. In order to have a fair comparison with the baseline configuration, the number of experiments used for training was fixed. Specifically, we trained for the target weight by using all the 40 experiments designated as training data. When training for the neighbouring weights, instead, we picked 20 experiments from the training data of each weight, resulting in 40 experiments overall.

The tracking performance achieved when using the model trained on the neighbouring weights remains quite acceptable. In this configuration, the RMSE and maximum positional errors increase by approximately 0.22 cm and 0.35 cm with respect to the baseline, respectively. The RMSE and maximum velocity errors increase by approximately 0.03 cm/s and 0.21 cm/s.

6.3.6 Qualitative results

In Fig. 6.9, we report some qualitative results on position and velocity tracking for one of the experiments involving the mustard bottle. The ground-truth signal is compared with the estimates obtained using the xy tactile configuration in three cases:

- using the model trained on the data of the mustard bottle object;
- using the model trained on all the objects, indicated as “all”;
- using the plain tactile measurements instead of their derivatives.

Furthermore, we consider the model trained on the data of the mustard bottle object when using the z tactile channel only.

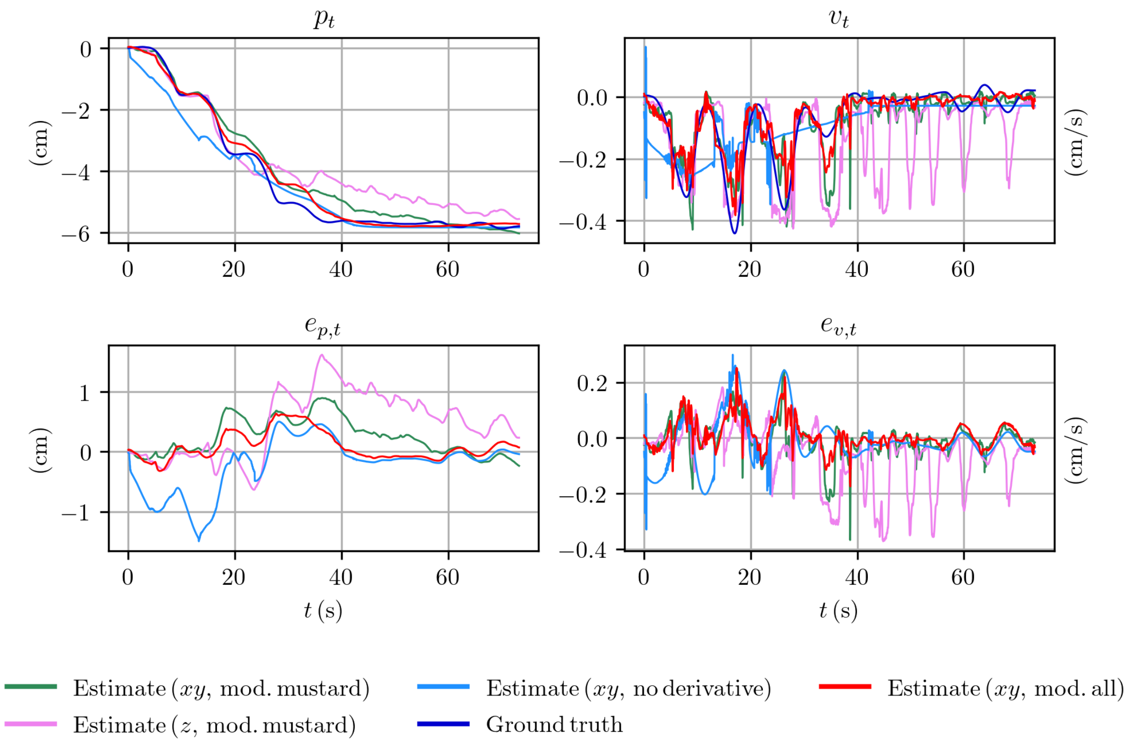


Figure 6.9 Comparison of the position p_t and velocity v_t trajectories, and associated errors $e_{p,t}$, $e_{v,t}$, for several configuration of the tactile measurements. The trajectories are from one of the experiments with the mustard bottle object. All the experiments, excluding the one labelled as "Estimate (z, mod.mustard)", were executed using the xy configuration of the tactile channels.

In the figure, we report the evolution of the position p_t and velocity v_t signals alongside the associated errors $e_{p,t}$ and $e_{v,t}$ with respect to the ground truth.

Overall, the configuration that best follows the actual profiles is given by the model trained on all the data. The model that uses only the z tactile channel fails in estimating the correct velocity in the final part of the experiment. Conversely, the model using the plain tactile measurements fails to estimate the correct position in the initial part of the experiment and completely fails to estimate the actual velocity of the object.

In Fig. 6.10, we show the outcome of running the filter online using the tactile data from the iCub humanoid robot. In each frame, we show, on the right, the estimate of the object position and the readings from the ArUco detection system. At the same time, we show the actual value of the desired grasp strength.

