SAPIENZA
UNIVERSITÀ DI ROMA

Computing Fast Search Heuristics for Physics-based Mobile Robot Motion Planning

Ph.D. in Engineering in Computer Science

Dottorato di Ricerca in Ingegneria Informatica – XXIX Ciclo

Candidate

Federico Ferri

ID number 1151532

Thesis Advisor

Prof. Fiora Pirri

A thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Engineering in Computer Science

September 2018

Thesis defended on 7 September 2017
in front of a Board of Examiners composed by:

Prof. Paolo Boldi (chairman)

Prof. Fabio Massimo Zanzotto

Prof. Filippo Furfaro

Computing Fast Search Heuristics for Physics-based Mobile Robot Motion Planning

Ph.D. thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: September 3, 2018

Author's email: federico.ferri.it@gmail.com

To those who inspired it
and will not read it.

Abstract

Mobile robots are increasingly being employed to assist responders in search and rescue missions. Robots have to navigate in dangerous areas such as collapsed buildings and hazardous sites, which can be inaccessible to humans. Tele-operating the robots can be stressing for the human operators, which are also overloaded with mission tasks and coordination overhead, so it is important to provide the robot with some degree of autonomy, to lighten up the task for the human operator and also to ensure robot safety.

Moving robots around requires reasoning, including interpretation of the environment, spatial reasoning, planning of actions (motion), and execution. This is particularly challenging when the environment is unstructured, and the terrain is *harsh*, i.e. not flat and cluttered with obstacles. Approaches reducing the problem to a 2D path planning problem fall short, and many of those who reason about the problem in 3D don't do it in a complete and exhaustive manner.

The approach proposed in this thesis is to use rigid body simulation to obtain a more truthful model of the reality, i.e. of the interaction between the robot and the environment. Such a simulation obeys the laws of physics, takes into account the geometry of the environment, the geometry of the robot, and any dynamic constraints that may be in place.

The physics-based motion planning approach by itself is also highly intractable due to the computational load required to perform state propagation combined with the exponential blowup of planning; additionally, there are more technical limitations that disallow us to use things such as state sampling or state steering, which are known to be effective in solving the problem in simpler domains.

The proposed solution to this problem is to compute heuristics that can bias the search towards the goal, so as to quickly converge towards the solution.

With such a model, the search space is a rich space, which can only contain states which are physically reachable by the robot, and also tells us enough information about the safety of the robot itself.

The overall result is that by using this framework the robot engineer has a simpler job of encoding the *domain knowledge* which now consists only of providing the robot geometric model plus any constraints.

Contents

List of Figures

Chapter 1

Introduction

Mobile robot navigation in harsh terrain is a challenging task. Although tracked robots are well equipped to operate in clutter, they still require a human operator to remotely teleoperate them in search and rescue operations. Improving autonomy for these robots is a pivotal issue in the search and rescue field.

In order to move autonomously and safely in the environment, a mobile robot must understand the effects its own dynamics and of its interactions with the terrain.

Figure 1.1. Mobile robots navigating through rubble in disaster areas. On the left: multiple search and rescue robots moving in a simulated disaster environment. On the right: mobile robot from TRADR project navigating amongst the remains of a collapsed church.

For example in a search-and-rescue setting, a mobile robot must go through rubble and collapsed buildings to locate survivors (see Figure 1.1).

Motion planning is the key aspect of robot locomotion which, given a map, a robot geometry, a start position and a goal position, aims to find a path for the robot which is free from collisions.

1.1 Motivation

The role of Robots in Search and Rescue environments Mobile robots are increasingly being employed to assist responders in search and rescue missions. Robots have to navigate in dangerous areas such as collapsed buildings and hazardous sites, which can be inaccessible to humans (see Figure 1.2). In the last decades, rescue robots participated in many of the most critical environmental disasters around the world, exhibiting extraordinary abilities in terms of mapping, vision and navigation. In northern Italy, where the city Mirandola was hit by a tremendous earthquake in June 2012, we deployed a team of humans and robot to assess damage to historical buildings and cultural artefacts located therein. This in-field experience has been really important because it led us to a better understanding of what are the main research challenges which are not yet widely addressed in rescue robotics.

Figure 1.2. Actual search and rescue environments scenarios.

During these missions we were able to observe how the human search and rescue operators interacted with the robots and the robot's software (see Figure 1.3), and we got an understanding of how the software platform can and should be improved.

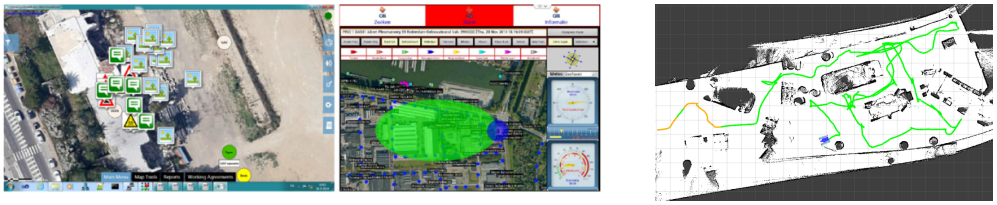

Figure 1.3. Human operators interface used to coordinate mission in TRADR EU project.

One critical issue the operators encountered concerned the robot's mobility and locomotion: the task of driving the robot around via remote controls, using only cameras and 3D LIDAR sensors proved to be very demanding for the human operators, and highly error prone. It could happen that subtle variations on the terrain surface can go unnoticed when looked at through a camera or a point cloud, can turn out to be a critical obstacle that can easily make the mobile robot stuck, and require complex maneuvers to get out of that. Motions can sometimes be fatally dangerous to the robot, for example the robot can tip over or fall into a hole. So

this task is supervised by several human operators, looking at various robot sensory data, in order to ensure maximum safety when driving the robot around.

To overcome the amount of cognitive load required to operate the robot, the operator could benefit from any computational aid that could be applied in this domain, from autonomous terrain adaptation (for articulated robots) to autonomous driving (motion planning) in known map areas, or autonomous exploration in unknown areas. Many of these approaches are already being used successfully in several search and rescue exercises. However path planning and autonomous driving is the most challenging task, and the one that could bring the greatest benefit to the search and rescue task.

Typical state-of-the-art methods use a planar representation of the environment (occupancy maps). These maps provide a binary classification of the environment into obstacle/free regions, that overly simplifies the interaction between the mobile robot and the world (see Figure 1.4).

For one reason, the classification of a region into obstacle can be dependent on context, for example a ledge that acts as an obstacle when approached from a lower level, but acts as the ground while the robot is on it. To solve this, sometimes digital elevation maps [Kwe+92] (also known as cartesian elevation maps [Dai+88; Oli+91]) are used instead of binary occupancy 2D maps.

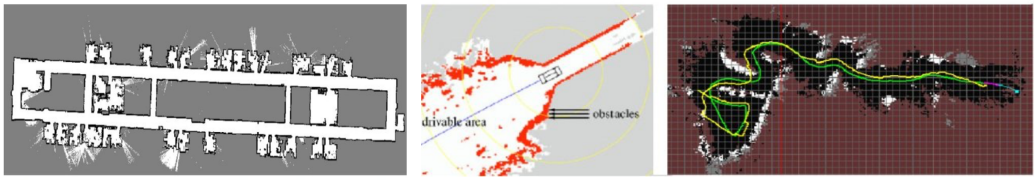

Figure 1.4. Classical occupancy maps in robotics: left: Centibots [Vin+08]; center: Grand Challenge [Mon+06]; right: Learning Applied to Ground Robots [Kon+08].

Moreover, neither 2D maps or elevation maps are able to represent environments with multiple overlapping levels, such as tunnels or multi-storey buildings, but are limited to the representative power of a Monge patch ($z = f(x, y)$).

Due to the typical terrain conditions usually encountered in this domain, maximum safety has to be guaranteed in order to ensure that a robot motion is really without any risk of losing or damaging the robot, so a 3D representation should be preferred.

1.2 Problem Statement

The general problem this thesis aims to solve is finding collision-free paths for mobile robots moving in a 3D environment, from a starting position to a goal position, with emphasis to navigation in challenging terrain.

The concept of challenging terrain can be summarized in all those types of terrains for which is not possible to reduce the motion planning problem to a simpler problem, i.e. 2D planning. Challenging terrains are typical of rescue environments and present a quantity of discontinuities and other problems so as to prevent the use of state of the art techniques to map the 3D surface to a manifold and assume that it is entirely traversable.

1.2.1 Assumptions

In order to solve the problem, some fundamental assumptions must hold, namely the availability of the 3D models of the robot and the environment.

Moreover, the environment is assumed to be static for the duration of the planning, and its model is updated only before or after a new motion planning query.

How to obtain these models, as well as how to estimate or model dynamic environments is out of the scope of this thesis.

Model of the robot The 3D model of the robot is known beforehand; it specifies the locomotion devices used such as tracks, legs, wheels, and any other joints and links the robot has to use while moving. Additional information such as mass and inertia is also required, for reasons that will be clarified in the next sections. All these specifications describe the robot in a sufficient way to allow planning the motion of the robot in advance.

Map of the environment A map of the environment in the form of a 3D model is also required.

While a 3D model of the environment is usually not available in domains such as search and rescue, it can be obtained from 3D reconstruction of point clouds (see Figure 1.5) obtained with robot mapping.

In real life scenario, an UAV is first sent to make a first assessment of the area, and that gives us a first 3D model (or point cloud) of the environment.

In lack of that, the initial map seen by the robot's laser scanner in the starting position is enough to allow the robot to move around to some extent. Typical laser

range finders have a range of 10 meters.

Vision based sensors can sometimes be unable to see all the terrain due to occlusion. Although motion planning can only plan the motion of the robot within the known parts of the environment, as the robot moves around, more map of the environment is acquired. The process of moving the robot (via motion planning) to strategic points in the known map, in order to acquire the missing parts of the map, is known as *next best view* selection, and it is not discussed in this thesis. More can be found in [Yam97; Gon+02; Sur+03; Wen+06; Str+08; Blo+11; Pot+13; Dor+13; Nik]. Instead it will be assumed that the part of the map relevant to the motion planning problem it is already available.

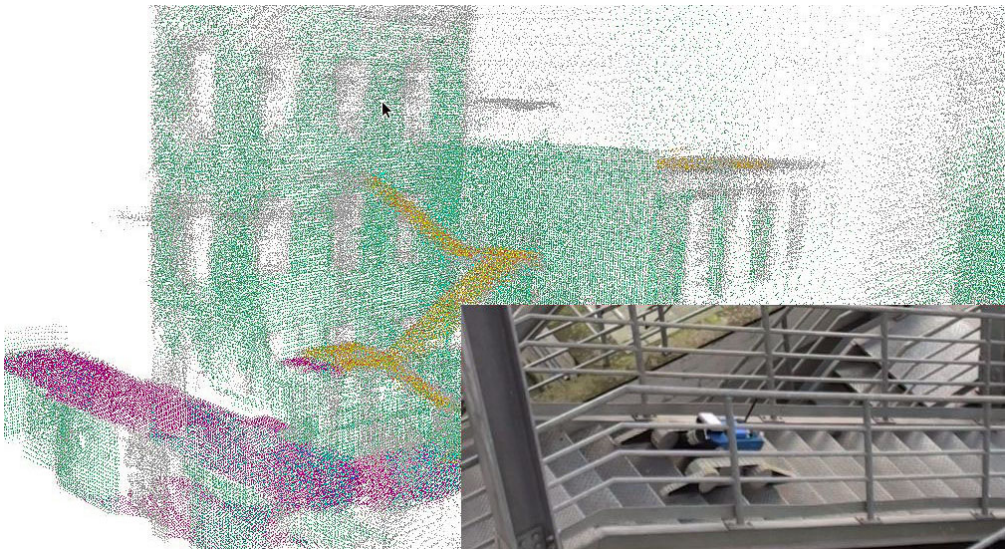

Figure 1.5. 3D map of an outdoor environment in the form of a point cloud, acquired by a robot during navigation using a tilting LIDAR sensor. in the few initial movements the robot is able to gather a significant amount of points describing its surroundings, in a range large enough to allow motion planning on it.

Static environment The environment shall be assumed to be static. Planning in dynamic environments requires the identification of dynamic obstacles, and estimation of their velocity and paths, and it has not been considered in this thesis.

The considered environments are mostly static, so this assumption makes sense in this context.

To ensure maximum safety, the robot should re-sense the environment as it goes, update the map accordingly, and possibly replan if the map has changed. For this purpose, a method for maintaining a map of a dynamic environment is presented in Appendix A.

1.3 Challenges in Motion Planning for Mobile Robots in Harsh Terrain

Providing a reliable mathematical model of the interaction between the robot and the terrain is an extremely difficult task. Finding a path for the robot motion which satisfies all the required geometric and dynamic constraints (which include heading, linear and angular velocities and curvatures) requires solving a system of nonlinear differential equations.

A good kinematic model should accurately predict, basing on the neighboring terrain morphology, if the robot will be capable of driving there and where it will arrive if applying certain control inputs.

Conversely, a bad model will not find all trajectories the real robot is capable of driving over, or worse, if can find a trajectory which is unfeasible to the robot, and it will cause some damage, or getting the robot stuck in some obstacle.

Doing motion planning in such complex environments goes beyond traditional motion planning: the terrain is not flat and the interaction between the robot wheels or tracks with the terrain is difficult to model analytically. While it has been shown that is possible to approximate the terrain with simpler geometries [ORo+79; Che99] or monge patches [Gra97; Vas+10], the resulting equations of motion of the robot are usually very complicated and specific to a robot model.

Moreover the robot can be underactuated or have a variety of kinematic and dynamic constraints that put even more restrictions on the computation of a solution; for example the most obvious constraint is being subject to gravity force, which in turn requires the robot to be in a static equilibrium pose, with at least three points of stable contact with the ground.

In the early years of motion planning [Cho05; LaV06], the dynamics aspect of the problem was not taken into account. Instead, only the geometric aspect of the problem, that is, the robot and the obstacles has been considered, such as the piano movers problem [Sch+83a; Sch+83b; Sch+83c]. Those early approaches focused on the explicit construction of the configuration space, which led to proving that complete algorithms are PSPACE-complete and exponential in time [Rei79; Sch+83a; Bro+85; Sch+88; Can88; Pla+10].

When considering also the dynamics of the problem, we usually talk about kinodynamic planning, whose goal is to synthesize robot motion subject to simultaneous kinematic constraints (such as obstacle avoidance, joint limits), and dynamics constraints (such as bounds on the velocity, acceleration and force/torque).

Much of the recent progress in motion planning is attributed to the development of sampling-based motion planning algorithms [LaV06; Şuc+08; Şuc+09]. Those

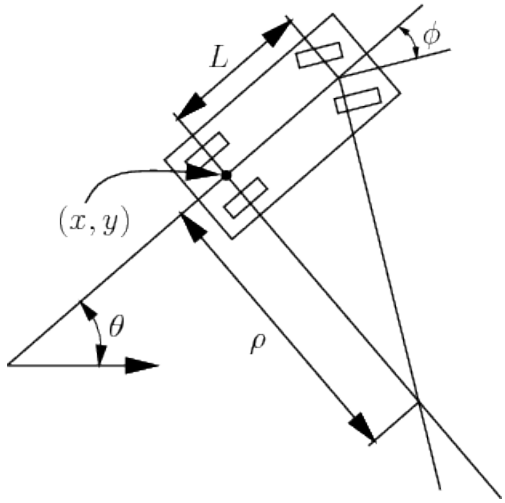

Figure 1.6. A geometrical representation of the "Dubins Car", a very simple model of a four wheeled steering car that can move along Dubins curves [Dub57], and one of the first kinematic models used in motion planning for mobile robots.

algorithms have weaker guarantees (they are probabilistically complete, which means that if a solution exists, it will be eventually found; on the contrary, if a solution does not exist, this cannot be reported). The first algorithms of this kind were the Probabilistic Roadmap Method (PRM) [Kav+96], Rapidly-exploring Random Trees (RRT) [LaV98] and Expansive Space Trees (EST) [Phi+04]. These planners build a tree of motions in the state space of the robot and attempt to reach the goal state.

These algorithms are better suited for planning the motion of robot arms or free flying bodies; however applying the same methodology to mobile robots poses additional challenges. For example, *sampling*-based motion planning cannot be applied to a underactuated robot, such as a mobile robot, with the same ease it can be applied to a robot arm. In the case of a robot arm, state sampling and validation simply accounts to selecting a state from its state space which is not in collision with the environment. This does not work with mobile robots, which have to touch the ground in a stable pose, but not collide with the environment in other ways.

Sampling a state in this setting is very expensive. Consider building the configuration space $\mathcal{C}$ as usual via collision checking of the robot with the environment. The free subset of the configuration space $\mathcal{C}_{\text{free}}$ is not enough to describe a valid pose: a pose where the robot is free-flying in mid air is not in collision, hence it belongs to $\mathcal{C}_{\text{free}}$, yet it is not valid, because it is not touching the ground in a stable pose. We must take into account the effect of gravity as well. Building the subset of $\mathcal{C}_{\text{free}}$ of valid poses is expensive because it requires running physics simulation (or an approximation of it) in addition to performing collision checking. Worse, this type of sampling, cannot be done lazily and/or sparsely, because the fact that two poses are

valid does not mean that we can blindly interpolate between the two and find a safe (collision-free, stable) motion. Under these considerations, seems more practical to run a physics (rigid body) simulation to drive the robot around and observe which states can be reached by operating on the space of the control inputs. The logic of driving the robot around, that is, searching for a path to the goal state, is the same as in traditional sampling based motion planning algorithms (such as RRT), but state sampling has been replaced with control input sampling. Physics simulation is used to expand the search tree, by applying the sampled control input from a state to obtain a resulting state, together with the resulting motion. This step is known as *state propagation*.

This approach has the advantage that does not require to model any equation of motion or any equation or approximation of the terrain but instead requires only to give a geometric and physical description of the robot (i.e. the 3D model, the joints, the mass and the inertia of each link, the motor torques) and of the environment (terrain, obstacles), letting the physics simulation engine to compute the interaction between the robot and the environment. However, also some limitations exist, such as: the impossibility of doing backwards simulation; the high computational cost of sampling in the state space; the lack of a method for connecting states in the state space (also known as state steering). These limitations make the use of sampling-based much less effective. Some new motion planners exist that are designed specifically for handling the above mentioned limitations in system with complex dynamics, such as Kinodynamic Planning by Interior-Exterior Cell Exploration (KPIECE) [Şuc+08]. However, since sampling based algorithms spend most of their time extending the tree, and this step is done using physics simulation, which is much computationally intensive than integrating equations of motion, this approach lacks the computational efficiency for real-time planning.