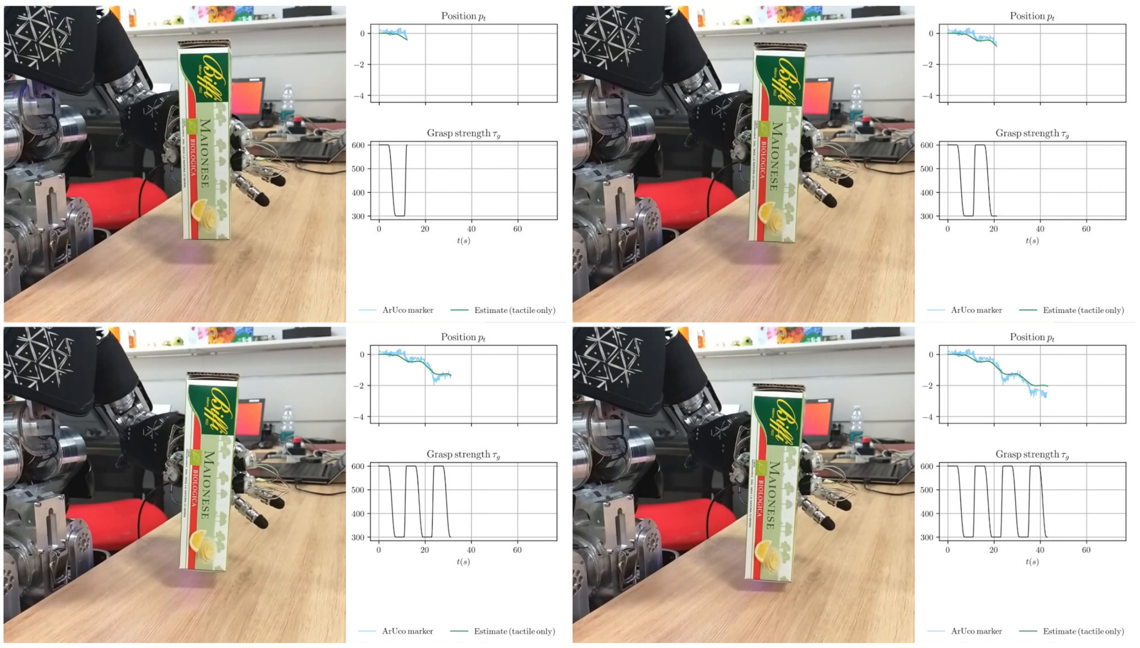


Figure 6.10 The output of the proposed tracker while it runs online using the tactile data from the iCub humanoid robot.

6.3.7 Training and inference times

Although the software implementation employed for the experiments uses the GPU-enabled machine learning framework PyTorch (Paszke et al., 2019), the experiments were run using CPU computations, instead of a GPU, as it was found that both training and inference times were lower. This outcome is expected given that our application does not involve images as input to the network architecture, which typically require the use of GPUs to reduce both training and inference times.

Taking into account the number of experiments whose data is used for training (see Sec. 6.3.1) and the adopted training protocol (see Sec. 6.2.4), the training procedure for a single object completes in 22 minutes and 55 seconds. The time required for smoothing both the training and testing data, as per the Sec. 6.2.3, amounts to 52 seconds while in order to evaluate the derivatives of the tactile measurements for all the experiments, as per the Sec. 6.1.7, the required time amounts to 5 minutes and 14 seconds.

Regarding inference times, the overall filtering pipeline can run at 119 Hz (including the time required to evaluate the derivatives of the tactile measurements), providing real-time state feedback for robot control purposes.

All the presented experiments were executed on an Intel i7-9750H multi-core CPU.

6.3.8 Assumptions and limitations

In this section I will further discuss the assumptions this work relies on, as per Sec. 6.1.1, and list possible limitations of proposed method.

Handling of object rotations

In this work, the focus was on the problem of tracking pure translational sliding motions instead of general slippage motions involving rotations of the object. Similarly to other works from the slip detection literature (Veiga et al., 2018), the adopted data collection procedure is not designed to capture object rotations. While ground-truth data for object translational motions can be easily collected even by using a simple marker and a limited amount of data post-processing, rotations during slippage might be of limited and subtle magnitude, hence more difficult to collect due to the noise in the marker pose detection process. Furthermore, in-hand object rotations heavily depend on the position of the contact points with respect to the object center of mass and on the inertial properties of the object. However, a precise control of the contact points, useful to achieve a repeatable data collection procedure, is out of scope of this Thesis. Nonetheless, I deem important to study how to account for object rotations in the proposed architecture and I plan to investigate this in future research. In this respect, we remark that some works, e.g. (Costanzo, 2021; Costanzo et al., 2020), have already explored the problem of tracking the rotational motion due to the object slippage using model-based approaches.

Handling of external disturbances

In this work, the problem of object sliding tracking is tackled in a scenario involving in-hand object manipulations in which gravity is the only external force applied to the object. It is important to remark that this scenario is not unrealistic and that it is adopted by other works, e.g. in the literature on tactile-based object manipulation and perception (Meier et al., 2016), (Bimbo et al., 2015), (Liang et al., 2020), (Suresh et al., 2021). Indeed, it is important to first address this scenario before considering the more general case in which other forces and external disturbances act on the object.

Characteristics of the considered objects

In this work, the considered objects comprise two prism-shaped objects (i.e. the box-shaped object and the water bottle made of plastic) and one object with non-convex shape, i.e. the mustard bottle. The former are characterized by a constant cross section such that the fingers can slide freely on the object surface while avoiding to be blocked by irregular changes in the object shape. While the mustard bottle presents some irregularity in the shape of the lateral edges, the experiments were limited to the case in which the fingers slide along the wider surface of the bottle. Although it is believed that tracking the sliding motion along irregular surfaces would be useful, it is also recognized that this problem is connected more with the ability of the fingers controller to follow an irregularly-shaped surface rather than the ability of the filtering architecture to process the signal produced by the sensors in that scenario.