1.4 Challenges in using a Rigid-Body Simulation as State Propagation

Physics simulation allows us to simulate complex real physics systems and thus to predict the behavior of a dynamic system. There are two classes of physics simulation, or physics engines: real-time engines and high-precision engines. High-precision engines are more demanding in terms of CPU time, while real-time physics engines allow to compute simulations in real-time, sacrificing some accuracy.

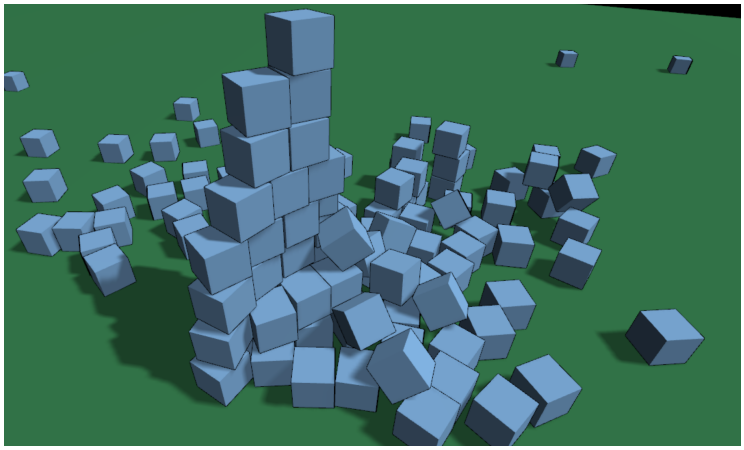

Figure 1.7. A screen snapshot of a realtime physics simulator performing a simulation of many cuboidal bodies. The simulator has to take into account gravity force, and thousands of interaction forces caused by each body colliding with several other bodies

The physics simulation approach is useful when we want to describe the motion of complex systems, which would be too complicate to describe with equations of motions. One example of this is the motion of a mobile robot in uneven terrains: the interaction between the robot's wheels or tracks and the shape of the terrain is in general too difficult to model analytically while maintaining a good level of approximation of the real terrain morphology.

This plays a central role in a motion planner because it is the part which expands the tree of states and it has to produce motions which match reality. This step is also known as state propagation. Using a physics simulation to compute the state propagation of the robot has its own appeal: it is a declarative approach, i.e. it states what needs to be solved rather than saying how to solve it. Instead of figuring out the equations of motion for a particular robot in a certain approximation model of the terrain, we simply plug into the system the geometric/kinematic description of the robot (3D CAD model, information about joints, actuators, masses and inertia).

However the accuracy of the simulation depends on many factors. A physics simulation is subject to small errors as well, in part due to limits in numerical

precision, and in part due to the fact that in order to design an efficient simulation, we want to describe the robot with primitive shapes, such as boxes, spheres, cylinders and capsules. Anyway none of these errors pose a problem, since the relevant elements of a robot are already primitive shapes (such as cylinders for wheels).

During a physics rigid body simulation, there are two stages that are run continuously: 1) integrating Newton's equations of motion 2) checking collisions between bodies and computing resulting forces. These stages are governed by some internals parameters of the simulator (which will be covered in the next chapters). The setting of these parameters can have impact on the accuracy and the degree of realism of the simulation: an incorrect setting of these parameters can even lead to unstable simulations, which are completely useless to our purposes. It is important that the same control input produces a similar effect both in simulation and in reality. To achieve this we perform a calibration so to tune these parameters.

Choosing the *physics world* as the model, we encounter unavoidable limitations, such as the impossibility of doing backwards simulation, or the inherent property of the system of not being holonomic nor steerable. Many planners rely on state steering functions, or local planners, to connect pairs of configurations. This step would be too computationally inefficient to compute when the underlying model is the physics world, and with a complexity similar to the global planning problem itself.

The other inevitable problem of using a simulation as state propagation, is that is computationally intensive. We might be able to run it faster than realtime, but not with great speedups. And a motion planning algorithm spends most of its CPU time in expanding the tree, so this becomes the bottleneck of the system: we want to waste the least possible time in performing simulation.

In the next section we will se how to avoid wasting CPU cycles expanding regions of the search space which are not useful in finding a valid motion from start to goal, by exploiting empirical knowledge about mobile robot locomotion, in particular by using a navigation function (see Figure 1.8) to prioritize most promising motions.

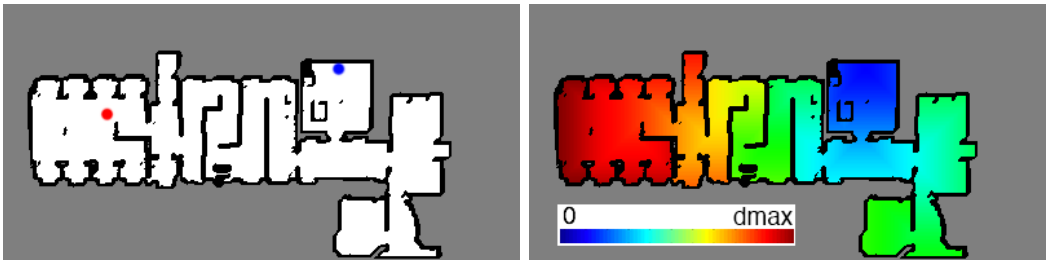

Figure 1.8. Example of a navigation function. Left: a planar map, with start and goal locations marked respectively in red and blue; right: navigation function, color-mapped, computed for the goal location. Note that the function is defined only for locations which are connected to the goal location.

1.5 Challenges in Physics Modeling of Mobile Robots

Figure 1.9. Some types of mobile robots used for search and rescue, planetary exploration, and other similar tasks.

Mobile robots have various type of traction and locomotion devices (see Figure 1.9), depending on the performance requirements and terramechanics (wheel—soil interaction mechanics). As an extension of the conventional terramechanics theory for vehicles, the terramechanics theory for mobile robots, which is becoming a new research hotspot, is unique and puts forward many new challenging problems.

Currently, the most common robot traction types are wheels and tracks with grousers.

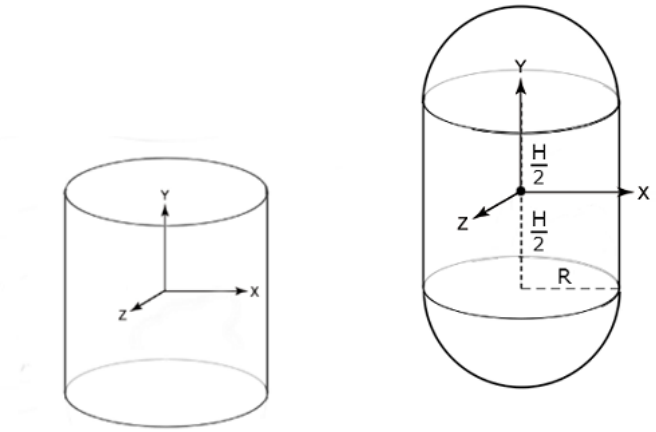

Figure 1.10. Left: a cylinder geometry. Right: a capsule geometry.

In case of a wheeled robot, no particular geometric design considerations are required: wheels can be simulated by using cylinder or capsule geometrical shapes (see Figure 1.10). However, it is very important to provide realistic parameters about friction.

On the other hand, there is no standard method of simulating those.

It is possible to create many finite elements, as shown in Figure 1.11, one for each grouser, connected with hinge joints (see chapter 4 and Figure 4.1), and leave

to the simulation engine the job of computing collisions and response forces (see subsection 4.3.3). Moreover, as it is the case with reality, some method for keeping the track centered must be used. This approach can give a very realistic result, but at the same time has a big computational cost.

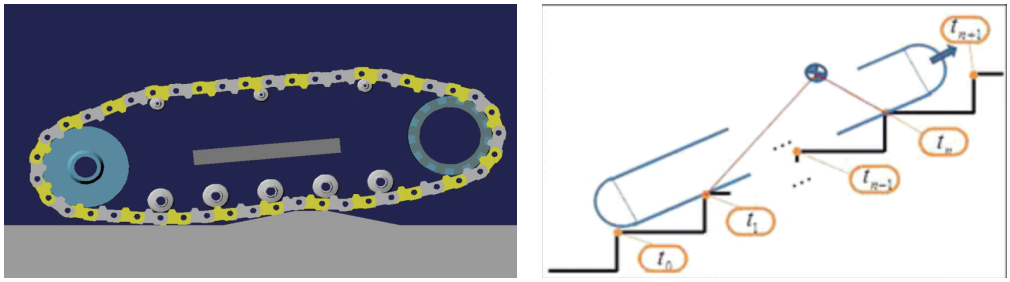

Figure 1.11. Modelling a track via a chain of rigid geometries (left) versus a single rigid geometry and contact joints (right).

In order to simulate traction in a cheaper way, another method is to just compute the contact points between the grouser geometry and the terrain mesh (Figure 1.11), and apply a force to those points. This method is an approximation, and does not take into account deformation of the track, neither the slippage of track versus the terrain.

In chapter 6 we see more in detail the aspects of tracked vehicle simulation.

1.6 Contribution and thesis structure

This thesis studies motion planning for mobile robots in harsh terrain and complex environments. The key aspect taken into consideration in this thesis is to use 3D models for the environment and for the robot, instead of projecting everything on a plane. To make an accurate analysis of navigation in such 3D terrains, a motion planning framework using physics simulation as state propagation is used. The principal contribution is a novel heuristic path-planning strategy that can be used as heuristic function for a physics based motion planner. The result obtained is better planning time compared to existing physics based motion planning approaches.

The structure the thesis is the following: a comprehensive literature survey of motion planning and related fields is given in chapter 2; then the preliminaries about motion planning are discussed in chapter 3, and the preliminaries about physics simulation, among with issues that may arise when designing a physics simulation for a mobile robot, are discussed in chapter 4; the physics-based motion planning framework is described in chapter 5; the computation of heuristic functions for the physics-based motion planning framework is described in chapter 7; conclusions are given in chapter 8. A method for updating the map in dynamic environments is presented in Appendix A.

Chapter 2

Literature Review

When the problem of mobile robot motion planning is studied from a theoretical point of view, certain simplifying assumptions are made; the most important simplification made is to assume a two-dimensional workspace. This is very convenient, as the typical mobile robot still has only three degrees of freedom (one rotational and two for translation) to move over the terrain, so this choice allows to compress the representation of robot motion into curves on a plane. This simplification is also very effective in certain environments, such as indoor, planar or structured environments, and to some extent also to moderately non-planar environments, such as urban roads. [Lat91] provides a description of these approaches commonly used in motion planning, where for the case of mobile robots the environment is always considered 2D. Since the scenario taken into consideration in this work is more general, and not limited only to indoor and structured environments, the comparison should be focused more towards approaches using a three-dimensional representation of the environment. Also, the presence of obstacles is commonly represented with a (discretized) occupancy map, for example in [Vin+08; Mon+06; Kon+08], these maps feature a binary classification of the environment into traversdable (ground) and non-traversable (walls, obstacles) regions. According to the premises explained in the introduction, a planar representation of the environment is not suitable to properly represent the terrain topology in the general case; additionally, the choice of a binary traversability map overly simplifies the interaction between the mobile robot and the world, especially when the terrain is not flat or does not follow some well known patterns, such as staircases, ramps, steps and small gaps which can be overcome with some predefined motion strategies. To overcome these problems, Semantic Mapping [Nüc+08; Ten+10] is used to share knowledge about navigation between humans and robots, especially in indoor environments [Nüc+08; Rus+08; Gun+13; Her+14] and specifically in mobile robot navigation for search and rescue [Fer+14] where terrain is classified in various semantic classes such as flat, ramp, stairs. However this approach requires knowledge about the various classes of terrain.

Even when the terrain semantic features are correctly identified, there must exist some specific strategy to interact with that part of the environment. This is not always the case [Rui+17].

While indoor environments present more standardized features ranging over a discrete number of classes, the environments taken into consideration in this work have more variable features, so a traversability analysis is to be preferred instead of semantic mapping. A comparison with existing traversability analysis approaches is given in the next section. The main difference between existing works is that here the traversability analysis is used to compute the navigation function, and does not directly affect how motions are selected. A hybrid approach which combines semantic mapping and traversability analysis can be found in [Mor+09], where 2D planning techniques are adapted to plan in 3D challenging terrain. Their motion planning subsystem works on a 3D map of the environment, and makes a grid-based decomposition of that into regions that are locally 2D. Depending on the type of region (flat, stairs, slopes) it uses a specific 2D planner that can generate a suitable motion for the specified terrain, such as restricting robot heading to face up or down when traversing stairs and slopes; while providing an improvement over semantic labeling, it is still based on strategies, and the underlying map used by the motion planner i only 2.5D so not comparable with this work.

The peculiar aspect of this line of research is to use geometric-exact motion planning. The first general motion planning problem was formulated in [Sha+84], where a rigid body must be moved from start to goal, using six degrees of freedom, while avoiding collisions. Since these problems have a high number of dimensions, random sampling algorithms are commonly used for solving this type of problems. Works such as [Bar+97] and [LaV98; LaV+00] are a reference point in the motion planning research. For a survey about motion planning algorithms, see also [Sch+88], [LaV11a] and [LaV11b].

Some works use other ways to speed-up the search, such as reducing the number of re-visited nodes during backtracking [ElH+13].

[Jos+05] makes a particular use of simulation which consists of finding "high-level" plan in a space of "strategies" using a genetic algorithm, and then using a simulator to validate it. In our opinion, searching in the space of strategies is too detached from reality and not reliable enough, as it can sometimes fail to see where strategies or motions lead to a damage of the vehicle itself, due to the uncertainty that exists between the idea of an action and its execution, therefore we consider this approach unsuitable according to the requirements of a search and rescue mission.

While RRT itself is capable of kinodynamic motion planning [LaV+99], the problem of using a physics engine "in the loop" has not been considered until recent years.

The use of a physics simulator in "the loop" has probably been pioneered by [Kav+05]

where control-based motion planning is used instead of geometric planning, and a propagation function is computed by a game physics simulation engine. Due to the computational load of approaching the problem in this way, they create a search algorithm to explore the state space which estimates the coverage using a non-uniform subdivision of the state space, which provides significant improvements in regards of finding the solution in less time. It shares similarity with the work presented in this thesis, as the physics simulation based motion planning is used here too, but a different approach for speeding up the search is used.

In [Şuc+09; Kav+12] a multi-level grid discretization of the state space is used to estimate the coverage. The coverage estimation helps the planner detect the less explored areas of the state space. While oriented for general, domain independent, kinodynamic motion planning, and thus not directly attacking the problem of mobile robot motion planning, their algorithm provides a remarkable improvement in terms of performance compared to others. However, better performance is obtained with domain specific heuristics such as the one presented in this thesis.

Finally, see [Lin+05] for a discussion about issues in sampling-based motion planning in problems with many degrees of freedom, including uniform and regular sampling, topological issues, and search philosophies.

3D Path Planning

This section provides a literature review of works doing path planning for mobile robots in 3D while not specifically performing kinodynamic planning, doing collision checking, or using a simulation as the state propagation model.

In [Sim+93] the authors formalize several constraints, such as wheel-ground contact, suspensions stretch limits, no tip-over, and no collision with terrain, but use an iterative approach to compute the pose of the robot in a particular patch of 3D terrain (collision-free placement) and then compute feasible trajectories using a numeric method under certain assumptions.

In [Che+94] and also in In [Che99] the authors take into account the kinematic and dynamic constraints of the robot, and integrate geometric and physics models of the robot and the terrain (modelled with viscous-elastic laws and surface-to-surface dry friction interaction) in a two-stage trajectory planning algorithm, which combines a "discrete search strategy" with a "continuous motion generation".

[Far+98] use an analytical physics dynamic model based on the geometry of the robot and local approximations of the terrain.

[Car+06] uses an interpolation-based planning and replanning which builds upon 3D grids, based on the assumption that moving in certain directions is more difficult

than moving in some other directions (like upwards). Other works using D* for mobile robot planning include [Ste95], [Koe+02] and [Fer+05].

[Kla+12] uses a 3D representation of the environment based on surfels, then extract 2D maps for path planning.

[Col+13] uses D*-Lite to plan for the position of the robot, ensuring that there is a feasible orientation, and considering the 26 neighbours given by cardinal directions plus all the diagonals in 3D, while using some heuristic feasibility criterion for cell validity, such as the robot not being in collision, there is a surface to support the robot, and the surface normal do not cause the robot to tip-over.

Traversability

The majority of traversability analysis methodologies takes into consideration geometric features of the terrain. [Lan+94] divides the map into cells, and then filters out cells based on maximum slope, elevation discontinuity and maximum elevation difference. In [Gen99] terrain traversability is computed by a cost function which aggregates elevation, slope and roughness. The work of [Lan+94] and [Gen99] set the core ideas for geometry-based traversability analysis.

3D Semantic mapping has been studied in [Nuc+03] where a LIDAR has been used to create a map in which semantic labels are added in order to identify architectural elements via geometric features. This type of semantic mapping works well with structured environments, such as indoor environments, and urban areas to some extent; however it doesn't provide help when the terrain is harsh. In [Lal+06] semantic mapping is extended to outdoor environments including vegetation covered areas such as grass, bushes and tree canopy, using broader classes such as "surface", "linear" and "scatter". Although semantic labels are used for the identification of terrain, local point statistics are then used to determine traversability. [Tal+02] uses a 3D obstacle detection algorithm based on segmentation, average and maximum slope, relative height, and texture-based classification.

Research on 3D mapping for mobile robots using laser range finder has been also explored in [Bib+04], whose representation does not make use of any additional semantic label and has not proved to work in real time with a fully autonomous robot system. In [Die+06] the only semantic label used is the "wall" label which is then used to reduce the 3D map to a 2D map, likewise [Col+13], since it is possible to show that stairs are 2D reducible and flipper angles can be set in advance.

[Sin+00] uses plane fitting over range data to compute an estimation of the expected roll and pitch. [Gen99] takes also into account point accuracy and sparseness in addition to usual point elevation statistics.

Chapter 3

Preliminaries: Motion Planning

Given the model of a robot, and the model of the environment, the motion planning task is to find a collision-free motion that moves the robot from a starting pose to a goal pose, without hitting obstacles in the environment, and satisfying all the robot constraints that arise from robot's kinematics, joints, dynamics, or any other constraints imposed by design on the robot itself. The problem was originally formulated as the Piano Mover's Problem, and the computational complexity of it has been shown to be PSPACE-hard for polyhedral robot and polyhedral environment [Rei79].

As it usually happens with search algorithms, the definition of the search space is key to solving the problem. In particular for motion planning, a point in the search space (which can be referred to also as the configuration space, or the state space depending on the context) must completely describe every point of the robot geometry.

Configuration Space Building on the idea that the robot is completely described by a point in the search space, the concept of *configuration space* arised and has been widely adopted [Udu77; Loz83]. The configuration space ($\mathbf{q} \in \mathcal{C}$) is the space of all possible transformations (if applicable) and joint configurations (if any, in case of articulated robot) that can be applied to the robot. The number of degrees of freedom of the robot is usually equivalent to the dimension of the configuration space, or at least that puts a lower bound to the dimensionality of the configuration space (see Figure 3.1).

The configuration space gets partitioned into *free space* and *obstacle region.*. Let $\mathcal{W}$ be the total workspace. For 3D motion planning we have that $\mathcal{W} = \mathbb{R}^3$. Let

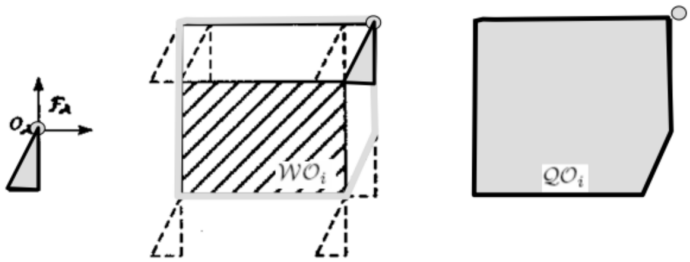

Figure 3.1. On the left: a triangle-shaped robot that can only translate in a 2D workspace. The robot's reference point is its point. Center: the workspace with one obstacle ($\mathcal{W}\mathcal{O}_i$). On the right: the obstacle mapped to the configuration space. The configuration space has size 2 because the two degrees of freedom of the robot. In the configuration space the robot is mapped to a point: if the ploint lies in the free space, then the robot is not in collision with the workspace, and vice-versa.

$\mathcal{A}(\mathbf{q}) \subset \mathcal{W}$ the set of points occupied by the robot in the configuration $\mathbf{q}$. A subset of the workspace $\mathcal{O} \subset \mathcal{W}$ represents the *obstacle region*, i.e. all the parts of the environment, which the robot must not collide with (i.e. $\mathcal{A}(\mathbf{q}) \cap \mathcal{O}$ must always be empty)[1]. The free space region can now be defined as $\mathcal{C}_{\text{free}} = \{\mathbf{q} \in \mathcal{C} \mid \mathcal{A}(\mathbf{q}) \cap \mathcal{O} = \emptyset\}$, and thus the *obstacle region* is defined by it's complement $\mathcal{C}_{\text{obs}} = \mathcal{C} \setminus \mathcal{C}_{\text{free}}$.

A continuous path in $\mathcal{C}_{\text{free}}$ from a start configuration to a goal configuration represents a solution to the motion planning problem. However, the computation of $\mathcal{C}_{\text{obs}}$ and $\mathcal{C}_{\text{free}}$ is not easy, also due to the fact that the dimensionality of $\mathcal{C}$ is often quite high.Instead of completely computing $\mathcal{C}_{\text{free}}$, sampling-based planners are used. In comparison to complete and exact algorithms for motion planning, sampling-based algorithm provide a weaker guarantee because they are probabilistically complete, meaning that a solution will be eventually found if enough iterations of the algorithm are run. Sampling-based algorithm however have been shown to solve problems that seemed impossible to tackle with complete algorithms. The typical principle of operation is the following: configurations are sampled from $\mathcal{C}$ and validated one by one by the collision checker, with the invalid ones being discarded; this has the same effect as sampling from $\mathcal{C}_{\text{free}}$ but without the explicit computation of $\mathcal{C}_{\text{free}}$; the samples are connected to other samples including start and/or goal configuration, by 1D curves in $\mathcal{C}_{\text{free}}$, provided those curves exists and are entirely in $\mathcal{C}_{\text{free}}$, effectively building a graph of configurations; when a path from start configuration to goal configuration exists, that is reported as the solution to the motion planning problem (Figure 3.2).

The notion of configuration space is useful in domains where *any* path entirely contained in $\mathcal{C}_{\text{free}}$ is valid. The case of robot arm is a typical example, where every

[1]These objects are usually expressed as a collection of polyhedra, 3D meshes, or algebraic surfaces

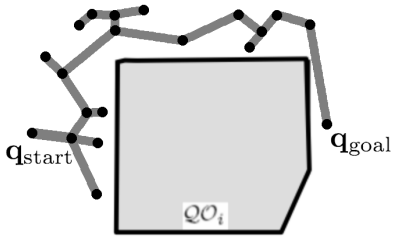

Figure 3.2. Solution to a motion planning problem found with a sampling-based planner. The path connecting $\mathbf{q}_{\text{start}}$ to $\mathbf{q}_{\text{goal}}$ is entirely in $\mathcal{C}_{\text{free}}$.

random state $\mathbf{q}$ can be considered valid as long as $\mathcal{A}(\mathbf{q}) \cap \mathcal{O}$ is empty. Instead, for the case of a mobile robot driving in 3D environments, that condition alone is not sufficient anymore, for example of the state where the robot is floating mid-air doesn't have intersection with the workspace, however it is not a reachable state, and so also any edge connecting that state to any other state should not exist.

Kinodynamic motion planning When it is not possible to sample from the configuration space, a *state propagation function* is introduced, together with a *control sampler* that replaces the former type of sampling from $\mathcal{C}_{\text{free}}$. Starting from an initial state, a tree of motions is built by application of the state propagation function, until a state satisfying the goal condition is reached.

Similarly, when additional bounds about velocity, force or torque or other type of kinematic constraints must be satisfied, we are talking about *kinodynamic* motion planning (term coined by Bruce Donald, Pat Xavier, John Canny and John Reif [Don+93]).

Sampling-based tree planners such as Rapidly-exploring Random Trees (RRT) [LaV98; LaV+00], Expansive Space Trees (EST) [Hsu+97] have been successfully used to solve such problems.

Using a physics simulation as state propagation falls into this category as well. In this scenario, only a forward propagation routine is available (that is, the simulation of the system can be done only forward in time), and no state sampling is used (because it would be too complex and expensive to compute). Planners such as Path-Directed Subdivision Tree (PDST) [Kav+05] and Kinodynamic Planning by Interior-Exterior Cell Exploration (KPIECE) [Şuc+09] have been designed specifically for these scenarios.

Definition A (kinodynamic) motion planning problem is defined as follows.

- A *state space* $\mathcal{Q}$.

- A *control space* $\mathcal{U}$.
- An *initial state* $\mathbf{q}_i \in \mathcal{Q}$.
- A set $\mathcal{Q}_F \subset \mathcal{Q}$ of *final states*.
- A *forward propagation* (or *state propagation*) function $f : \mathcal{Q} \times \mathcal{U} \mapsto \mathrm{Tg}\mathcal{Q}$ where $\mathrm{Tg}\mathcal{Q}$ is the tangent space of $\mathcal{Q}$ (f does not need to be explicit).
- A *state validation* predicate v, to tell if the state is valid or not.

A solution to a motion planning problem instance consists of a sequence of controls $u_1, u_2, \ldots, u_n \in \mathcal{U}$ and times $t_1, t_2, \ldots, t_n \in \mathbb{R}^{\geq 0}$ such that $\mathbf{q}_0 \in \mathcal{Q}_I$, $\mathbf{q}_n \in \mathcal{Q}_F$, $\mathbf{q}_k$ $(k = 1, \ldots, n)$ can be obtained sequentially by integration of f, and $v(\mathbf{q})$ holds for $\mathbf{q} = \mathbf{q}_1, \ldots, \mathbf{q}_n$.

In this work, the function f is computed by a physics simulator. Instead of equations of motion to be integrated, a model of a robot and its environment needs to be specified. Although simulation incurs more computational costs than simple integration, the benefits outweigh the costs: increased accuracy is available since physics simulators take into account more dynamic properties of the robot (such as gravity, friction) and constructing models of systems is easier and less error prone than deriving equations of motion. Limited numerical precision will still be a problem regardless of how f is computed. However, as robotic systems become more complex, physics-based simulation becomes a necessity.

Chapter 4

Preliminaries: Rigid Body Simulation

Rigid body dynamics studies the motion of bodies under the action of external forces. In dynamic models of mechanical systems, a popular and useful idealization of a solid object is as a *rigid body*, where macroscopic deformations occurring in materials like cloth or rubber are not allowed. The world of ideal rigid bodies has clear and well defined rules of how objects move under the action of forces and torques, as well as how constraints like rolling, sliding or pivoting affect the motions of systems of objects. Different rigid bodies may be connected by joints to form articulated bodies. Friction is modeled according to the Coulomb law; restitution is modeled using a single coefficient. The latter provides a crude way of accounting for microscopic deformation during a collision. Despite these simplifications, the predictive power is quite good for many applications.

Game physics engines like ODE [Smia] have been developed for gaming and virtual worlds with an emphasis on real-time performance and efficient handling of contact.

The rest of this chapter explains general concepts of physics engines and some aspects particular to ODE and is adapted from [Smib].

4.1 General concepts

Simulating the motion of a rigid body is almost the same as simulating the motion of a particle [Bar01]

A rigid body is characterized by several properties. Some properties are either constant over time, or for the purposes of the simulation, are assumed to not change over time. Some of the properties constant over time are:

- Mass of the body.
- Center of mass of the body with respect to the reference point.
- Inertia matrix, which describes how the body's mass is distributed around the center of mass.

There are also properties which change over time, such as:

- Position vector $\mathbf{p} = (x, y, z)$ of the point of reference of the body.
- Linear velocity vector $\dot{\mathbf{p}}$ of the point of reference.
- Orientation matrix $\mathbf{R}$ (or equivalently, a quaternion).
- Angular velocity vector $\omega = (\omega_x, \omega_y, \omega_z)$ which describes how the orientation changes over time.

4.2 Integration

The simulation of rigid bodies happens via integration of equations of motion. At each timestep, each body's position, orientation and velocities are adjusted according to the forces acting on that body.

The main issue with integration is to consider how accurate an integrator is, i.e. how closely the simulation would match the real-life object's behavior, and how stable it is, that is wether the calculation errors will lead to a non-physical behavior of the body.

The most common integration methods are:

- Euler Method [Eul68]
- Verlet Method [Ver67]
- Runge-Kutta Method [Run95; Kut01]

The Euler Method is a first order integration scheme, i.e. the total error is proportional to the step size. However it can be numerically unstable, regardless of how small the step size is, or wether the system is linear or not. Despite being easy to implement, is not used due to these problems.

The Verlet Method is a symplectic method, and it is good at simulating systems with energy conservation. It is very stable, but not particularly accurate unless the step size is small. For most uses of physics simulation this is not a problem.

The Runge-Kutta method may be more accurate in certain situations, but it is more expensive to compute and more complex to implement.

The OpenDE library [Smia] uses the Verlet Method.

4.3 Collisions

At each timestep the simulator has to also check for collisions, in order to prevent compenetration of rigid bodies.

Since collision detection can be time-consuming, it is split in two parts: a Broad-Phase and a Narrow-Phase. The reason for this is to accelerate computation: the Broad-Phase collision detection is fast to compute, and can reduce the number of computations that need to be computed by the Narrow-Phase collision detection.

4.3.1 Broad-Phase Collision Detection

Each object geometrical shape is wrapped with a bounding volume. The most common bounding volume types are:

- spheres;
- axis-aligned bounding boxes (AABB)
- oriented bounding boxes (OBB)
- convex hulls.

Spheres provide the simplest description of bounding volume, but on the other hand provide a very loose bound, meaning that a bounding sphere occupies more empty volume, and hence can generate false positives in the Broad-Phase collision detection.

A better volume bound is given by axis-aligned bounding boxes, which are also very easy to check for intersection.

Oriented bounding boxes and convex hulls provide even better volume bounds, but at the cost of more complex algorithm for intersection test, which would be slower in terms of computation time.

The physics engine creates a Boundary Volume Hierarchy (BVH), which is a tree-like structure where each node contains objects that are most likely to collide.

The physics engine tests each node and creates a list of potential collision pairs. The Broad-Phase is fast, and it may report false positives collisions. But even so, it removes all non-possible collision pairs. False positives pairs are not a problem, because the Narrow-Phase collision detection will properly check those pairs and discard pairs which are not in collision.

4.3.2 Narrow-Phase Collision Detection

The collision-pair list is then passed down to the Narrow-Phase Collision Detection. In this phase every pair of objects which are potentially intersecting is analyzed by a specific algorithm. There are algorithms for computing collisions for every possible pair of shapes (cuboid, sphere, cylinder, plane, convex mesh, non-convex mesh). Collision checking for primitive shape pairs (e.g. cuboid-cuboid, cuboid-sphere, sphere-plane) is very fast. For convex meshes the Gilbert-Johnson-Keerthi (GJK) algorithm [Gil+88] is used. This algorithm is precise but expensive.

The result of the Narrow-Phase collision detection is a set of collision points, together with the contact normal, and the penetration depth.

4.3.3 Collision Response

After the collision points and normals are computed, a finite impulse consisting of linear and angular velocities must be generated, so that the two colliding objects will move away from each other, in a physically plausible manner.

This calculation requires information such as mass, a coefficient of restitution, collision points, and collision-normal vector.

At the time of collision, the most significant force acting on the objects is the collision force (or impact force), so all other forces are ignored for that instant. The impact force is high in magnitude but short in duration.

After a collision, the impact force dies out, and external forces act on the object once again. The equation of motion is solved, providing a new position and velocity. The physics engine performs this process continuously, and it creates the illusion that an object is falling due to gravity.

The purpose of a physics engine is to solve the equation of motion and detect collisions. The goal is simple to understand yet complicated to implement.

4.4 Joints and Constraints

A joint is a constraint enforced between two bodies, so that one body can have only certain configurations (position and orientation) with respect to the other body.

For example a hinge (or revolute) joint would take away two rotational degrees of freedom, allowing one body to rotate only around one axis. A piston (or prismatic) joint would force two bodies to have the same orientation. See also Figure 4.1.

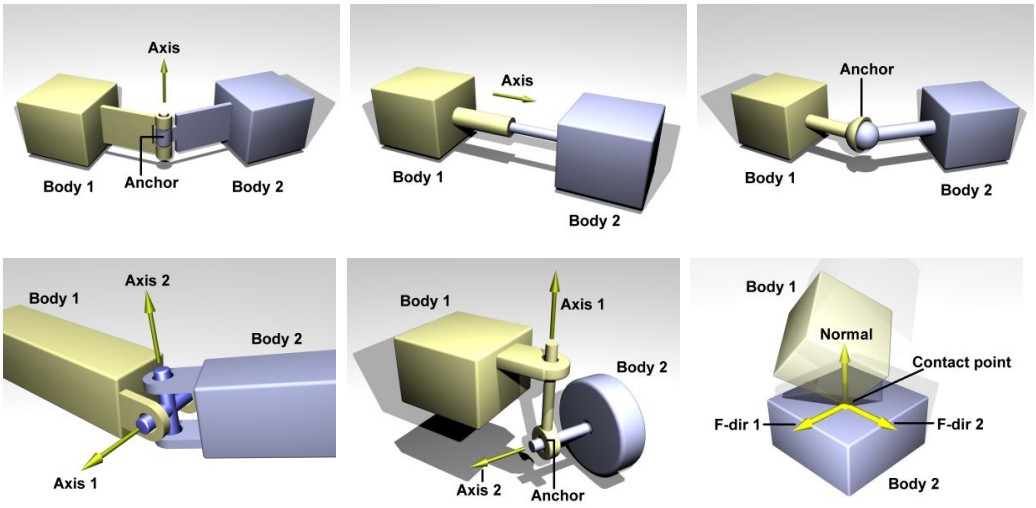

Figure 4.1. Some examples of joints. From left to right, top to bottom: a hinge joint (also known as revolute joint); a piston join (also known as prismatic joint); a ball and socket joint (also known as spherical joint); a universal joint; a double hinge joint; a contact joint. Image courtesy of [Smia].

Usually a constraint is not applied *analitically* in one shot, as that would cause instabilities in the simulations. Instead, constraints zero their error iteratively: at each integration step, the joints are allowed to apply constraint forces to the bodies they affect, in order to keep the constraint satisfied. The modulo of that forces is controlled by the ERP parameter, described in the next section.

4.5 Joint error and the error reduction parameter (K_{ERP})

When two bodies are attached via a joint, they usually maintain certain positions and orientations relative to each other. However it may happen that the the position and orientation are wrong. This type of joint error may be caused by:

- the user setting the position or orientation of one body without correctly placing of the other body;
- during simulation, numerical errors, integration errors, or errors caused by collision response, may push the bodies away from their correct positions.

In order to reduce the joint error, a special force is applied to bring bodies back into correct alignment. The error reduction parameter (K_{ERP}) controls how much this action is taken at each timestep. The K_{ERP} specifies what proportion of the joint error will be fixed during the next simulation step. If $K_{\text{ERP}} = 0$ then no correcting force is applied and the bodies will eventually drift apart as the simulation proceeds. If $K_{\text{ERP}} = 1$ then the simulation will attempt to fix all joint error during the next time step. However, setting $K_{\text{ERP}} = 1$ is not recommended, as the joint error will not be completely fixed due to various internal approximations. [Smib].

K_{ERP} can be set globally so that it affects all the joints, but can be set also on a per-joint basis.

4.6 Soft constraint and constraint force mixing (K_{CFM})

There is another parameter, named constraint force mixing (K_{CFM}), which controls how "hard" the constraints are. For example, the non-penetration constraint is a hard constraint by default, but can be made into a soft constraint, meaning that the constraint will be allowed to be violated, thus allowing to simulate soft bodies. See [Smib] for more details.

4.6.1 Usage of ERP and CFM

K_{ERP} and K_{CFM} can be used to control the spongyness and springyness of the joints and joint limits. If K_{CFM} is set to zero, the constraint will be hard. If K_{CFM} is set to a positive value, it will be possible to violate the constraint by “pushing on it” (for example, for contact constraints by forcing the two contacting objects together). In other words the constraint will be soft, and the softness will increase as K_{CFM} increases.

K_{ERP} and K_{CFM} can be selected to have the same effect as any desired spring and damper constants. Let k_p the spring constant, and k_d and damping constant k_d, we have:

$$K_{\text{ERP}} = \frac{hk_p}{hk_p + k_d} \tag{4.1}$$

$$K_{\text{CFM}} = \frac{1}{hk_p + k_d} \tag{4.2}$$

where h is the step size. These values will give the same effect as a spring-and-damper system simulated with implicit first order integration.

4.7 Limitations

Due to the fact that the physics simulation is carried at discrete time steps, an important aspect is the precision (or lack of precision) resulting by the choice of the framerate, or the number of times per second in which physics are computed. Every frame is computed independently from other frames, so the volume occupied by an object in the transition from one step to another is not computed or considered at all. This is not a problem for a slow moving object or in a simulation with a sufficiently high framerate; but if an object is moving fast or the framerate is low, we incur in a situation where the object makes big jumps from one frame to another. For example, a projectile bullet moving at fast speed, can be at one timestep in the proximity of one face of a wall, and at the next timestep can be at the opposite side, without the collision detector registering any collision, thus producing the effect of the projectile going thru the wall (see Figure 4.2). Some physics engines (Bullet, Havok) use continuous collision detection and do not suffer this problem.