Regarding the size, and especially the length, of the objects, the considered objects have at least one direction with a non-negligible length (see Figs. 6.2 and 6.7) as compared to that of the palm of the hand of the adopted robotic platform, i.e. the iCub humanoid robot. The reason behind this choice is mostly practical, as there is interest in the problem of estimating the sliding motion of the object, hence it is required to have objects that could actually slide between the fingers of the hand for a certain number of centimeters. This assumption does not prevent the possibility to use the proposed pipeline with objects that are smaller than those adopted in the experiments, as long as the desired tracking precision stays within the average position error achieved by the proposed pipeline.

Application to closed-loop control

A natural question arising from the proposed experiments is whether it is possible to control the motion of the fingers continuously, using the feedback from the learned filter, in order to control the sliding velocity of the object and track a desired reference velocity. Furthermore, it would be important to understand whether the learned motion and measurement models can be utilized in the synthesis of a control system using model-based control techniques. I plan to investigate these aspects in future work.

Chapter 7

Conclusions and future directions

In this Thesis, I proposed several methods for tracking the pose and the velocity of an object using RGB-D images and tactile sensing. The proposed methods are *hybrid* as they combine Kalman filters for state tracking with Machine Learning techniques for object perception, from RGB images, and tactile sensing encoding. The hybrid approach effectively exploits techniques for state tracking without the need for learning the tracking task with time consuming data-driven training. On the other hand, learning techniques are useful to extract high-level information, from unstructured data as digital images, that the methods proposed in this Thesis help integrating in the tracking process.

The first contribution is the design of a real-time 6D object pose and velocity tracker, named MaskUKF, which combines depth information and 2D segmentation provided by a deep neural network within an Unscented Kalman filter (Chapter 4). The proposed algorithm uses the object segmentation to extract the part of the point cloud relative to the object and associate it with the 3D model of the object in order to update its pose and velocity. It performs similarly or better than other state-of-the-art pose estimators and trackers on a standard computer vision benchmark involving objects from the YCB Model Set. In this regard, it was found that MaskUKF is more robust to perturbations of the initial condition of the pose when compared with a full deep learning-based tracker. This property depends on the basin of attraction of the filter that is function of its parameters but does not depend on the training data as in the case of deep end-to-end alternatives.

MaskUKF also has modest training requirements that resolve to that of a deep neural network for object segmentation. Such a network can be trained once for a set of objects, while most deep learning-based trackers require a per-object training procedure. Moreover, training an object pose tracker end-to-end requires 6D pose annotations, which are much more complex to be achieved.

In addition, MaskUKF also provides an estimation of the velocity of the object. This turned out to be advantageous for closed-loop control. By using MaskUKF in our experiments with the iCub humanoid robot, it was possible to increase the bandwidth of the controller more than when using the feedback from a pose estimation network. This allowed reaching lower tracking errors on the task at hand, while avoiding instabilities.

The second main contribution of this work is the design of a real-time 6D object pose and velocity tracker, named ROFT, that can handle objects involved in moderate-to-fast motions with the help of optical flow (Chapter 5). Specifically, ROFT combines depth information, optical flow and external sources of object segmentation masks and 6D poses within a cascaded Kalman filtering architecture. The proposed pipeline is able to track the 6D object velocity from the 2D domain using the optical flow and then fuse the estimated velocities with the 6D pose measurements to provide real-time pose tracking. Remarkably, ROFT is designed to take advantage of high frame rate optical flow in order to handle low frame rate, hence delayed, object segmentation masks and 6D poses. This scenario is common if the latter are provided by deep neural networks.

While benchmarking tracking algorithms, I noticed that commonly adopted datasets, in particular the YCB-Video dataset, contain sequences with slowly moving objects and not sufficiently challenging for the state-of-the-art. An additional contribution of my thesis consisted in generating a dataset specifically designed to address this problem. The dataset, called Fast-YCB, contains scenes of objects from the YCB Model Set involved in fast motions. ROFT was compared against other pose tracking methods from the literature on the Fast-YCB dataset showing better performance, especially with respect to orientation tracking.

One of the key features of ROFT, that differentiates it from end-to-end approaches, is its modular design which allows increasing the performance by replacing the segmentation and pose estimation sources with better ones if at disposal. This aspect has been verified experimentally via ablative studies where the latter sources are replaced with their ideal ground-truth counterparts, resulting in a performance increase.

ROFT has been also successfully employed on the iCub humanoid robot to perform gaze-tracking and grasping of objects by using the tracked poses and velocities as references for the gaze controller and as inputs to the grasp planner, respectively.

The third and final contribution of this Thesis is the development of a method for real-time tracking of sliding motions of hand-held objects using tactile feedback (Chapter 6). This method was implemented using the differentiable Kalman filtering paradigm. Specifically, the measurement model, describing the association between the object state and the readings from the tactile sensors, is learned from data while the algorithmic structure of the Kalman filter is preserved. This combination is particularly useful if the working principle of the employed sensor is too complex to be modelled or even unknown.