Figure 4.2. An example of what happens when the collisions are only checked at discrete steps and not continuously: a bullet moving at a velocity significantly higher than the framerate, such that the displacement of the object between two frames is larger than the object size, will pass through a thin geometry without registering collision.

Another aspect which may limit the realism of the simulation is due to the numbers representing the positions, velocities and accelerations of objects. When the precision used to represent those quantities is too low, rounding errors may happen and that can negatively affect the outcome of the simulation. This is particularly bad for chain links under high tension and wheels with actively physical bearing surfaces. It is often solved by using greater precision, and the cost of more computational power needed to carry the simulation.

Chapter 5

Physics-based Planning Model

This chapter will present the aspects concerning the use of rigid body simulation as the state propagation function in a kinodynamic motion planning problem.

In this setting the state space $\mathcal{Q}$ must completely describe the state of the physics simulation, in order to allow saving and restoring its state; thus a state $\mathbf{q} \in \mathcal{Q}$ has to completely describe the state of every dynamically-enabled rigid body which is part of the simulated environment. The state of a rigid body is characterized by those properties which change over time, namely: position, orientation, linear velocity, angular velocity, as described in section 4.1. Thus the overall state space is composed by several subspaces which are as many as the properties we need to represent times the number n of bodies in the environment. For 3D position and orientation the space would be $\mathbb{R}^3 \times SO(3)$, while linear and angular velocity are represented by a $\mathbb{R}^3$ space each[1].

$$\mathcal{Q} = \overbrace{\mathbb{R}^3 \times SO(3) \times \mathbb{R}^3 \times \mathbb{R}^3}^{\text{body 1}} \times \cdots \times \overbrace{\mathbb{R}^3 \times SO(3) \times \mathbb{R}^3 \times \mathbb{R}^3}^{\text{body n}} \tag{5.1}$$

This representation becomes redundant in presence of constraints, such as joints. For example in case of two bodies anchored by a hinge joint, one rotational degree of freedom is taken away by the joint, so the dimensionality of $\mathcal{Q}$ is of no indication of the number of degrees of freedom of the system; anyway we are not interested in directly exploring the topology of $\mathcal{Q}$, but rather let the physics simulation engine solve all constraints due to joints and contacts, thus the redundancy of the simulator's state representation is not a problem. What is relevant to consider here is that states

[1] Numerically, an orientation is represented as a quaternion, because that takes the least number of parameter while remaining numerically stable.

in $\mathcal{Q}$ as computed by the physics simulation engine will always be valid and their constraints will be satisfied within the bounds provided by the simulation engine itself.

The control space $\mathcal{U}$ is the set of all possible inputs we can apply to the robot. The topology and dimensionality of $\mathcal{U}$ is not pre-defined and it depends on the robot model and which objects and quantities are being controlled. In most of the cases, when the input is a linear force, torque, or the target velocity of a controller, that subspace is going to be $\mathbb{R}$ or, more realistically, some bounded subset of it.

There's an additional dimension which concerns the application of a specific control input to produce a motion, which is the *duration* of the control input. So when talking about controls we usually refer to the space $\mathcal{U} \times \mathbb{R}^+$.

In order to produce a *motion*, starting from state $\mathbf{q}_k$, the control $(\mathbf{u}, d)$ is repeatedly applied (via physics simulation) until the duration of that piece of simulation is equal to d, and the resulting state $\mathbf{q}_{k+1}$ is obtained.

Searching for the solution to the motion planning problem implies constructing (implicitly or explicitly) a tree of the motions, whose nodes are states $\mathbf{q} \in \mathcal{Q}$ and whose edges are motions of the robot. An edge represents a motion which connects state $\mathbf{q}_k$ to state $\mathbf{q}_{k+1}$. The tree of motions is repeatedly expanded until a state $\mathbf{q} \in \mathcal{Q}_F$ is reached.

The state propagation function $f : \mathcal{Q} \times \mathcal{U} \mapsto \mathcal{Q}$ describes the evolution of the system when applying a certain input, considering all the constraints. In the case of physics-based motion planning, the function f is the result of running such a simulation for one timestep (that is, performing numerical integration and constraints resolution).

5.1 Obstacles and Valid States

In classic motion planning, a fixed obstacle region $\mathcal{W}_{\text{obs}} \subset \mathcal{W}$ must be avoided; its elements can represent invalid states due to bad collision, for example with obstacles in the environment.

In general the obstacle region it is not known explicitly, but given a state it can be checked if it belongs to the obstacle region of the workspace.

Obstacles can be easily identified in domains such as motion planning for robot arms, or for mobile robots moving in the euclidean plane.

However, when planning motion for mobile robots navigating in harsh terrain, the distinction between obstacles and terrain becomes blurry, up to the point that the same patch can be terrain and obstacle at the same time, for example a ledge that acts as an obstacle when approached from a lower level, but acts as the ground while the robot is on it.

Instead of classifying the regions of the map into obstacles or free, we can define the valid subset of $\mathcal{Q}$ by validation of the state, for example by checking kinematic constraints, dynamic constraints, validity of the pose of the robot to ensure it has not tipped over, or collisions between certain parts of the robot and the rest of the environment, in particular for the case of a tracked vehicle, to ensure that only the underside of the tracks is in collision, while the upper parts of the robots (chassis, sensors, electronics) are free from collisions (see Figure 5.1).

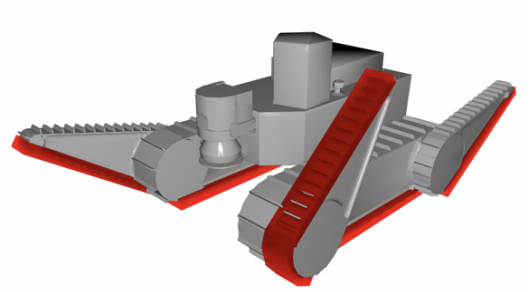

Figure 5.1. A mobile robot model, with some designated contact zones shown in red. Collisions happening within the red regions are considered valid. Any other collisions will cause the state to be considered invalid.

5.2 Unfeasibility of State Sampling

Definition By *state sampling* we mean the random sampling of a valid state according to some probability distribution of choice (e.g.: uniformly random).

In the scenario of physics-based motion planning states are obtained by expanding the tree of motions, by running iterations of the physics engine. Although the state sampler can be domain-aware, for example it is possible to have an obstacle-aware state sampler, for the physics-based approach it is also required that the state is valid in terms of the physics simulation, i.e. there are no colliding bodies.

For every non-trivial robot, with lots of rigid bodies interacting with each other linked via joints, it becomes impossible to sample a valid state from scratch, i.e. a state which is not in collision, valid for the physics simulation, but also reasonable for the context (for example: a mobile robot must absolutely be placed just above the ground in a stable pose).

For this reason, instead of sampling the states, controls are sampled, and then applied via the physics engine. In this way the resulting states are always valid under any point of view.

The disadvantage of sampling controls (versus sampling states) is that the new obtained states will tend to be in the vicinity of other states. In fact, the probability of generating an invalid state grows with the duration of the sampled control. So the tree will grow slowly, and the coverage will also be poor. For example, in the case of a simple tracked robot, we have left and right track velocity as control inputs, and a given control input will make the robot drive along an arc; the longer the duration of the control, the longer the traveled distance will be, and higher the probability of hitting some obstacle or tipping over will also be higher.

5.3 Unfeasibility of State Steering

Another important aspect in motion planning, which can provide important speedups and simplifications for the search algorithm, is the *state steering*.

Definition State steering is the problem of finding a control that brings one state $\mathbf{q}_1$ to another state $\mathbf{q}_2$, that is, to find the control $\mathbf{u} \in \mathcal{U}$, $d \in \mathbb{R}^+$ that applied to state $\mathbf{q}_1$ will have $\mathbf{q}_2$ as the final state.

In the Probabilistic Roadmap Method (PRM) [Kav+96], the use of state/control sampling and state steering are the key points of the algorithm, which are used for building a graph, which should be ideally covering all the state space, and have a high degree of connectivity, so that then with a simple graph search is possible to find the motion from any state to any other state.

The ability to find these arcs in the tree of motions allows the planner to *connect* states which are in proximity of each other, thus turning the tree of motions into a *graph* (sometimes referred to as *roadmap*).

Occupancy-based path planners can steer the states easily, because they assume that within a free cell, the robot is able to perform *any* motion, for example for a tracked robot in a 2D map, it is commonly assumed that the robot is always free to rotate in plance, hence every free cell can be connected to every other free cell with a straight line, provided that there are no obstacles in between.

This is not the case with physics-based motion planning, as finding a motion from a state to another, is a stricter version of the problem being solved.

Lack of state steering leaves us with a tree of motions which is suboptimal, because there can be branches sharing similar states, with lot of motions which are actually duplicated or equivalent, back up to the root (i.e. the initial state).

5.4 Path planning methods

The physics-based motion planning framework is not tied to a particular motion planning algorithm. Here for completeness, one of the most common algorithms is summarized, and other relevant algorithms derived from it are mentioned.

5.4.1 Rapidly-exploring random trees (RRT) algorithm

A rapidly exploring random tree (RRT) [LaV98] is an algorithm designed to efficiently search, high-dimensional spaces by randomly building a space-filling tree. The tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large unsearched areas of the problem. RRTs easily handle problems with obstacles and differential constraints (nonholonomic and kinodynamic) and have been widely used in autonomous robotic motion planning.

RRTs can be viewed as a technique to generate open-loop trajectories for nonlinear systems with state constraints. An RRT can also be considered as a Monte-Carlo method to bias search into the largest Voronoi regions of a graph in a configuration space. Some variations can even be considered stochastic fractals.

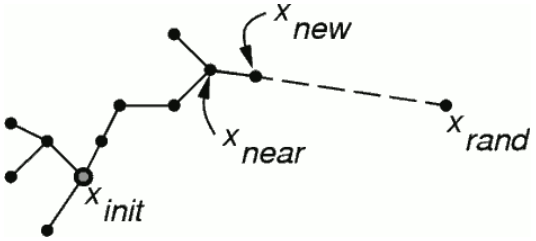

Figure 5.2. The RRT algorithm consists of repeating these five operations: 1) pick a random sample in the search space 2) find the nearest neighbor of that sample 3) select an action from the neighbor that heads towards the random sample 4) create a new sample based on the outcome of the action applied to the neighbor 5) add the new sample to the tree, and connect it to the neighbor.

An RRT grows a tree rooted at the starting configuration by using random samples from the search space. As each sample is drawn, a connection is attempted between it and the nearest state in the tree. If the connection is feasible (passes entirely through free space and obeys any constraints), this results in the addition of the new state to the tree. With uniform sampling of the search space, the probability of expanding an existing state is proportional to the size of its Voronoi region. As the largest Voronoi regions belong to the states on the frontier of the search, this means that the tree preferentially expands towards large unsearched areas.

The length of the connection between the tree and a new state is frequently limited

```
GENERATE_RRT(x_init, K, Δt)
 1  T.init(x_init);
 2  for k = 1 to K do
 3      x_rand ← RANDOM_STATE();
 4      x_near ← NEAREST_NEIGHBOR(x_rand, T);
 5      u ← SELECT_INPUT(x_rand, x_near);
 6      x_new ← NEW_STATE(x_near, u, Δt);
 7      T.add_vertex(x_new);
 8      T.add_edge(x_near, x_new, u);
 9  Return T
```

Figure 5.3. The RRT algorithm.

by a growth factor. If the random sample is further from its nearest state in the tree than this limit allows, a new state at the maximum distance from the tree along the line to the random sample is used instead of the random sample itself. The random samples can then be viewed as controlling the direction of the tree growth while the growth factor determines its rate. This maintains the space-filling bias of the RRT while limiting the size of the incremental growth.

RRT growth can be biased by increasing the probability of sampling states from a specific area. Most practical implementations of RRTs make use of this to guide the search towards the planning problem goals. This is accomplished by introducing a small probability of sampling the goal to the state sampling procedure. The higher this probability, the more greedily the tree grows towards the goal.

5.4.2 KPIECE

In [Şuc+09] an estimate of the coverage is maintained, in order to favor the exploration of less-explored areas. The boundary of the explored region of the state space is also tracked, and the planner focuses exploration on the less covered parts of this boundary.

5.4.3 EST

Expansive Space Trees (EST) [Phi+04] perform search for a low cost and relatively straight path in a space with motion constraints. The algorithm chooses vertices for expansion that have few neighboring vertices around them. Path Gradient Descent finds the locally optimal path for an existing path.

5.4.4 PDST

Path-Directed Subdivision Trees (PDST) [Kav+05] represents samples as motions (path-segments) rather than individual states, and uses non-uniform subdivisions of the state space to estimate coverage. This supposedly avoids many of the problems that previous sampling-based planners have had with metrics and coverage-estimation.

5.4.5 Exploration vs. Exploitation

The strength of RRT lies in the fact that it uniformly samples the state space. Can in fact also be considered a Monte-Carlo method to bias the search into specific regions of the configuration space. Even when the state space is not directly steerable, it uniformly samples controls and select the one which is *closest* to the randomly sampled state, so it still achieves similar properties to the original state-space planning problem. In other words, RRT facilitates *exploration* of the state space.

On the other hand, suppose we know how to compute an heuristic function, that can encode domain knowledge about mobile robot navigation in 3D terrain. This function $h : X \mapsto \mathbb{R}$ can for example take into account the distance to closest obstacle, the slope and roughness (mean and variance) of the terrain, the uncertainty in the map data, i.e. the density of the point cloud map, the distance to goal, and many other factors, which will be treated in detail in the next chapter.

The heuristic function tells us some preference, that is to prefer state x_i over state x_j, because $h(x_i) < h(x_j)$.

By modifying the RRT algorithm, and instead of selecting the state which is closest to x_{rand}, we select the state which is best ranked by the heuristic function $h(\cdot)$, we are clearly biasing the search towards *exploitation* (of our domain knowledge).

Like in simulated annealing algorithms and Markov-Chain Monte Carlo methods, a tradeoff between exploration and exploitation is the key to success [Smi+93; And+10].

In chapter 7 we will see in detail how these search heuristics based on geometrical features and empirical knowledge about mobile robot navigation can be computed.

5.5 Architecture of the physics-based motion planner

The architecture of the physics-based motion planner works as follows. A data structure is used to maintain a tree of motions. The query state is inserted into the tree of motions. If the goal is reached, the procedure stops. Otherwise a state is selected from the tree of motions. From that state, a control is sampled, and a new state is computed using the physics simulation engine. The new state is weighted with the heuristic function. The process repeats. See also Figure 5.4.

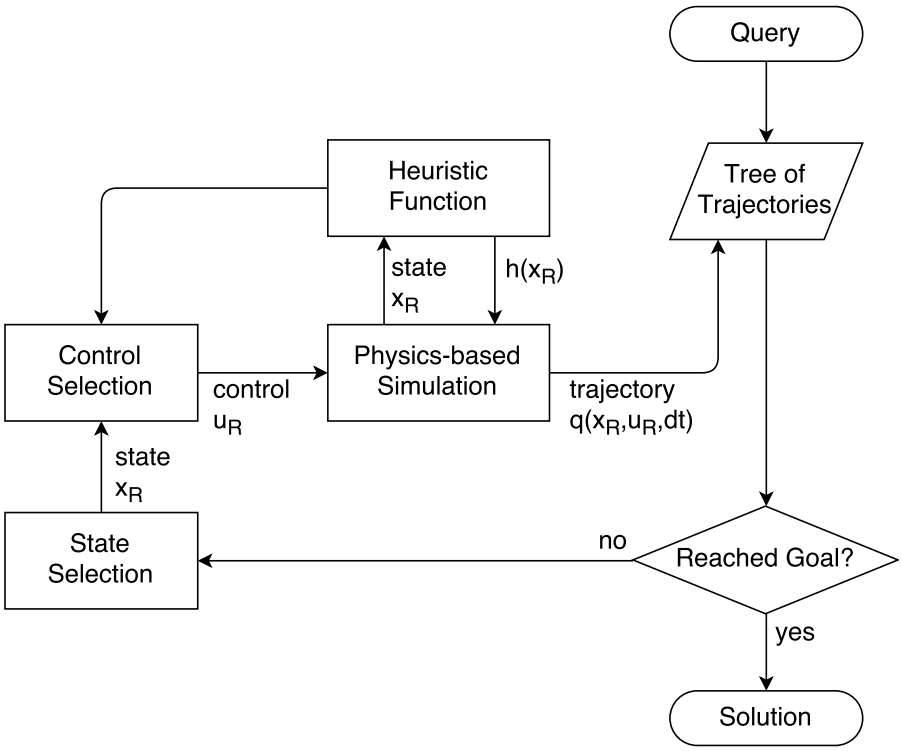

Figure 5.4. A diagram illustrating the architecture of the physics-based motion planner: the tree of trajectories is built iteratively and incrementally (e.g.: upon receiving a new query); if the goal has not been reached yet, a state is selected from the tree of trajectories for expansion, and several control inputs are sampled, propagated via the physics based simulation, and ranked according to the heuristic function, and the resulting trajectory increment is put back into the tree of trajectories.

5.6 Implementation

This architecture has been implemented in C++ using the OMPL motion planning library [Mol+12], which is designed to be easily extended with custom state space samplers and control space samplers.

In the custom control space sampler we sample some (one or more) controls, perform state propagation, rank the resulting states according to the heuristic function, and choose the best one.

If the number of sampled controls is equal to one, we are using the default behavior of the planning algorithm, which is to randomly explore the state space.

The more controls are sampled and evaluated with the heuristic function, the more we are biasing the search towards exploitation of our domain knowledge.

In chapter 7 we will see in detail how these search heuristics based on geometrical features and empirical knowledge about mobile robot navigation can be computed.

Chapter 6

Simulation of Tracked Vehicles

In this section we analyze the design process of the simulation of a track with grousers. This is the type of locomotion device that rescue robots, exploration robots, and planetary rovers are equipped with, and is suitable for navigation in a wide range of terrain types, from indoors, including stairs, ramps, steps, surmounting small obstacles and holes, to outdoors, hilly terrain, harsh terrain, disaster areas, collapsed buildings, debris, and so on.

The track can be a single flexible (i.e. hard rubber) belt with or without grousers, or a chain of rigid elements (see Figure 6.1) with or without a grouser on each element.

Grousers are physical elements meant to increase the traction of tracks in loose terrain such as soil, snow or debris. This is achieved by increasing the contact between track and terrain with several protrusions, which have a similar function to the treads of tires used in vehicles.

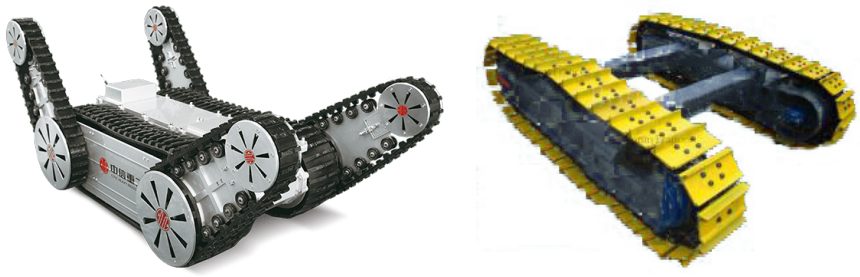

Figure 6.1. Left: a robot with flexible (rubber) tracks. Right: another robot with rigid, chain-like, tracks.

6.1 Chain-like track

A track which is built with many identical elements, connected by hinge joints in a chain fashion, just like a transmission chain (see Figure 6.3), is an approach which goes very close to reality. Big vehicles (like tanks) have tracks which work exactly like this. Smaller vehicles can have rubber tracks, but the interaction with the terrain is the same as rigid chain-like tracks. The advantage of using chain-like tracks is that they can be simulated quite well using rigid body simulation, while for rubber tracks we would need soft-body dynamics which is more computationally expensive.

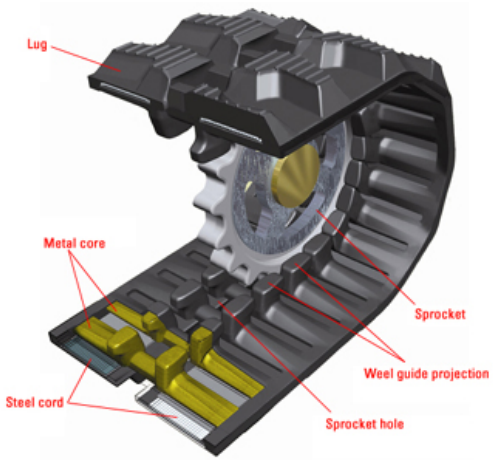

Figure 6.2. Section of steel track with center sprocket holes to avoid track slipping off.

One problem that has to be faced (both in reality and in simulation) is the slipping of the chain off the rollers. When mounting a chain around some rollers (i.e. cylinder geometries) the chain tend to slip away, especially when driving the robot along curved paths. Real tracks have one or more central guides to ensure that the chain doesn't slip away (see Figure 6.2)

In simulation we cannot afford to make a chain element geometry which is too much complicated, because that would slow down the simulation too much. Consider that a track can be composed in average of hundred of elements, and each element can generate several contact points colliding with the rollers and with the terrain. The situation is even worse if the chain element geometry is a mesh instead of a pure shape (like a cuboid, a cylinder or a capsule), and even worse if the mesh is not convex, because in that case the process of checking collision with other elements would be exponentially slow.

What we would like to have is a chain element geometry which is a pure shape (like a cuboid), or also a collection of pure shapes (see Figure 6.3), that can be rigidly coupled together while maintaining the efficiency of collision checking with pure

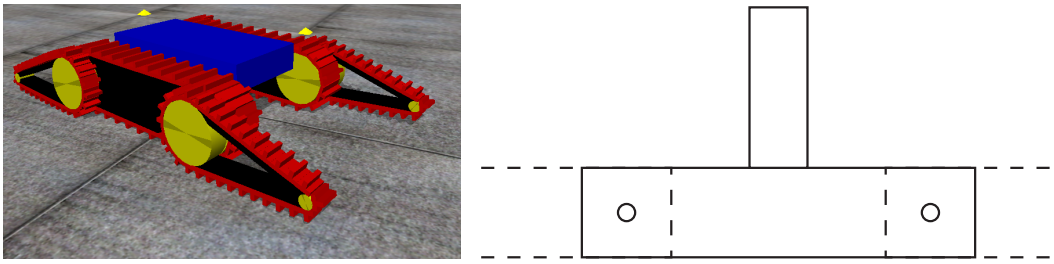

Figure 6.3. The tracks modeled as a chain of identical geometries. Each element is connected to the next element via a hinge joint. In addition, the elements are connected to the planar joint, to avoid slipping away while driving on curved paths.

shapes.

Moreover, we want to avoid collision as much as possible. It is better to have a geometric constraint (also known as joint) to constrain the motion (i.e. remove some degree of freedom) of some object with respect to another object. In that case we would have a mathematical model to force the object to be in the correct place, that is, in the case of grousered tracks, to not slip off the rollers.

We implement this constraint as the *planar joint*, which restricts 2 rotational and 1 translational degrees of freedom.

The 2 rotational constraints are essentially the same as in the piston joint, so the implementation here is mostly just a copy of them.

The translational 1DOF constraint has not yet been implemented for any joint in ODE, so we'll talk more about it. It is similar to the constraints in a slider joint, but in this case we only want body2 to move the plane, and body1 anchor has no use in this case, too.

We basically want to express the plane in body2 local coordinates (because it should move along with body2), and check, if body1 origin lies in the plane.

Let's say n' is the plane normal in body2 coordinates and a' is the anchor point in body2 coordinates (with n, a, being the corresponding global variables). Further, let's call the global position of body1 p_1, and similarly p_2 for body2. Last, let's call body2 rotation matrix R_2 (so that $n = R_2 n'$ and $a = R_2 a' + p_2$).

The plane can then be expressed in global coordinates as:

$$n^\top x - n^\top a = 0 \tag{6.1}$$

where x is in global coordinates.

Rewriting the equation using body2 local coordinates and setting $x = p_1$, we get the constraint:

$$(R_2 n')^\top * p_1 - (R_2 n')^\top * (R_2 a' + p_2) = 0 \tag{6.2}$$

$$\dots$$

$$(n'^\top R_2^\top p_1) - (n'^\top R_2^\top p_2) - (n'^\top a') = 0 \tag{6.3}$$

$$(n^\top p_1) - (n^\top p_2) - (n'^\top * a') = 0 \tag{6.4}$$

The part left to = will be the "c" on the right-hand side of the Jacobian equation (because it expresses the distance of p_1 from the plane).

Next, we need to take time derivative of the constraint to get the J_1 and J_2 parts. v_1, v_2, w_1 and w_2 denote the linear and angular velocities od body1 and body2. $[a]_\times$ denotes the skew-symmetric cross-product matrix of vector a.

$$\frac{d}{dt}[(n'^\top R_2^\top p_1) - (n'^\top R_2^\top p_2) - (n'^\top a') = 0] \tag{6.5}$$

(n', a' are constant in time)

$$n'^\top((\frac{d}{dt}[R_2^\top]p_1) + (R_2^\top v_1)) - n'^\top((\frac{d}{dt}[R_2^\top]p_2) + (R_2^\top v_2)) = 0 \tag{6.6}$$

$$n'^\top((([w_2]_\times R_2)^\top p_1) + (R_2^\top v_1)) - n'^\top((([w_2]_\times R_2)^\top p_2) + (R_2^\top v_2)) = 0 \tag{6.7}$$

$$\dots$$

$$n^\top v_1 - n^\top v_2 + [n]_\times (p_1 - p_2) w_2 = 0 \tag{6.8}$$

$$-n^\top v_1 + n^\top v_2 + [n]_\times (p_2 - p_1) w_2 = 0 \tag{6.9}$$

Thus we see that

$$J_{1l} = -n \tag{6.10}$$

$$J_{2l} = n \tag{6.11}$$

$$J_{1a} = 0 \tag{6.12}$$

$$J_{2a} = [n]_\times (p_2 - p_1) \tag{6.13}$$

$$c = \epsilon(n^\top p_1 - n^\top a) \tag{6.14}$$

Chapter 7

3D Path Planning Search Heuristics

The physics-based motion planning framework described at beginning of chapter 5 is capable of finding proper solutions (i.e. kinematically and dynamically feasible) to mobile robot motion planning problems, regardless of any search heuristics. However it takes in general enormous amounts of time to find a solution.

The cause of this problem has to be identified in two facts: i) physics simulation is quite demanding in terms of CPU time, even though it is designed to be run in real-time, it cannot be sped up by even an order of magnitude; ii) common kinodynamic planning algorithms, which apply domain-independent strategies to explore the state space, try to expand states in every direction, in order to maximize state space coverage (such as [Kav+12]) or have some simple bias such as straight-line distance to goal (if such metric is available). This means that these domain-independent algorithms can spend most of the time expanding *useless states*, i.e. states which are not much useful, when compared to the states of a shortest-path solution.

In fact, the quality of the solution found by these algorithms, especially when applied to non-steerable systems, is quote poor: the solution appears to be like a random walk which eventually, unexpectedly, hits the goal (see Figure 7.1). This makes perfectly sense and has to be expected, because we cannot use optimizing planners, due to the state-steering limitation, which always applies in case of physics-based motion planning.

Suppose we already know the shortest-path solution to the problem: in this case it would be possible to avoid expanding useless states, by just sampling controls which go along the states touched by the optimal solution. Practically, this is not really possible, because this approach is unstable, as we have no guarantee that a sampled control will land exactly on a chosen state: it's the state steering problem all over

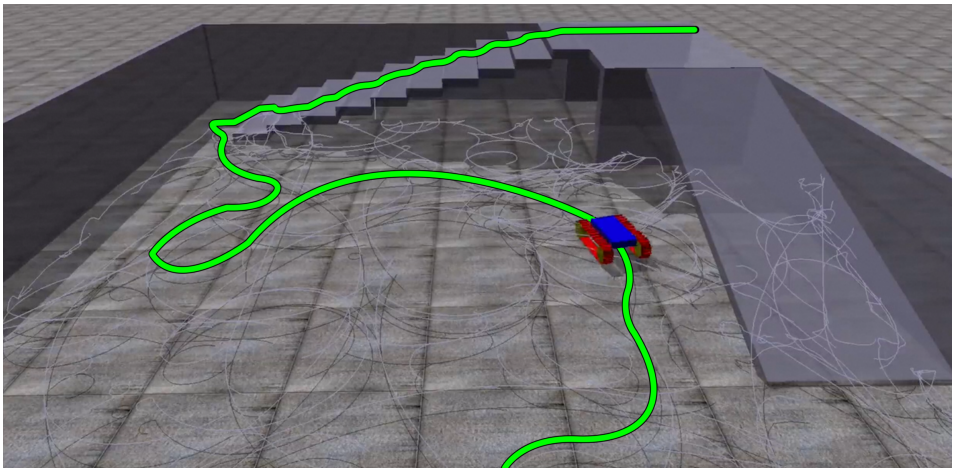

Figure 7.1. The solution found by the physics-based motion planning described in chapter 5: due to lack of state-steering, the path (green) cannot be optimized. Also, lots of motions (light gray) are expanded which are not relevant to the solution.

again.

Suppose instead we know all the shortest-path solutions, from any point in the space (to be precise, from any point on the terrain) to the goal. In this case a different strategy is possible: once a control is sampled, and the resulting state has been obtained via state-propagation, we know what is the distance from that state to the goal. According to this information, it is possible to *rank* controls, in order to give preference to a good one that would quickly go towards the goal.

So, if we already know all the optimal solutions, why bother with physics simulation, state propagation, control sampling and such? Sadly, because this "oracle" does not exist, and cannot exist. If it is computing solutions much faster than a physics simulation, for example using geometric approaches, then it must be using some kind of approximation on the available information, which is the thing we wanted to avoid in the first place and the reason to use physics simulation to perform motion planning.

However, if we limit the use of this approximate approach only for reordering the choices made by the physics-based motion planner, we are not discarding any precious information, but if the approximation is good, the planner may converge to a solution much faster; consider the hypothetical case where the approximation is perfect: the physics-based motion planner will expand one motion after the other, in a depth-first fashion, going straight to the goal, and expanding only few other states in the neighborhood of the optimal solution.

Thus, rearranging the order in which controls are sampled and states gets visited, is not going to affect the soundness or completeness of the physics based planner, and that's exactly the purpose of the heuristic function.

We compute search heuristics based on shortest distance over a graph of points in euclidean space, where the connectivity between points and the cost of going from one point to another, are specially crafted to approximate the robot locomotion ability.

The quality of the search heuristics is not going to affect the validity of the solutions found in any way, as solutions get found by physics-based motion planning anyway; however the search heuristics will change the order in which the state space is visited, so it can greatly affect of rapidly the solution is found: the better the heuristics are, the faster a solution is found.

These search heuristics have a role similar to search heuristics in the A* algorithm [Har+68] in finding the solution, except that in this case there is no notion of admissible costs like it is in A*.

$G \leftarrow computeGraph(Map)$
$G' \leftarrow dijkstra(G, goal)$
$G'' \leftarrow distanceTransform(G', goal)$
$T \leftarrow \emptyset$
for all v in G'' **do**
    add v to kdtree T
end for

function H(p)
    $v \leftarrow$ closest vertex in T
    if $\|p - position(v)\| <$ treshold **then return** $distance(v)$
    end if
end function

Figure 7.2. Algorithm for heurstics. Above: initialization of data structures (to be run each time map or goal changes). Below: pointwise function H computing heuristic distance to goal.

7.1 Computation of heuristics

To compute heuristics, several methods, presented in the next sections, are used to induce a weighted undirected graph over a 3D map of the environment. These methods consist mainly of terrain classification, and traversability cost estimation.

The obtained graph, together with the goal query, is given in input to an all-pairs shortest path algorithm, such as Dijkstra's algorithm [Dij59].

The information about node predecessors is used together with edge weights to label each vertex with the minimum cumulative cost towards goal. This step is sometimes referred to as the *distance transform*, but in this case edge weight is used in place of distance.

The set of labeled vertices is stored in a Kd-tree, and is queried with a 3D projection of the robot, by looking at the closest vertex within a maximum distance threshold, and the distance label is returned.

The algorithm is summarized in Figure 7.2.

This allow practical estimation of the shortest distance to the goal from any position, and allows ranking of sampled controls. See architecture description in Figure 5.4.

7.2 Graph-based heuristics

In this section we present a framework for computing search heuristics based on clustering and segmentation of point clouds, and traversability analysis. This framework has been shown to be effective for 3D autonomous navigation of tracked vehicles [Men+14].

We consider the practical case of a mobile tracked robot, equipped with two main track grousers, and four articulated flipper grousers. For better traction on harsh terrains these robotic platforms can either increase or decrease the tracks contact area with the ground. However, terrain adaptability depends not only on the mechanical design but mainly on the controller ability to accurately adapt the active parts of the tracks. In fact, even if the terrain surface is well approximated on the basis of vision and point cloud modeling, a compliance interaction between active parts of the tracks and the terrain needs to be established.

The robot configuration state is defined by the vector

$$q = (x_r, y_r, z_r, \psi, \phi, \theta, v_r, \omega_r, \alpha_1, ..., \alpha_4)^\top \tag{7.1}$$

with $x_r, y_r, z_r, \psi, \phi, \theta$ the 6D pose of the robot, v_r and ω_r the linear and angular robot velocities, respectively. Here, $\alpha_1, \ldots, \alpha 4$ are the configurations of the robot flipper. The state can be separated into two controllable parts, assuming that the control of the robot pose is independent of the flippers control. Under this assumption, the 3D motion planning controller can be divided into two decoupled control modules: (1) a trajectory tracking controller, and (2) a flippers position controller. These modules work in parallel and are synchronized so as to generate, at time stamp, the control commands needed to track a given 3D path and to simultaneously adapt the position of the flippers to the surfaces on which the path lies, namely to the planes tangent at each path point. The trajectory tracking controller receives as input a path P, generated on the 3D labeled map of the environment and computes the steering commands to allow the robot to follow the given path P. The control strategy underlying the trajectory tracking controller is based on input-output linearization via feedback. The flippers position controller receives the desired flipper positions $\alpha_d(t) = (\alpha_1(t), \ldots, \alpha_4(t))^\top$, as input, generating suitable internal speed commands to asymptotically stabilize to zero the flippers position error, on the basis of a proportional-derivative (PD) control law. The positions $\alpha_d(t)$ are selected from a set of predefined configurations, depending on both the current robot attitude and the information provided by the 3D labeled map of the environment. These configurations allows the robot to climb stairs and to overcome both positive and negative obstacles. However, during the control loop, the flipper position controller is not able to detect, after positioning the flippers according to the selected posture, whether the flippers effectively are either in contact with the surface or with the

obstacle to be surmounted. Still, the flippers are neither endowed with contact sensors nor with proximity sensors. To face this limit, the flipper position controller integrates a statistical model assessing the touch and the detach of the flippers with the surface. This model is based on a function, learned from a set of features, extracted from the direct measurements of both the actual angles feedback(t) of the flippers and the electrical currents ii(t) of the flipper servo motors. During the control loop, the controller adjusts the flippers posture $\alpha_d(t)$, depending on the feedback provided by the contact sensor model.

7.2.1 Point cloud segmentation and labeling

In this section we describe a real time point cloud PC segmentation and labeling, registered by an ICP-based SLAM [Pom+13]. The point cloud PC stores the geometric position (x, y, z) of each point $\mathbf{p} \in PC$, approximating a surface M. The objective is to establish traversability of the surface for the path control, requiring real-time computation. To this end we define just four categories for traversability: *ground, walls, ramp or stairs, surmountable obstacle.* These categories have been specified to match the effective robot overcoming abilities: 30 cm steps with rounded edge; 40 cm gap; 20 cm stairs at 40° slope; spiral staircases; 45° slope / 15° roll; $80cm \times 80cm$ cavities. Here we assume that segmentation is always performed on the incoming laser measures, and we consider the current one, in so avoiding to parametrize points with time.

We propose here a novel approach to simple categories segmentation based on geometric features. The approach is specified by the following steps (1) patch construction and filtering; (2) estimation of the normals and of the principal curvatures of the surface at all points and (3) segmentation and labeling of the PC.