The proposed method has been trained and tested with experimental data collected on the iCub humanoid robot platform that has been equipped with modern 3D magnetic-based soft tactile sensors. The experimental results on several objects showed good overall performance on the sliding tracking task using tactile feedback alone. Furthermore, they highlight and confirm the necessity of using 3D tactile sensing in order to track the sliding motion reliably.

The aforementioned contributions suggest possible future directions of research related to the presented work.

As mentioned in Chapter 4, MaskUKF requires the 3D model of the object, without textures, in order to track the object pose. A possible extension of this work would entail substituting the complete model of the object with simpler descriptions, e.g. given by superquadrics primitives reconstructed from the point cloud. This extension would not only allow to relax the requirements on object modelling but also to understand the effects of using less precise models on the tracking performance.

Considering the ROFT pipeline, described in Chapter 5, it requires, as input, a source of 6D pose measurements, usually provided by a pose estimation network. One possible limitation is that the pose estimation network needs to be trained for each object separately. A solution to mitigate this requirement would be to substitute the network with a category level-based one which generalizes to unseen instances of objects belonging to specific categories. A different solution, would be to exploit recent advances in 2D object keypoints detection and use them, in conjunction with depth measurements and the 3D object model, to substitute the 6D pose measurements.

The advantage of the latter solution is that the keypoints detection network could be trained once, for all the objects of interest, thus relaxing the aforementioned limitation.

As mentioned in Chapter 5, the ROFT pipeline makes use of optical flow measurements which might be less effective in presence of out-of-plane object rotations, due to object self-occlusions. This limitation leads to underestimation of the angular velocity of the object. A viable solution to this limitation would be to exploit the differentiable Kalman filtering approach to learn an additional measurement model, to be integrated in ROFT, providing the missing part of the angular velocity when necessary.

With regard to the third contribution, as per Chapter 6, a clear direction of research is motivated by the necessity to avoid collecting real data of the tactile sensor to train the proposed method. In this respect, the development of simulation platforms that can reproduce the behavior of soft materials, a common ingredient of modern robotic tactile sensors, could help exploiting simulated data for the training stage. Nonetheless, acquiring real labelled data might still be important for testing purposes and to augment the dataset in order to improve generalization. In this respect, the visual trackers developed in this Thesis could be used to collect such data.

In conclusion, the studies presented in this Thesis explored how a balanced combination of Kalman filtering for state tracking and Machine learning allows attaining remarkable improvements in object pose tracking, with respect to full data-driven approaches, in terms of accuracy, modularity, relaxation of the training requirements and suitability for robot motion control purposes. The proposed algorithms will benefit a wide range of robotic applications that require precise information on the state of the objects, including object tracking, grasping and manipulation.

References

Anderson, B. D. and Moore, J. B. (2012). *Optimal filtering*. Courier Corporation.

Bar-Shalom, Y., Kirubarajan, T., and Li, X.-R. (2002). *Estimation with Applications to Tracking and Navigation: Theory, Algorithms and Software*. John Wiley & Sons, Inc., USA.

Bauer, D., Patten, T., and Vincze, M. (2020). VeREFINE: Integrating object pose verification with physics-guided iterative refinement. *IEEE Robotics and Automation Letters*, 5(3):4289–4296.

Besl, P. J. and McKay, N. D. (1992). A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256.

Bimbo, J., Kormushev, P., Althoefer, K., and Liu, H. (2015). Global estimation of an object's pose using tactile sensing. *Advanced Robotics*, 29(5):363–374.

Brown, K. R. (1991). *The theory of the GPS composite clock*. Institute of Navigation.

Brox, T., Rosenhahn, B., Cremers, D., and Seidel, H.-P. (2006). High Accuracy Optical Flow Serves 3-D Pose Tracking: Exploiting Contour and Flow Based Constraints. In *Computer Vision – ECCV 2006*, pages 98–111, Berlin, Heidelberg. Springer Berlin Heidelberg.

Calli, B., Singh, A., Walsman, A., Srinivasa, S., Abbeel, P., and Dollar, A. M. (2015). The YCB object and Model Set: Towards Common Benchmarks for Manipulation Research. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 510–517.

Ceola, F., Maiettini, E., Pasquale, G., Rosasco, L., and Natale, L. (2021). Fast Object Segmentation Learning with Kernel-based Methods for Robotics. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13581–13588.

Chen, S. Y. (2012). Kalman Filter for Robot Vision: A Survey. *IEEE Transactions on Industrial Electronics*, 59(11):4409–4420.

Chiella, A. C., Teixeira, B. O., and Pereira, G. A. S. (2019). Quaternion-Based Robust Attitude Estimation Using an Adaptive Unscented Kalman Filter. *Sensors*, 19(10):2372.

Choi, C. and Christensen, H. I. (2013). RGB-D object tracking: A particle filter approach on GPU. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1084–1091.

Costanzo, M. (2021). Control of robotic object pivoting based on tactile sensing. *Mechatronics*, 76:102545.

Costanzo, M., De Maria, G., Lettera, G., and Natale, C. (2021). Can Robots Refill a Supermarket Shelf?: Motion Planning and Grasp Control. *IEEE Robotics Automation Magazine*, 28(2):61–73.

Costanzo, M., De Maria, G., and Natale, C. (2020). Two-Fingered In-Hand Object Handling Based on Force/Tactile Feedback. *IEEE Transactions on Robotics*, 36(1):157–173.

Curless, B. and Levoy, M. (1996). A Volumetric Method for Building Complex Models from Range Images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312.