Patch construction and Filtering

An initial points filtering is already provided by the ICP algorithm, here we are mainly concerned with constructing a surface patch for each PC region, used for geometric features computation, and to discard ill-conditioned neighborhoods. A patch is defined by a mapping $\psi : (x, y) \mapsto M$, with $\psi(x, y) = \mathbf{p} \in \mathbb{R}^3$, and $(x, y) \in \mathbb{R}^2$, the parameters of the mapping. To simplify we denote $\psi(x, y)$ by $\mathbf{x}$ intending also the coordinates of the point $\mathbf{p}$ on the surface M; similarly, we identify the parameters with the x and y coordinates. The patch can be obtained by transforming the point representation of PC into a voxel grid, building a suitable neighborhood for points that can be reordered into surface patches of contiguous points, and ensuring that there is none or little overlapping between patches. First we decimate the PC of a factor of 0.9 to obtain the downsampled PC_s, and then we initialize the

neighborhood of each point $\mathbf{q}$ in PC_s with k-means taking the $\mathbf{q}$ as seeds, using only the points coordinates together with the Euclidean distance. Since k-means does not take care of ill-conditioned clusters we adjust each cluster by expanding it or reducing it, by splitting or merging, according to the cluster spatial frequency $\rho = \frac{|H|}{r^3}$, where $|\cdot|$ is the cardinality of the cluster H and $r = mean(d(H))$ is the radius defined by the mean distance within the cluster. An optimal density $\rho^\star$ is given by a radius $r_{min} = 0.5\sqrt{2}m$, (m, meters), and a neighborhood of 81 points. If $\rho < \frac{\rho^\star}{2}$ then r is increased to catch more points joining with the cluster that ensures to keep the optimal dimension, among the closer ones. To control the radius growth and filter out noise, a noise function is specified according to the spatial frequency of points as follows. Let $\varphi(H) = exp(-\pi(r_{min} - r))$, then the probability that a neighborhood H is affected by noise is:

$$p(noise(H)) = 1 - \varphi(H) \tag{7.2}$$

The noise function is clearly zero when $r = r_{min}$. Neighborhoods with noise value greater than 0.5 are removed from the PC. Once the clusters are stabilized then, each of them is labeled into a voxel grid, possibly resorting to interpolation, mostly required at the boundary of a patch, so that close points are contiguous. The grid representation leads to the surface patch representation of the three matrices $\mathbf{X}$, for the x coordinates of the points in the cluster, $\mathbf{Y}$ for the y coordinates, and $\mathbf{Z}$ for $\psi(x, y)$. These matrices have size $m \times n$ each.

Normal and principal curvature estimation

Having transformed the original PC into a number of surface patches well approximating the surface M, using the neighbors construction and the grid, a very precise estimation of the normals, and likewise of the principal curvatures, can be obtained quite simply, using first and second derivatives. This allows us to avoid the computation based on least mean square (see for example [Hop+92; Pau+03]), which can be quite sensitive to noise and it is computationally very expensive, since it requires to compute neighborhoods for each point of the cloud and to perform SVD decomposition.

Here we consider finite-size linear-phase separable kernels, introduced by [Far+04], for first and second order differentiation of discrete multidimensional signals. These kernels are used to convolve the matrices $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ to obtain the derivatives, by first expanding the matrices at the boundaries. Then, differentiation for each coordinate x and y is obtained by collecting the convolution of each matrix in each direction. Namely, let f_x, g_x, h_x be the first-order derivatives of $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$, respectively, with respect to x, obtained with the convolution kernel, these are matrices of size $m \times n$ too. Therefore $\mathbf{x}_x$ is the tensor $(f_x, g_x, h_x)^\top$ and, analogously, $\mathbf{x}_y = (f_y, g_y, h_y)^\top$. In an analogous way we obtain the second-order derivatives $\mathbf{x}_{xx}$,

$\mathbf{x}_{yy}$, $\mathbf{x}_{xy}$. Having defined the patches in explicit form, it is now straightforward to obtain the normals $\mathbf{n}s$ for all the points in a patch, by simply applying the cross product to the first derivative tensors, obtaining the tensor $3 \times n \times m$:

$$\pm \begin{pmatrix} g_x \cdot h_y - g_y \cdot h_x \\ f_y \cdot h_x - f_x \cdot h_y \\ f_x \cdot g_y - f_y \cdot g_x \end{pmatrix} \tag{7.3}$$

and normalizing to get the tensor $\mathbf{N}$ of unit normals, noting that in the above equation $\cdot$ is the element-wise matrix multiplication. Note that the sign can be actually determined by simply computing the smallest angle between the obtained normals and the laser source position, say $\mathcal{S}$. Namely, we look for the angle θ minimizing $\cos^{-1}(\mathbf{s}^\top \mathbf{n}'_{3\times mn})/\|\mathbf{s}\|\|\mathbf{n}'_{3\times mn}\|$, where

$$\mathbf{s} = (\mathcal{S} \otimes \mathbf{1}_{1\times nm} - \mathbf{x}'_{3\times nm})/\|\mathcal{S} \otimes \mathbf{1}_{1\times nm} - \mathbf{x}'_{3\times nm}\|$$

with $\mathcal{S}$ the source position, $\otimes$ the Kronecker product, $\mathbf{x}'_{3\times nm}$ and $\mathbf{n}'_{3\times nm}$ tensors of size $(3 \times n \times m)$ reshaped into a $3 \times nm$ matrix.

As for the principal curvatures, at each point $\mathbf{p}$ in a patch, we exploit the fact that with the first and second derivatives and the normals tensor $\mathbf{N}$ available, we can obtain the following elements, named according the I and II fundamental forms:

$$\begin{array}{lll} E = \Sigma_{dim=3}\mathbf{x}_x \cdot \mathbf{x}_x & F = \Sigma_{dim=3}\mathbf{x}_x \cdot \mathbf{x}_y & G = \Sigma_{dim=3}\mathbf{x}_y \cdot \mathbf{x}_y; \\ L = \Sigma_{dim=3}\mathbf{x}_{xx} \cdot \mathbf{N} & M = \Sigma_{dim=3}\mathbf{x}_{xy} \cdot \mathbf{N} & N = \Sigma_{dim=3}\mathbf{x}_{yy} \cdot \mathbf{N} \end{array} \tag{7.4}$$

In other words E, F, G, L, M, N are all matrices $m \times n$. Thus, k is computed as the solution of the quadratic equation

$$(EG - F^2)k^2 + (2FM - EN - GL)k + (LN - M^2) \tag{7.5}$$

And it is well-known that the discriminant of the above equation is greater or equal zero, therefore there are either two distinct real roots k_1 and k_2, namely the two principal curvatures, or a single one. While at umbilical points $L/M = M/F = N/G$. Given the principal curvatures we can also obtain both the mean curvature $(1/2)(k_1 + k_2)$ and the Gaussian curvature $k_1 k_2$. We observe that the only computational difficulty here is the construction of the patches. The main difficulties coming from far points, which are those mostly requiring merging and dropping, therefore it might be better for those noisy regions to delay the patch construction, according to the noise function defined in Equation 7.2.

Segmentation and Labeling

Segmentation still uses the matrices $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ yet all integrated into the voxel grid. We note that without hampering the information for the categories, the kernel convolution included a significant smoothing of the matrix $\mathbf{Z}$. Then we can obtain from

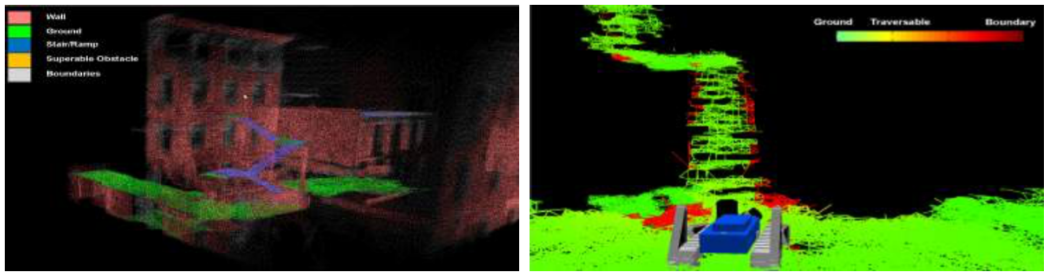

Figure 7.3. Left: point cloud segmentation and labeling. Right: weighted graph representation of a fire escape stairs scenario..

the normals, the Gaussian curvature and the principal curvatures an initialization of the categories as follows. Let $|\mathbf{n}| = (|n_1|, |n_2|, |n_3|)^\top$ be the unit normal absolute value at a point $\mathbf{p}$, let $K = k_1 k_2$ be the Gaussian curvature, with k_1 and k_2 the principal curvatures, let $\theta_1 = 0.17$ and $\theta_2 = 0.95$ then the following table illustrates the initialization values:

	Parameters	
Classes	$\lvert n_1\rvert, \lvert n_2\rvert, \lvert n_3\rvert$	K, k_1, k_2,$\mathbf{Z}$
ground	$\lvert n_1\rvert, \lvert n_2\rvert$<0.01, $\lvert n_3\rvert \geq 0.8$	$\lvert K\rvert$<0.1
walls	$\lvert n_2\rvert, \lvert n_3\rvert$<0.01, $\lvert n_1\rvert \geq 0.8$	$\lvert K\rvert$<0.1
ramp	$\theta_1 \leq \lvert n_3\rvert \leq \theta_2$ $\lvert n_1\rvert \leq 0.1$, $\theta_1 \leq \lvert n_2\rvert \leq \theta_2$ $\lvert n_2\rvert \leq 0.1$ $\theta_1 \leq \lvert n_1\rvert \leq \theta_2$	 $\lvert k_2\rvert < 10^{-2}$, $\lvert k_1\rvert \geq 1$ $\lvert k_1\rvert < 10^{-2}$, $\lvert k_2\rvert \geq 1$
surmountable obstacles	$\theta_1 \leq \lvert n_3\rvert \leq \theta_2$	$\lvert K\rvert \geq 1$, $\mathbf{z} > 0.3m$.

Because of noise these parameters produce scattered segmentation, to even the neighborhoods of the above defined categories we introduce an energy functional for each class as follows.

All points in a class are assigned kinetic energy E_k; a stop condition is given by a function $g(\mathbf{p})$ which stops the front collecting points, at a given direction of growth, whenever there is a jump in energy level. Let the normal for the given classes be specified by $\mathbf{n}_e$ with $e = \{ground, wall, ramp, surm.obstacle\}$. The energy level for class e is $E_{\ell_e}(\mathbf{p}) = \cos^{-1}(\mathbf{n}^\top \mathbf{n}_e)$, with $\mathbf{n}$ the normal at the current point $\mathbf{p}$ considered, with $\mathbf{p}_0$ a starting point. Let κ denotes either the Gaussian curvature or the principal curvatures, according to the class and its constraints considered. Let:

$$\eta(\mathbf{p}_0, \mathbf{p}) = s\left(|\kappa(\mathbf{p}) - \kappa(\mathbf{p}_0)|\right) + s\left(|E_{\ell_e}(\mathbf{p}_0) - E_{\ell_e}(\mathbf{p})|\right) \tag{7.6}$$

Here s is the logistic function $y = c/(1 + a\exp(-bx))$, with $b = 25$ and $a = \pi^5$. We have chosen for the limiting upper bound $c = 100$, these values ensures that when the difference in energy level/curvature increases beyond a threshold 0.08 then the

function fires high values up to c. Let

$$g(\mathbf{p}) = E_k(\mathbf{p}_0) - \eta(\mathbf{p}_0, \mathbf{p}) \tag{7.7}$$

Points $\mathbf{p}$, around $\mathbf{p}_0$, are collected as far as $g(\mathbf{p}) > 0$: Therefore when $g \leq 0$ then the front expansion stops for that point. To resume:

1. *Input*: the set of surface points, the normals, the curvatures.
2. Assign kinetic energy E_k to the chosen points $\mathbf{p}_0$ in the class.
3. Compute a path to a farthest point such that for every point in the path the difference in energy level and curvature is low, namely $\eta(\mathbf{p}_0, \mathbf{p}) \leq 0.1$.
4. Move the neighborhood front in all directions, checking for each direction the stopping criterion $g(\mathbf{p})$, Equation 7.7.
5. Remove the points collected into the formed clusters, if the set of surface points is empty stop, else go to item 2).
6. *Output*: the list of clusters.

Note that we choose a furthest point together with the path, and define the kinetic energy in place of a cost, confront with [Sch+11]. The final obtained clusters are labeled according to the specified categories.

7.2.2 Graph generation

In this section we describe how the connectivity and traversability graph is obtained. The proposed method requires three steps:

- estimation of the inflated regions the environment;
- definition of a graph G, connecting the points of the labeled cluster;
- weighting of the the graph, returning the traversability graph.

These steps are illustrated in the following sections.

Inflated region estimation

Let $\mathcal{A}$ be the set of clusters, labeled with the class they belong to. Let $\mathcal{A}' = \{(C_k, c_k) : c_k = wall\} \subseteq \mathcal{A}$ and $\mathcal{A}^\star = \mathcal{A} \backslash \mathcal{A}'$. We introduce the points belonging to inflated regions $\mathcal{B}$, which are used to specify a feasible path. Let $u : PC \mapsto PC$ be the function:

$$u(\mathbf{p}) \triangleq \underset{\mathbf{p}_j \in C_j}{\arg\min} \{ \left\| \mathbf{p} - \mathbf{p}_j \right\| \} \quad \text{s.t. } (C_j, c_j) \in \mathcal{A}' \tag{7.8}$$

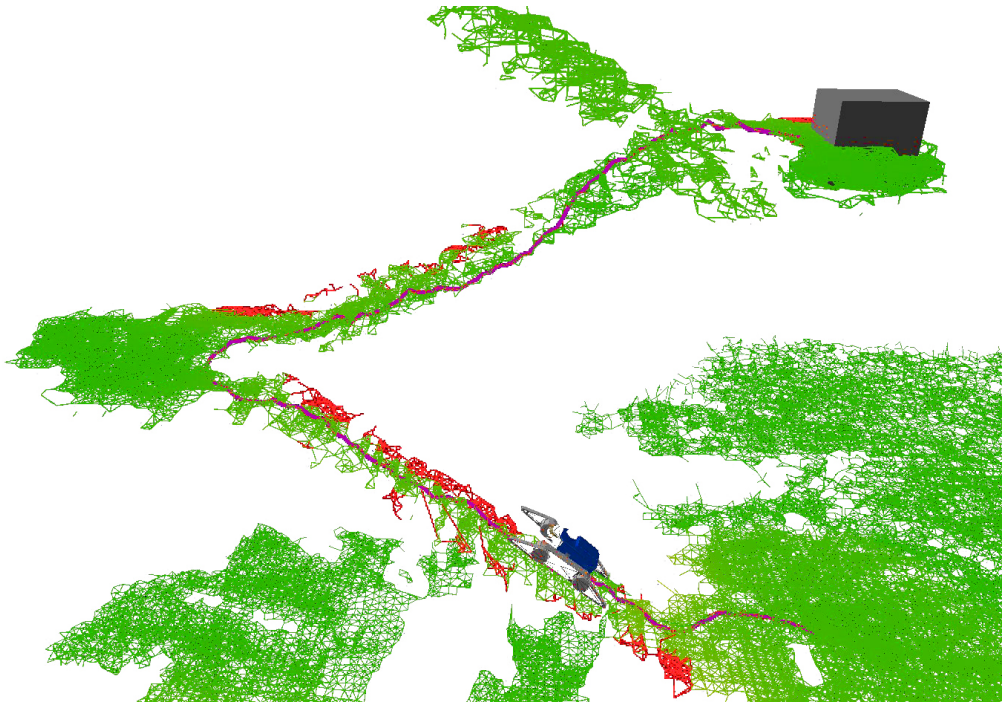

Figure 7.4. Robot overlayed with path on the weighted graph.

Then the set of inflated region points $\mathcal{B}$ is defined as follows:

$$\mathcal{B} = \{\mathbf{p} \ : \mathbf{p}{=}(\mathbf{p}'_x, \mathbf{p}'_y, \mathbf{p}''_z)^T, \exists(C, c){\in}\mathcal{A}', \mathbf{p}'{\in}C, \mathbf{p}''{=}u(\mathbf{p}')\} \tag{7.9}$$

We can note that the estimation of the inflated regions preserves a 3D discrete representation (see Figure 7.4).

Connectivity and traversability graph

Let $G(\mathcal{N}, \mathcal{E})$ be the graph whose nodes are all the points in $\mathcal{A}^\star$ and whose edges $E \in \mathcal{E}$ are defined as follow:

$$E(\mathbf{p}_i, \mathbf{p}_j) = \begin{cases} true & \text{if } \mu(\mathbf{p}_i - \mathbf{p}_j) \leq \eta \\ false & otherwise \end{cases} \tag{7.10}$$

Here $\mu(\Delta\mathbf{p}) = \left\|\Delta\mathbf{p}^T \cdot (1, 1, \delta)^T\right\|$, η is set equal to half the robot length and $\delta = \frac{\eta}{\Delta z_{max}}$ is set according to both the robot morphology and the robot overcoming capabilities. Here Δz_{max} is the maximum obstacle height surmountable by the robot.

The graph G is further weighted to build the traversability graph wG, where edge weight includes factors such as density, roughness, clearance. Traversability analisys and computation of these weights is discussed in the next sections.

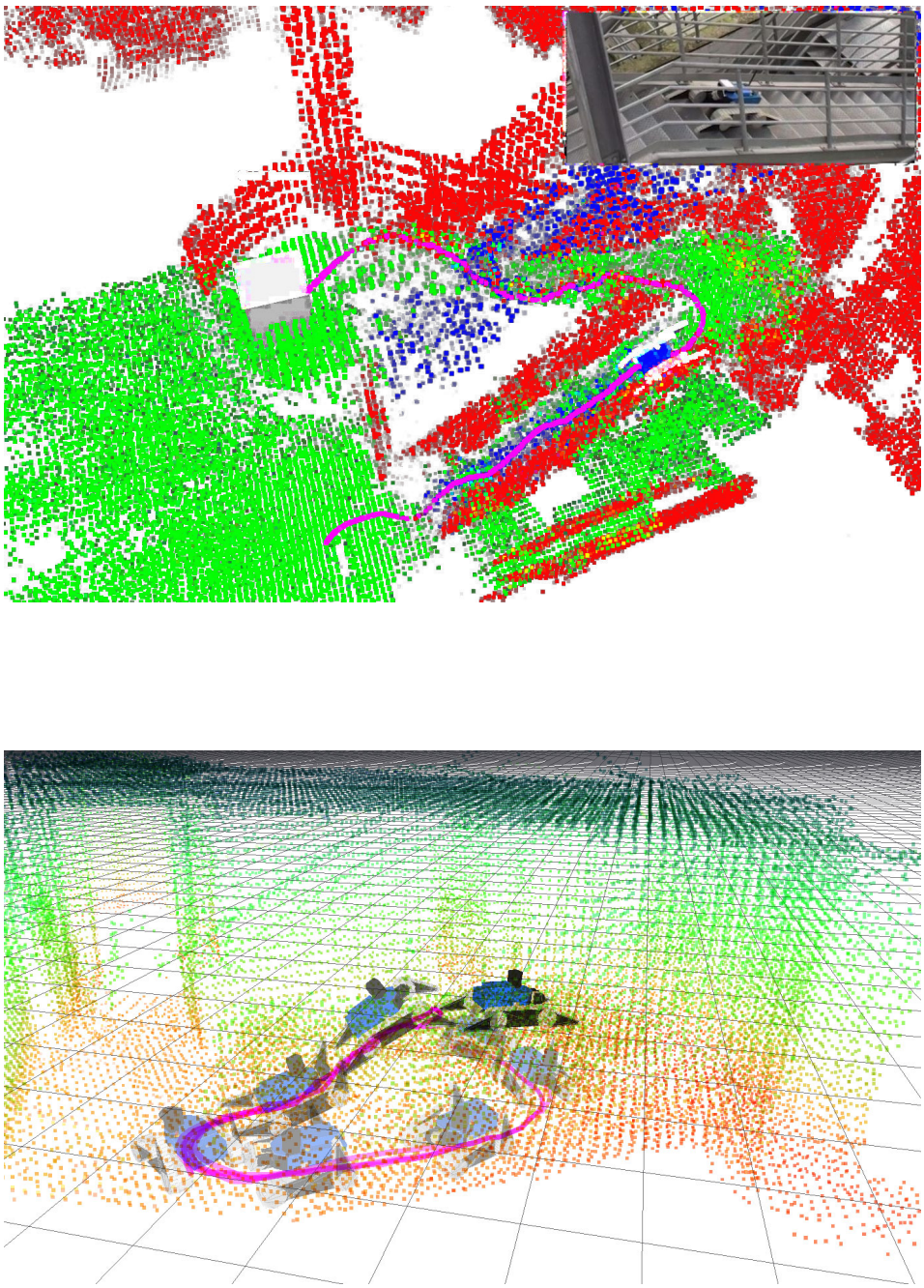

Figure 7.5. Above: segmentation of the fire escape stairs, with path drawn in magenta, and detail of the robot climbing the stairs (from Fire Escape stairs experiment). Below: a composed path of the robot from the gallery up to the top of the ramp and the bridge (from the Full 3D experiment). It is interesting to note that the robot passes under the gallery still following its path, up to the goal

7.3 Traversability analysis

7.3.1 Geometric features and segmentation

Point cloud segmentation of large scenes, for the purpose of robot planning, requires robust and real-time evaluation of geometric features that can establish the meaningful components of the environment in terms of traversability. A point cloud can be locally considered to be the discrete representation of a surface patch. Despite this, point clouds are unorganized point sets, and differential operations are not easily performed. The only effective structure to base geometric features computation is the neighborhood of a point. For the computation of normals this is quite straightforward. Indeed, given the neighborhood $\mathcal{N}_m(\mathbf{p})$ of a point, having cardinality m, where $\mathbf{p} = (x, y, z(x, y))^\top$, $z : \mathbb{R}^2 \mapsto \mathbb{R}$, the computation of the normal amounts to the computation of the plane $\pi = (\mathbf{n}, d)$ tangent to $\mathbf{p}$, and minimizing the distance of each point $\mathbf{p}_i \in \mathcal{N}_m(\mathbf{p})$, under the constraint $\|\mathbf{n}\| = 1$. This turns out to be the vector $\hat{\mathbf{u}}$ corresponding to the smallest eigenvalue λ in D, given $\mathbf{V}\ \mathbf{D}\ \mathbf{U}^\top = (\mathcal{N}_m(\mathbf{p}) - (\mathbf{p} \otimes \mathbf{1}_m))^\top$, where $\otimes$ is the Kronecker product. Then for $\pm\hat{\mathbf{u}}$ the sign compatible with view point *view*, namely the sign ensuring smallest angle for $\cos^{-1}(view - \pm\hat{\mathbf{u}})/\|view - \pm\hat{\mathbf{u}}\|$, is chosen. Where *view* is assumed to be directed toward the viewer. Then the sought for normal $\mathbf{n}$ at $\mathbf{p}$ is taken to be $\hat{\mathbf{u}}$ with the chosen sign. This method, being subject to the vantage point cannot provide an instantaneous correct assessment of the normals of an object, therefore in several cases a number of conflicts need to be handled in the merging and integration of the point cloud features.

For the curvature computation, on the other hand, the problem is more involved, since normal curvature, principal curvatures and the Gaussian and mean curvatures obtained by the principal ones, are defined by the coefficients of the first and second fundamental forms, which are essentially differential operators.

To alleviate the difficulty of computing derivatives on an unorganized point set, many researchers (see [Ber+94; Aga+05; Kal+07]) resorted to statistical computation of the second fundamental form, via some suitable approximation. These statistical approximations capture somehow the curvature variation but often are not enough discriminative. One of the most well known approximation is the one introduced by Pauly in [Pau+02] based on the above computation of the normal, but using the symmetric form, which actually returns the covariance of the neighborhood, and the eigen-decomposition of this last. Then the surface variation is given by $\lambda_1 / \sum_{i=1}^{3} \lambda_i$ with $\lambda_1 \leq \lambda_2 \leq \lambda_3$. This quantity varies between 0 and 1/3, and has the property of being high if the region is highly curved and it is small if the region is flat, however does not capture the principal curvatures and the induced classes. For these reasons we discriminate between close and far regions for segmentation, providing a more

accurate estimation of features for points with a distance $\|\mathbf{p}, Robot\| < 3m$ and a light feature space for further points. For far points we consider both normals and curvature variation as defined above [Pau+02], which is adequate to characterize large regions and classify major obstacles. Note that for far points we compute the light features on the integrated map Map_t at time t, and consider large clusters of points with mean radius of $0.2m$.

On the other hand for the region right in front of the robot a more accurate evaluation, taking into account the classes induced by the principal curvatures, needs to be considered. Let $H \in \mathbf{R}^3$ be the point set with distance $< 3m$ from the robot. This set is partitioned into clusters $C_i \subset H$ with centroid $\mathbf{c}_i \in H$, such that each cluster is not greater than 50 points, and not smaller than 7 points. Then, given the normal to $\mathbf{c}_i$, computed as indicated above, the neighborhood is rotated and translated so that $\mathbf{n}$ is transformed to $\mathbf{n}_z$ parallel to the z-axis of the reference frame of the robot and $\mathbf{c}'_i = \mathbf{T}(\mathbf{c}_i) = (0, 0, z(0, 0))$, with $\mathbf{T}$ the transformation. Consider a point $\mathbf{q} \in \mathbf{T}(C)$, $\mathbf{q} = (x, y, z(x, y))$, the projection of the vector $\mathbf{cq}$ on the normal $\mathbf{n}_z$ is, from Taylor expansion:

$$\begin{aligned}(z(x + dx, y + dy) - z(x, y))\mathbf{n}_z = ((z_x dx + z_y dy)+ \\ (1/2)(z_{xx}dx^2 + 2z_{xy}dxdy + z_{yy}dy^2) + o(dx^2 + dy^2))\,\mathbf{n}_z\end{aligned} \tag{7.11}$$

here dx and dy are the differentials w.r.t. the origin and are less than 0.2, by the choice of the cluster, then the last term can be ignored, and also $z_x\mathbf{n} = 0$ and $z_y\mathbf{n} = 0$. Furthermore, since $\mathbf{c}$ is centered at the origin we obtain: $2z(dx, dy)\mathbf{n} = (z_{xx}dx^2 + 2z_{xy}dxdy + z_{yy}dy^2)\mathbf{n}$ as the projection of the point $\mathbf{q}$ on the transformed normal, and this is, indeed, the second fundamental form. Therefore a principal direction is toward the point $\mathbf{q} \neq \mathbf{c}$ that maximizes this projection and the other principal direction is toward the point $\mathbf{q}' \neq \mathbf{c}$ that minimizes this projection. We can see that in this formulation the sign of the principal curvatures is maintained and the induced classes (elliptic, planar, paraboloid, ellipsoid [Abb+06]) for the points can be defined.

We have thus obtained two sets of features: (1) the light ones specified by surface variation and normals, on the integrated map Map_t, for points far from the robot, and using large and possible overlapping neighborhoods, and (2) the accurate ones, specified by the principal curvatures κ_1 and κ_2 and normals, on the scan map $Scan_t$, for points close to the robot, and using small clusters obtained by partitioning the close point set H.

Segmentation then is initialized with histogram distribution of normals and curvatures to highlight the sought for classes: *walls*, *terrain*, *surmountable obstacle*, and *stairs/ramps*. Further, clusters are refined and suitably combined according to the energy minimization algorithm illustrated in [Men+14].

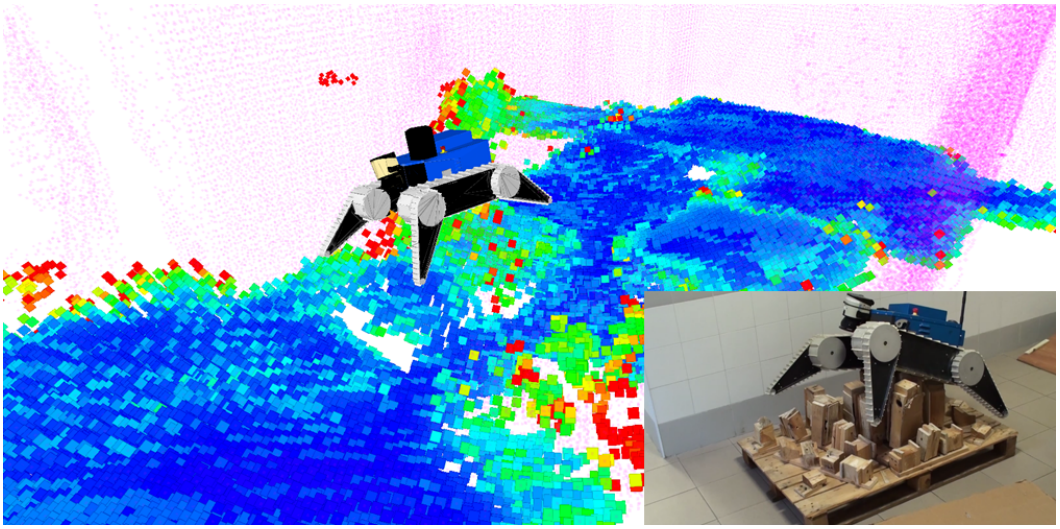

Figure 7.6. Traversability map $TMap$ computed real-time while the robot is traversing a home-made rubble pallet, in the lab. The colors indicate the traversabilty cost, from minimal cost (dark blue) to maximal cost (dark red).

7.3.2 Traversability in clutter

In this section we describe the method proposed to assess the traversability of the environment. Traversability is defined as a composite index depending on elevation statistics (e.g., height difference, slope, discontinuity, terrain roughness), physical properties of the terrain (e.g., hardness, slippery) and robot mobility (e.g., wheels rather than tracks, or legs, maximum superable step, steering efficiency). In the proposed approach traversability is assessed with a cost function, which takes into account the following geometrical features of the surrounding environment: (1) terrain roughness; (2) terrain classification, based on both terrain slope and robot locomotion capabilities; (3) obstacle clearance and (4) point cloud density.

This cost function has been defined as follows. Let $SMap_t$ be the segmented point cloud, at time t, introduced in subsection 7.3.1. Let $\mathbf{p} \in SMap_t$ a point of the point cloud. The point-wise traversability cost function $w : SMap \mapsto \mathbb{R}$, returning the static traversability cost of the point $\mathbf{p} \in SMap_t$, is given by

$$w(\mathbf{p}) \quad := w_L^{(\mathbf{p})} \cdot \left(w_{CL}^{(\mathbf{p})} + w_{Dens}^{(\mathbf{p})} + w_{Rough}^{(\mathbf{p})} \right) \tag{7.12}$$

Here the term $w_L^{(\mathbf{p})}$ represents the contribution to the overall cost of $\mathbf{p} \in SMap_t$, computed according to the segmentation of the local map. This cost term is lower for points belonging to clusters labeled as *ground*, while increases for points belonging to clusters, which are more difficult to traverse, such as those labeled with *stairs* or *ramps*. $w_{CL}^{(\mathbf{p})}$ is the cost contribution provided by the obstacle clearance. Points belonging to clusters which are relatively close to obstacles, inflated by the inscribed

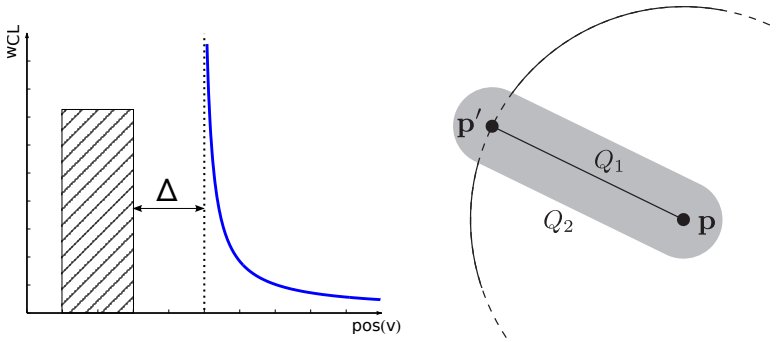

Figure 7.7. On the left a plot of the term w_{CL} as a function of the vertex position. The dashed region represents an obstacle, and the continuous blue line is the cost. On the right the boundary of a point neighborhood, with highlighted the two sets Q_1 and Q_2.

radius of the robot, have been assigned with a higher cost, since they are unreachable for the robot. This cost term is computed as follows

$$w_{CL}^{(\mathbf{p})} = \frac{1}{\max\left\{\min\left\{||\mathbf{p} - \mathbf{p}'||\right\}, \Delta\right\} - \Delta} \tag{7.13}$$

for all $\mathbf{p}' \in SMap_t$, such that $\mathbf{p}'$ belongs to clusters labeled either as *wall* or *ceiling*. Here Δ is the inflated radius (see Figure 7.7). Note that this approach is similar to the artificial potential fields methodology, proposed in [Rim+92]. $w_{Dens}^{(\mathbf{p})}$ represents the contribution to the overall cost of $\mathbf{p} \in SMap_t$, computed with respect to the point cloud density, as follows

$$w_{Dens}^{(\mathbf{p})} = \frac{|\mathcal{N}_r(\mathbf{p})|}{\frac{4}{3}\pi r^3} \tag{7.14}$$

Here $|\mathcal{N}_r(\mathbf{p})|$ is the cardinality of the set of points belonging to the neighborhood of $\mathbf{p} \in SMap_t$. High values of $w_{Dens}^{(\mathbf{p})}$ are assigned to points $\mathbf{p} \in SMap_t$ which are relatively close to terrain regions not correctly segmented, due to unevenness of the terrain, holes or lack of laser scan data. This cost term has the effect of deterring navigation near such areas. $w_{Rough}^{(\mathbf{p})}$ measures the terrain roughness. Roughness is estimated by computing the average distance of the outliers from the plane fitted on the points belonging to the neighborhood $\mathcal{N}_r(\mathbf{p})$ of $\mathbf{p} \in SMap_t$. By applying the defined traversability cost function $w(\mathbf{p})$, $\forall \mathbf{p} \in SMap_t$, we obtain the traversability map $TMap_t$ of the environment.

7.4 Distance Transform on Weighted Graph

By running an all-pairs shortest path algorithm such as Dijkstra's algorithm we obtain all the shortest paths from a source to all destinations, or equivalently, since the graph is undirected, all shortest paths from all sources to a single destination, by specifying the goal vertex as the source node for the Dijkstra's algorithm.

Let $p \rightsquigarrow q$ be a *path*, namely a list of unique vertices $v_1, \ldots, v_n$, where $v_1 = p$, $v_n = q$, and $E(v_i, v_{i+1}) \in \mathcal{E}$, where v_i is the predecessor of v_{i+1} as computed by the Dijkstra's algorithm.

Let $\mathit{distance}(p \rightsquigarrow q)$ be the *path length*, which is given by the sum of the weight of each edge in the path:

$$\mathit{distance}(p \rightsquigarrow q) = \sum_{i=1}^{n-1} w(v_i, v_{i+1}) \tag{7.15}$$

We label each vertex v in the graph with the value of $\mathit{distance}(v, q)$, where q is the goal vertex.

The distance label is then used in the heuristic function, see algorithm in Figure 7.2.

Chapter 8

Empirical Evaluation

It has been thoroughly discussed how the use of physics-based motion planning provides an enormous simplification in the design of a motion planner for a mobile robot. However, this type of motion planning also comes with a big computational requirement.

By providing search heuristics, we bias the search towards more promising states. We do so by encoding the domain knowledge related to robot kinematics and dynamics in the form of search heuristics. Those heuristics are simpler to design than ordinary equations of motion.

The heuristic doesn't need to be geometrically accurate, because the process of expanding the search tree uses a physics engine, and any invalid state will be discarded, such that the search can backtrack. Example: the heuristic encoded the strong belief that some path would lead to the goal, but instead it is a dead end; the search will expand the motion tree towards that point, up to the point an alternative path becomes more promising.

Yet, the search heuristic is able to reduce the time required to find a solution to the physics-based motion planning problem, by 1 or 2 orders of magnitude.

8.1 Experiments

In this section we present the experiments performed to evaluate the accuracy of the 3D map segmentation and the generated path feasibility, on the 3D labeled map. The segmentation has been evaluated with respect to the time required to both segment and label the point cloud, and with respect to its accuracy. The segmentation accuracy has been measured by manually assigning a label to each cluster and by comparing these labels with the labels returned by the segmentation process. The percentage of the clusters correctly classified states the accuracy of the segmentation. The feasibility of the 3D path is measured in terms of the time required to build the graph structure representing the traversable areas of the environment, with respect to the value of the path smoothing algorithm, and in terms of the time needed to the robot to complete the path, namely the journey time. This time has been taken by hand with a stopwatch. To test the effectiveness of the autonomy capabilities of the overall system three different scenarios have been considered.

8.1.1 The Italian Fire Fighters rescue training area in Prato (IT)

First experiment has been performed at the Italian Fire Fighters rescue training area in Prato (IT), during the final review meeting of the EU project NIFTi (247870). In this experiment the robot traversed the harsh terrain of the rescue area, though not climbing any ramp or stairs, overcoming small obstacles, following different paths toward several target poses, manually posted by an operator on the 3D map. Figure 8.1 illustrates the segmented map with the inflated region in red, the generated path toward the goal (the gray cube) and screen shots of the robot following the path. The dialog window accommodates automatic goal generation, which is not treated here. Table 8.1 reports the accuracy of the segmentation of the 3D map of the environment and the feasibility of the path, generated on the 3D segmented and labeled map. Here the values are averaged over eleven trials to obtain the same goal position. On average, the map had about 52 thousand points.