Curtin, R. R. and Gardner, A. B. (2016). Fast Approximate Furthest Neighbors with Data-Dependent Candidate Selection. In Amsaleg, L., Houle, M. E., and Schubert, E., editors, *Similarity Search and Applications*, pages 221–235, Cham. Springer International Publishing.

de Lima, D. P., Sanches, R. F. V., and Pedrino, E. C. (2017). Neural Network Training Uusing Unscented and Extended Kalman Filter. *Robotics & Automation Engineering Journal*, 1:555–568.

Dellaert, F. and Kaess, M. (2017). Factor Graphs for Robot Perception. *Foundations and Trends in Robotics*, 6(1-2):1–139.

Deng, X., Mousavian, A., Xiang, Y., Xia, F., Bretl, T., and Fox, D. (2019). PoseRBPF: A Rao-Blackwellized Particle Filter for 6D Object Pose Estimation. In *Proceedings of Robotics: Science and Systems*.

Deng, X., Mousavian, A., Xiang, Y., Xia, F., Bretl, T., and Fox, D. (2021). PoseRBPF: A Rao-Blackwellized Particle Filter for 6D Object Pose Tracking. *IEEE Transactions on Robotics*, 37(5):1328–1342.

Denisko, D. and Hoffman, M. M. (2018). Classification and interaction in random forests. *Proceedings of the National Academy of Sciences*, 115(8):1690–1692.

Denninger, M., Sundermeyer, M., Winkelbauer, D., Olefir, D., Hodan, T., Zidan, Y., Elbadrawy, M., Knauer, M., Katam, H., and Lodhi, A. (2020). BlenderProc: Reducing the Reality Gap with Photorealistic Rendering. In *International Conference on Robotics: Science and Systems, RSS 2020*.

Dong, S., Ma, D., Donlon, E., and Rodriguez, A. (2019). Maintaining Grasps within Slipping Bounds by Monitoring Incipient Slip. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3818–3824.

Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., Van Der Smagt, P., Cremers, D., and Brox, T. (2015). FlowNet: Learning Optical Flow with Convolutional Networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2758–2766.

Dubeau, E., Garon, M., Debaque, B., de Charette, R., and Lalonde, J.-F. (2020). RGB-D-E: Event Camera Calibration for Fast 6-DOF Object Tracking. In *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–135. IEEE.

Evensen, G. (2009). The Ensemble Kalman Filter for Combined State and Parameter Estimation. *IEEE Control Systems Magazine*, 29(3):83–104.

Fronckova, K. and Slaby, A. (2020). Kalman Filter Employment in Image Processing. In Gervasi, O., Murgante, B., Misra, S., Garau, C., Blečić, I., Taniar, D., Apduhan, B. O., Rocha, A. M. A., Tarantino, E., Torre, C. M., and Karaca, Y., editors, *Computational Science and Its Applications – ICCSA 2020*, pages 833–844, Cham. Springer International Publishing.

Gao, W. and Tedrake, R. (2019). FilterReg: Robust and Efficient Probabilistic Point-Set Registration Using Gaussian Filter and Twist Parameterization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11095–11104.

Garon, M. and Lalonde, J.-F. (2017). Deep 6-DOF Tracking. *IEEE Transactions on Visualization and Computer Graphics*, 23(11):2410–2418.

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., and Marín-Jiménez, M. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292.

Ge, R. and Loianno, G. (2021). VIPose: Real-time Visual-Inertial 6D Object Pose Tracking. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4597–4603.

Gentle, J. (2007). *Matrix Algebra: Theory, Computations, and Applications in Statistics*. Springer Texts in Statistics. Springer.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. *Advances in Neural Information Processing Systems*, 27.

Grinvald, M., Tombari, F., Siegwart, R., and Nieto, J. (2021). TSDF++: A Multi-Object Formulation for Dynamic Object Tracking and Reconstruction. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14192–14198.

Gustafsson, F. (2010). *Statistical sensor fusion*. Studentlitteratur.

Haarnoja, T., Ajay, A., Levine, S., and Abbeel, P. (2016). Backprop KF: Learning Discriminative Deterministic State Estimators. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Hampali, S., Rad, M., Oberweger, M., and Lepetit, V. (2020). HOnnotate: A Method for 3D Annotation of Hand and Object Poses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3196–3206.

Hang, K., Bircher, W. G., Morgan, A. S., and Dollar, A. M. (2020). Hand–object configuration estimation using particle filters for dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(14):1760–1774.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation.* International edition. Prentice Hall.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988.

Hebert, P., Hudson, N., Ma, J., Howard, T., Fuchs, T., Bajracharya, M., and Burdick, J. (2012). Combined Shape, Appearance and Silhouette for Simultaneous Manipulator and Object Tracking. In *2012 IEEE International Conference on Robotics and Automation*, pages 2405–2412.

Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., and Navab, N. (2012). Model Based Training, Detection and Pose Estimation of Texture-less 3D Objects in Heavily Cluttered Scenes. In *Proceedings of the 11th Asian conference on Computer Vision-Volume Part I*, pages 548–562.

Hodan, T., Haluza, P., Obdržálek, Š., Matas, J., Lourakis, M., and Zabulis, X. (2017). T-LESS: An RGB-D dataset for 6D Pose Estimation of Texture-less Objects. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 880–888. IEEE.

Holgado, A. C., Piga, N., Tomo, T. P., Vezzani, G., Schmitz, A., Natale, L., and Sugano, S. (2019). Magnetic 3-axis Soft and Sensitive Fingertip Sensors Integration for the iCub Humanoid Robot. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, pages 1–8.

Huynh, D. Q. (2009). Metrics for 3D Rotations: Comparison and Analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164.

Issac, J., Wüthrich, M., Cifuentes, C. G., Bohg, J., Trimpe, S., and Schaal, S. (2016). Depth-Based Object Tracking using a Robust Gaussian Filter. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 608–615.

Iwase, S., Liu, X., Khirodkar, R., Yokota, R., and Kitani, K. M. (2021). RePOSE: Fast 6D Object Pose Refinement via Deep Texture Rendering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3303–3312.

Izatt, G., Mirano, G., Adelson, E., and Tedrake, R. (2017). Tracking Objects with Point Clouds from Vision and Touch. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4000–4007. IEEE.

Jongeneel, M., Bernardino, A., van de Wouw, N., and Saccon, A. (2021). Model-Based Visual Object Tracking with Impact Collision Models. In *International Conference on Intelligent Robots and Systems (IROS).*

Joseph Tan, D., Tombari, F., Ilic, S., and Navab, N. (2015). A Versatile Learning-based 3D Temporal Tracker: Scalable, Robust, Online. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 693–701.

Julier, S. J. and Uhlmann, J. K. (2004). Unscented Filtering and Nonlinear Estimation. *Proceedings of the IEEE*, 92(3):401–422.

Kaess, M., Ranganathan, A., and Dellaert, F. (2008). iSAM: Incremental Smoothing and Mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378.

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME-Journal of Basic Engineering*, 82(Series D):35–45.

Kitagawa, G. (1994). The two-filter formula for smoothing and an implementation of the Gaussian-sum smoother. *Annals of the Institute of Statistical Mathematics*, 46(4):605–623.

Kloss, A., Martius, G., and Bohg, J. (2021). How to train your differentiable filter. *Autonomous Robots*, 45(4):561–578.

Koenig, N. and Howard, A. (2004). Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154, Sendai, Japan.

Koval, M. C., Dogar, M. R., Pollard, N. S., and Srinivasa, S. S. (2013). Pose Estimation for Contact Manipulation with Manifold Particle Filters. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4541–4548. IEEE.

Kumru, M. and Özkan, E. (2021). Three-Dimensional Extended Object Tracking and Shape Learning using Gaussian Processes. *IEEE Transactions on Aerospace and Electronic Systems*, 57(5):2795–2814.

Lambeta, M., Chou, P.-W., Tian, S., Yang, B., Maloon, B., Most, V. R., Stroud, D., Santos, R., Byagowi, A., Kammerer, G., Jayaraman, D., and Calandra, R. (2020). DIGIT: A Novel Design for a Low-Cost Compact High-Resolution Tactile Sensor With Application to In-Hand Manipulation. *IEEE Robotics and Automation Letters*, 5(3):3838–3845.

Lee, M. A., Yi, B., Martín-Martín, R., Savarese, S., and Bohg, J. (2020). Multimodal Sensor Fusion with Differentiable Filters. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10444–10451.

Leeb, F., Byravan, A., and Fox, D. (2019). Motion-Nets: 6D Tracking of Unknown Objects in Unseen Environments using RGB. *arXiv preprint arXiv:1910.13942*.

Li, H. and Stueckler, J. (2021). Tracking 6-DoF Object Motion from Events and Frames. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14171–14177.

Li, Y., Wang, G., Ji, X., Xiang, Y., and Fox, D. (2018). DeepIM: Deep Iterative Matching for 6D Pose Estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 683–698.

Liang, J., Handa, A., Wyk, K. V., Makoviychuk, V., Kroemer, O., and Fox, D. (2020). In-Hand Object Pose Tracking via Contact Feedback and GPU-Accelerated Robotic Simulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6203–6209.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. In *Computer Vision – ECCV 2014*, pages 740–755, Cham. Springer International Publishing.

Lynch, K. and Park, F. (2017). *Modern Robotics*. Cambridge University Press.

Ma, Y., Soatto, S., Košecká, J., and Sastry, S. (2004). *An Invitation to 3-D Vision: from Images to Geometric Models*, volume 26. Springer.

Manhardt, F., Kehl, W., Navab, N., and Tombari, F. (2018). Deep Model-Based 6D Pose Refinement in RGB. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 800–815.

Martin, S., Zoltan-Csaba, M., Maximilian, D., Manuel, B., and Rudolph, T. (2018). Implicit 3D Orientation Learning for 6D Object Detection from RGB Images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 699–715.

Masuda, M., Sekikawa, Y., Fujii, R., and Saito, H. (2021). Neural Implicit Event Generator for Motion Tracking. *arXiv preprint arXiv:2111.03824*.

McGee, L. A., Schmidt, S. F., Mcgee, L. A., and Sc, S. F. (1985). Discovery of the Kalman Filter as a Practical Tool for Aerospace and Industry. In *National Aeronautics and Space Administration, Ames Research*. Citeseer.

McManus, C. and Barfoot, T. (2011). A Serial Approach to Handling High-Dimensional Measurements in the Sigma-Point Kalman Filter. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA.

Meier, M., Walck, G., Haschke, R., and Ritter, H. J. (2016). Distinguishing Sliding from Slipping during Object Pushing. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5579–5584.

Metta, G., Natale, L., Nori, F., Sandini, G., Vernon, D., Fadiga, L., Von Hofsten, C., Rosander, K., Lopes, M., Santos-Victor, J., et al. (2010). The iCub humanoid robot: An open-systems platform for research in cognitive development. *Neural Networks*, 23(8):1125 – 1134.

Miele, A. (2016). *Flight Mechanics: Theory of Flight Paths*. Courier Dover Publications.

Mitash, C., Boularias, A., and Bekris, K. E. (2019). Robust 6D Object Pose Estimation with Stochastic Congruent Sets. In *29th British Machine Vision Conference, BMVC 2018*.

Mozer, M. C. (1989). A Focused Backpropagation Algorithm for Temporal Pattern Recognition. *Complex systems*, 3(4):349–381.

Nguyen, P. D., Bottarel, F., Pattacini, U., Hoffmann, M., Natale, L., and Metta, G. (2018). Merging Physical and Social Interaction for Effective Human-Robot Collaboration. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–9.

NVIDIA (2021). NVIDIA Optical Flow SDK.

Osher, S. and Fedkiw, R. (2003). Signed Distance Functions. In *Level set methods and dynamic implicit surfaces*, pages 17–22. Springer.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Pattacini, U., Nori, F., Natale, L., Metta, G., and Sandini, G. (2010). An Experimental Evaluation of a Novel Minimum-Jerk Cartesian Controller for Humanoid Robots. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1668–1674.

Pauwels, K., Rubio, L., and Ros, E. (2016). Real-Time Pose Detection and Tracking of Hundreds of Objects. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(12):2200–2214.

Peng, S., Liu, Y., Huang, Q., Zhou, X., and Bao, H. (2019). PVNet: Pixel-Wise Voting Network for 6DoF Pose Estimation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4556–4565. IEEE.

Periyasamy, A. S., Schwarz, M., and Behnke, S. (2019). Refining 6D Object Pose Predictions using Abstract Render-and-Compare. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, pages 739–746. IEEE.

Piga, N. A., Bottarel, F., Fantacci, C., Vezzani, G., Pattacini, U., and Natale, L. (2021a). MaskUKF: An Instance Segmentation Aided Unscented Kalman Filter for 6D Object Pose and Velocity Tracking. *Frontiers in Robotics and AI*, 8:38.

Piga, N. A., Onyshchuk, Y., Pasquale, G., Pattacini, U., and Natale, L. (2022). ROFT: Real-Time Optical Flow-Aided 6D Object Pose and Velocity Tracking. *IEEE Robotics and Automation Letters*, 7(1):159–166.

Piga, N. A., Pattacini, U., and Natale, L. (2021b). A Differentiable Extended Kalman Filter for Object Tracking Under Sliding Regime. *Frontiers in Robotics and AI*, 8:251.

Pressigout, M., Marchand, E., and Memin, E. (2008). Hybrid tracking approach using optical flow and pose estimation. In *2008 15th IEEE International Conference on Image Processing*, pages 2720–2723. IEEE.

Rauch, H. (1963). Solutions to the Linear Smoothing Problem. *IEEE Transactions on Automatic Control*, 8(4):371–372.

Regoli, M., Pattacini, U., Metta, G., and Natale, L. (2016). Hierarchical Grasp Controller using Tactile Feedback. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 387–394.

Revach, G., Shlezinger, N., Ni, X., Escoriza, A. L., van Sloun, R. J., and Eldar, Y. C. (2021). KalmanNet: Neural Network Aided Kalman Filtering for Partially Known Dynamics. *arXiv preprint arXiv:2107.10043*.

Ristic, B., Arulampalam, S., and Gordon, N. (2003). *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech house.

Rogers, R. M. (2003). *Applied Mathematics in Integrated Navigation Systems*, volume 1. AIAA.

Roncone, A., Pattacini, U., Metta, G., and Natale, L. (2016). A Cartesian 6-DoF Gaze Controller for Humanoid Robots. In *Proceedings of Robotics: Science and Systems*, AnnArbor, Michigan.

Runz, M., Buffier, M., and Agapito, L. (2018). MaskFusion: Real-Time Recognition, Tracking and Reconstruction of Multiple Moving Objects. In *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 10–20. IEEE.

Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *2011 IEEE International Conference on Robotics and Automation*, pages 1–4.

Schmidt, T., Hertkorn, K., Newcombe, R., Marton, Z., Suppa, M., and Fox, D. (2015). Depth-Based Tracking with Physical Constraints for Robot Manipulation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 119–126. IEEE.

Schmidt, T., Newcombe, R., and Fox, D. (2014). DART: Dense Articulated Real-Time Tracking. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA.

Simon, D. (2006). *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. John Wiley & Sons.

Smith, C. (2006). On Vertex-Vertex Systems and Their Use in Geometric and Biological Modelling, PhD thesis, The University of Calgary.

Smith, G. L., Schmidt, S. F., and McGee, L. A. (1962). *Application of Statistical Filter Theory to the Optimal Estimation of Position and Velocity on Board a Circumlunar Vehicle*. National Aeronautics and Space Administration.

Sodhi, P., Kaess, M., Mukadam, M., and Anderson, S. (2021a). Learning Tactile Models for Factor Graph-based State Estimation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13686–13692.

Sodhi, P., Kaess, M., Mukadam, M., and Anderson, S. (2021b). PatchGraph: In-hand tactile tracking with learned surface normals. *arXiv preprint arXiv:2111.07524*.

Sola, J., Deray, J., and Atchuthan, D. (2018). A micro Lie theory for state estimation in robotics. *arXiv preprint arXiv:1812.01537*.

Song, C., Song, J., and Huang, Q. (2020). HybridPose: 6D Object Pose Estimation under Hybrid Representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 431–440.

Stevens, B. L., Lewis, F. L., and Johnson, E. N. (2015). *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. John Wiley & Sons.

Suresh, S., Bauza, M., Yu, K.-T., Mangelson, J. G., Rodriguez, A., and Kaess, M. (2021). Tactile SLAM: Real-time inference of shape and pose from planar pushing. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11322–11328.

Särkkä, S. (2013). *Bayesian Filtering and Smoothing*. Institute of Mathematical Statistics Textbooks. Cambridge University Press.

The CGAL Project (2022). *CGAL User and Reference Manual*. CGAL Editorial Board, 5.4 edition.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. The MIT Press.

Tjaden, H., Schwanecke, U., Schömer, E., and Cremers, D. (2019). A Region-Based Gauss-Newton Approach to Real-Time Monocular Multiple Object Tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1797–1812.

Tomo, T. P., Wong, W. K., Schmitz, A., Kristanto, H., Sarazin, A., Jamone, L., Somlor, S., and Sugano, S. (2016). A Modular, Distributed, Soft, 3-Axis Sensor System for Robot Hands. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 454–460.

Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., Fox, D., and Birchfield, S. (2018). Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects. In *Conference on Robot Learning*, pages 306–316.

Van Der Merwe, R., Doucet, A., De Freitas, N., and Wan, E. (2000). The Unscented Particle Filter. *Advances in neural information processing systems*, 13.

Veiga, F., Peters, J., and Hermans, T. (2018). Grip Stabilization of Novel Objects Using Slip Prediction. *IEEE Transactions on Haptics*, 11(4):531–542.

Vezzani, G., Pattacini, U., Battistelli, G., Chisci, L., and Natale, L. (2017). Memory Unscented Particle Filter for 6-DOF Tactile Localization. *IEEE Transactions on Robotics*, 33(5):1139–1155.

Viña, F. E., Karayiannidis, Y., Pauwels, K., Smith, C., and Kragic, D. (2015). In-hand manipulation using gravity and controlled slip. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5636–5641.

Wan, E. A. and Merwe, R. V. D. (2000). The Unscented Kalman Filter for Nonlinear Estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pages 153–158.

Wang, C., Xu, D., Zhu, Y., Martin-Martin, R., Lu, C., Fei-Fei, L., and Savarese, S. (2019). DenseFusion: 6D Object Pose Estimation by Iterative Dense Fusion. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3343–3352.

Wells, C. (2013). *The Kalman Filter in Finance*, volume 32. Springer Science & Business Media.

Wen, B., Mitash, C., Ren, B., and Bekris, K. E. (2020a). se(3)-TrackNet: Data-driven 6D Pose Tracking by Calibrating Image Residuals in Synthetic Domains. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10367–10373. IEEE.

Wen, B., Mitash, C., Soorian, S., Kimmel, A., Sintov, A., and Bekris, K. E. (2020b). Robust, Occlusion-aware Pose Estimation for Objects Grasped by Adaptive Hands. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6210–6217.

Wettels, N., Fishel, J. A., and Loeb, G. E. (2014). *Multimodal Tactile Sensor*, pages 405–429. Springer International Publishing, Cham.

Williams, O. and Fitzgibbon, A. (2006). Gaussian Process Implicit Surfaces. In *Gaussian Processes in Practice*.

Wong, J. M., Kee, V., Le, T., Wagner, S., Mariottini, G., Schneider, A., Hamilton, L., Chipalkatty, R., Hebert, M., Johnson, D. M. S., Wu, J., Zhou, B., and Torralba, A. (2017). SegICP: Integrated Deep Semantic Segmentation and Pose Estimation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5784–5789.

Wüthrich, M., Pastor, P., Kalakrishnan, M., Bohg, J., and Schaal, S. (2013). Probabilistic Object Tracking using a Range Camera. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3195–3202.

Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. (2018). PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania.

Yang, H., Shi, J., and Carlone, L. (2020). TEASER: Fast and Certifiable Point Cloud Registration. *IEEE Transactions on Robotics*, 37(2):314–333.

Yu, K.-T. and Rodriguez, A. (2018a). Realtime State Estimation with Tactile and Visual Sensing. Application to Planar Manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7778–7785.

Yu, K.-T. and Rodriguez, A. (2018b). Realtime State Estimation with Tactile and Visual Sensing for Inserting a Suction-held Object. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1628–1635. IEEE.

Yu, Y., Si, X., Hu, C., and Zhang, J. (2019). A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural computation*, 31(7):1235–1270.

Yuan, W., Dong, S., and Adelson, E. H. (2017). GelSight: High-Resolution Robot Tactile Sensors for Estimating Geometry and Force. *Sensors*, 17.

Zhang, J., Henein, M., Mahony, R., and Ila, V. (2020). Robust Ego and Object 6-DoF Motion Estimation and Tracking. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5017–5023. IEEE.