8.1.2 Fire Escape stairs

The second experiment has been performed on the fire escape stairs of the Department. The goal is located on the landing at the end of the second stairway, and when the robot is up the goal is moved to the ground. The robot has simply to climb up the stairs to reach the goal and then turning on its self and step down back to reach the new selected goal. In some trials the robot is actually not able to get down autonomously, because localization problems might arise when the robot is turning around itself. Figure 7.4 shows the robot climbing the fire escape stairs, visualizing only the graph where the inflated regions are in red. Figure 8.2, left panel, shows

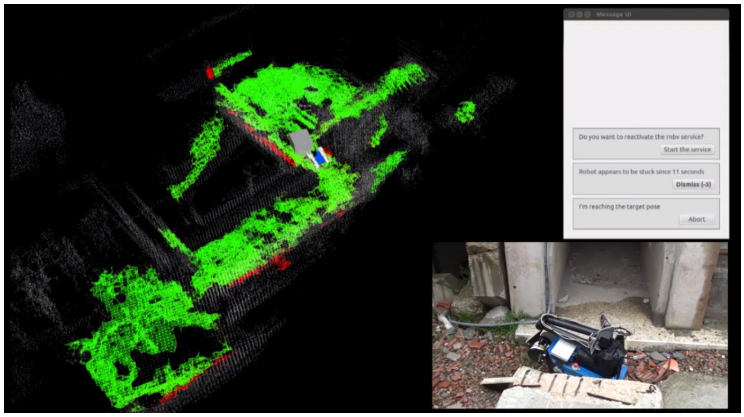

Figure 8.1. The Italian Fire Fighters rescue training area in Prato (IT)

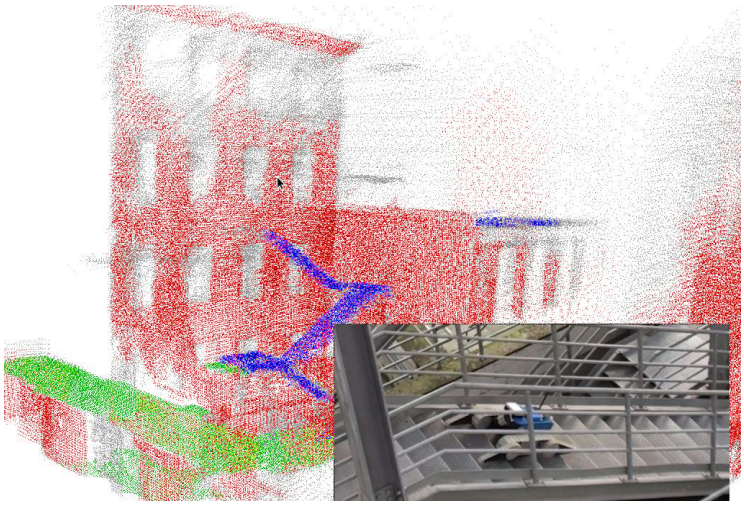

Figure 8.2. Fire Escape stairs

the segmentation of the courtyard with the fire escape stairs and, in the window up right, a detail of the robot climbing the stairs. Table 8.1 reports the average segmentation time, first table; the percentage of the clusters correctly classified, middle table; and in the last table the average time needed to generate the 3D path on the stairs, with respect to the value of the path smoothing and the journey time. On average the map has 41 thousand points.

8.1.3 Full 3D designed scenario

. The third experiment has been designed purposefully to test the effective 3D autonomy, within an environment whose structure is composed of multiple levels. A gallery, surmounted by a ramp, extended with a bridge, is built and it lies between a step on the floor and a wall of bricks (see Figure 8.3). The robot has to first traverse the gallery and then climb the ramp passing over the gallery and continue up to

Table 8.1. Data from the three experiments

	Point Cloud Segmentation		Graph	Path Smoothing	Journey
	Time (s)	Accuracy	Time (s)	k	Time (s)
Prato experiment	0.27	0.86	1.19	0.2	357
Fire escape stairs	0.32	0.91	1.45	0.2	210
3D designed	0.12	0.97	0.92	0.2	143

the end of the bridge. The goal is located at the end of the bridge. The robot is constrained to pass under the gallery by obstacles. It is interesting to note that in this experiment the space is fully 3D since the robot has to face both the levels: under and over the construction (see Figure 8.2, right panel). Table 8.1 reports, for three trials, the accuracy of the segmentation of the scenario, as well as the feasibility of the complex 3D path. On average the map has 15 thousand points.

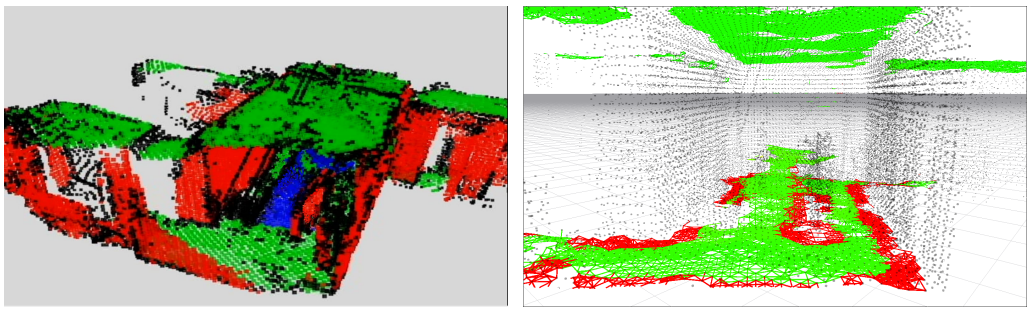

Figure 8.3. On the left segmentation of the third experiment map. On the right a screen shot of the generated graph.

8.1.4 Computational time performance

In addition to the previous experiments, several tests have been performed to evaluate the computational performance of both the segmentation and the path planning algorithm, with respect to the size of the point cloud. During these tests the robot was teleoperated, so as to explore a wide area and to acquire, at real-time, the point cloud. The goal was fixed and the robot computed a new path every new point cloud. The results, in terms of time of computation, of the different algorithms are reported in Figure 8.4. Note that the robot was able to elaborate a point cloud composed of 55000 points and to generate a path in less than $3.5s$.

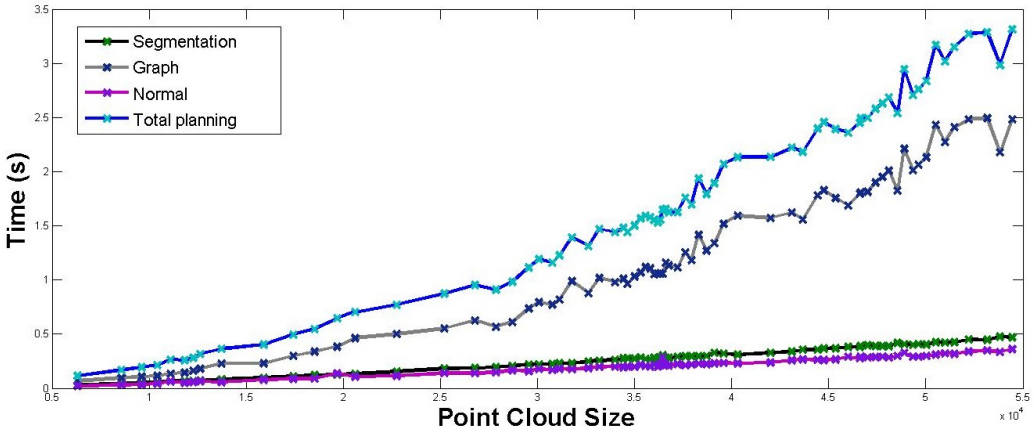

Figure 8.4. Time of computation for both the segmentation and the path planning algorithm with respect to the size of the point cloud.

8.2 Limitations and future improvements

One limitation of the approach discussed in this thesis, is the fact that the answer returned by this motion planning "oracle", does not give us any information of how likely (or how difficult?) is for a robot to execute a certain motion.

The oracle only showed us one way to achieve the goal, but perhaps he was very lucky to successfully perform that motion.

If the motion is about driving the robot on a smooth terrain, then no problem, it will be fairly easy for any trajectory tracking controller to follow that motion, and there is big margin for execution errors.

But if the motion is about surmounting some obstacle, that might lead the robot to risk too closely of tipping over or falling into some cliff, then it becomes more risky.

In the future, when massive parallel setups of CPUs might be widely available, it will be possible to use an embarrassingly parallel setup to run many motion planning searches in parallel, not only to speed up the search, like said in [Şuc+09; Dev+11], but also to have an estimate of the probability of the success of performing some motion, like in [Ich+17].

Appendix A

Point cloud mapping in dynamic environments

Scanning the environment is a continuous process which, due to the presence of dynamic obstacles, needs to be periodically repeated in order to ensure that the built map is up to date, and the robot can safely navigate on its own. When the environment is static, mapping can be performed by simply registering and adding the 3D scan to the accumulated map. However in a dynamic environment special procedures are needed for merging the new 3D scan with the accumulated map. Not performing the dynamic obstacles removal on the previous map would suffer of an error in the ICP registration, and of course would result in a map cluttered by outdated dynamic obstacles, which in the end renders the autonomous navigation of the robot impossible.

A.1 Introduction

3D map update with dynamic obstacles detection is a prerequisite for path planning and autonomous navigation of mobile and tracked robots. Despite its noteworthy importance for navigation and planning, this problem is only recently receiving the attention it deserves (see [Pom+14; Kum+13; Azi+12; Mor+11; Fok+10]).

When the problem of dynamic obstacles is not addressed, moving objects bring artifacts and spurious trails in the map, so that in place of the object at its current position the map produces a stream of positions. If these spurious trails are not promptly removed and the map of the environment is not correctly updated, the robot will no longer be able to plan and execute navigation tasks, after a short time period. Moreover, the presence of these artifacts causes an incorrect feature-matching, leading to scan misalignments, which affect the robot localization accuracy.

The proposed method works in real time and also addresses a new representation of the voxel space, inspired by ray casting methods. The main advantage of our approach is that the update can be very well used during navigation and planning tasks. Furthermore, we show that the sampling necessary to work in real time can be pushed up to a specified limit, in order to give reliable and approximately correct updates.

The conventional approach to update dynamic obstacles in 3D maps, is to detect them and then either treat them as outliers [Hah+03; Häh+03; Ser+05; Hol+08] or track them separately using off the shelf multi-target tracking algorithms [Wan+03; Roh+08]. The main drawback of the outliers removal based methods is that they throw the baby out with the bathwater and any dynamic object is removed together with the outliers. This problem is partially addressed in [Ser+05; Hol+08], where the authors propose two distinct representations of the environment (see also [Sch+12]), differentiating static and dynamic objects. However, dynamic objects are assumed to both move at a good speed, in order to be detected, and to be not too close to the static parts of the environment [Ser+05], required to be well structured for accurate robot localization.

More recently, [Mor+11] provides a summary of the approaches to navigation in 3D dynamics environments considering both visual and laser based SLAM, though the main target is people tracking. Actually two main research streams on dynamic map have emerged. One more focused on detection of dynamic objects, inspired by the ideas of background subtraction and exploiting the difference between two scans, as for example [Azi+12; Ryd+11], or keeping a history of past local maps to compare with the current map [And+11]. The other more concerned with dynamic objects tracking [Kum+13] and establishing their velocity [Sch+12; Kae+12; Pom+14], for prediction or for assessing long lasting structures in 3D maps [Pom+14]. Details

on map updating, though, are lacking in all these approaches. Despite most of the approaches are real time, using sparse sampling of the map, how this can affect the update correctness is not clarified.

On the other hand we present a detailed method for updating the map, taking into account objects displacement at scan time, and show the advantage of our approach for planning and navigation. For example, differently from [Pom+14] we are able to take into account the whole last scan and compare it with the current integrated map. Therefore we can detect the motion at any point in the environment that can be reached by the laser beam. Furthermore, rather than specifying a probability density to discriminate between static and dynamic objects, we give an algorithm that discriminates between spurious trails and objects, in so keeping a continuously updated map. The proposed method builds upon the well established ICP scan-matching algorithm, for both 3D map building and robot motion estimation [Pom+13], which is very robust in the presence of outliers, and we exploit the fact that new point cloud measurements are registered against the union of the previously acquired and registered scans.

The chapter is organized as follows. In section A.2 we formulate the problem, introducing some notation too. section A.3 describes the 3D laser scanner measurement model used for dynamic object removal. section A.4 describes in details the proposed approach.

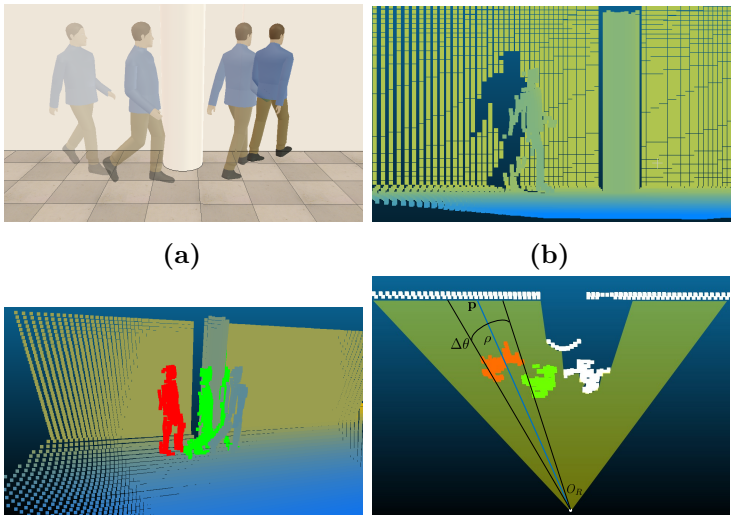

Figure A.1. The above sequence shows the typical problem encountered in 3D maps when not correctly updated; (a) 3D model of a simple scenario with a moving person; (b) point cloud S_t of the scene, at time t; (c) new point cloud S_{t+1}, at time $t+1$, showing the trail of points of the moving person; (d) top view of the scene subtended by $\Delta\theta$ and $\Delta\phi$ which should be empty if the map is correctly updated.

A.2 Problem Statement

Let us consider an instance of the 3D map updating problem, as illustrated in Figure A.1. Figure A.1a shows the scene, a 3D model of a person walking around. Figure A.1b and Figure A.1c show the point cloud S_t and S_{t+1}, at time t and $t+1$, referred to a global reference frame G_Ω, with $\Omega \subseteq \mathbb{R}^3$, the space domain, and $t \in \mathbb{N}$, the time domain.

S_{t+1} results from registration and filtering operations performed by an ICP-based algorithm [Pom+13] to merge the set of points $Scan_{t+1} = \{\mathbf{p}\}_{j=1}^{K} \subset \mathbb{R}^3$, taken by a 3D depth sensor at time $t+1$, with the 3D map S_t, built so far.

Note that, differently from the cumulated map S_t, each measure $Scan_t$ is expressed in a local reference frame G_R, namely, the reference frame of the 3D depth sensor. The origin O_R of G_R coincides with the sensor optical center.

The ICP algorithm also estimates the ego-motion and provides the rigid transformation $\mathbf{T}_R^\Omega(\mathbf{p}) = \mathbf{R}_R^\Omega \mathbf{p} + \mathbf{t}_R^\Omega$, at time $t+1$, between the sensor reference frame G_R and the global reference frame G_Ω, so as to merge $Scan_{t+1}$ with the current map S_t.

Going back to Figure A.1, while the person is moving a new measure $Scan_{t+1}$, at time $t+1$, is acquired by the 3D depth sensor. The ICP-based mapping algorithm gathers the points of $Scan_{t+1}$, does the above mentioned operations and merge them with S_t returning the accumulated point cloud S_{t+1} of the scene (see Figure A.1c). As illustrated in Figure A.1c, the new point cloud S_{t+1}, at time $t+1$, has not removed the points echoing the person previous positions, as the algorithm does not know that the specific object is dynamic and therefore characterized by constant change.

The problem that we face in the following is exactly that of removing these points, to correctly update the map, without any prior knowledge about what is in the scene.

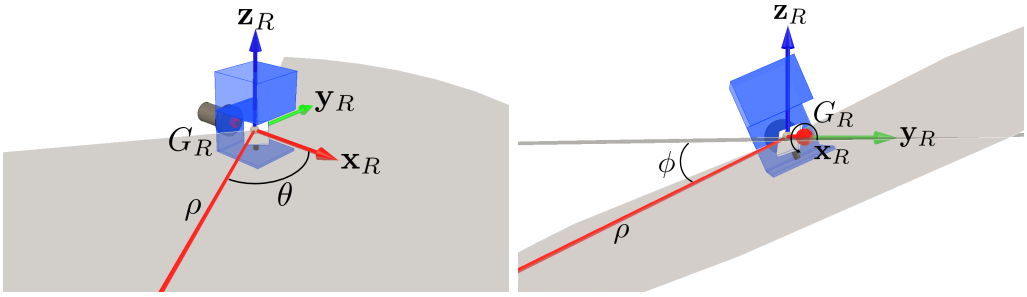

Figure A.2. Basic LIDAR system mounted on a servomotor.

A.3 3D Laser scanner and space model

A typical 3D representation of a map S_t is given by a voxel space in Cartesian coordinates, with resolution specified according to context and to memory requirements. Here we introduce a representation of the voxel space in spherical coordinates and derive the resolution by the properties of the sensor.

We consider the system illustrated in Figure A.2 with the origin O_R at the center of the mirror. The $\mathbf{x}_R$ axis of the system reference frame G_R is the roll axis, facing forward, around which the servomotor spins the LIDAR sensor. We denote with $\theta \in [\theta_{min}, \theta_{max}]$ the angle between the axis $\mathbf{x}_R$ and the laser pulse, reflected by the mirror, and with $\phi \in [\phi_{min}, \phi_{max}]$ the roll angle of the LIDAR system. As the values of θ and ϕ change, within their range, the unit vector $\widehat{\mathbf{d}}_{\theta,\phi} = (\cos\theta,\ \sin\theta\cos\phi,\ \sin\theta\sin\phi)^\top$ specifies the laser pulse directions.

Now, a point $\mathbf{p} = (x,\ y,\ z)^\top \in Scan_t$, acquired by the rotating LIDAR sensor, is represented in spherical coordinates $(\rho,\ \theta,\ \phi)^\top$ as $\mathbf{p} = \rho\widehat{\mathbf{d}}_{\theta,\phi}$, with $\rho = \sqrt{\mathbf{p}^\top\mathbf{p}}$, $\theta = \tan^{-1}\left(\sqrt{y^2+z^2}/x\right)$ and $\phi = \tan^{-1}(z/y)$. To abbreviate the transformation from Cartesian to spherical coordinates we shall use the functions f_ρ, f_θ and f_ϕ.

Rotating LIDAR systems emit laser pulses within discrete narrow laser-beams $\Delta\theta$, at discrete spinning angle intervals $\Delta\phi$. If we denote with N_l the number of laser pulses emitted by the LIDAR within a single scan and with N_s the number of scans acquired during an entire revolution period of the sensor, then

$$\Delta\theta = \frac{\theta_{max} - \theta_{min}}{N_l} \quad \text{and} \quad \Delta\phi = \frac{\phi_{max} - \phi_{min}}{N_s}. \tag{A.1}$$

$\Delta\theta$ and $\Delta\phi$, in Equation A.1, specify the minimal resolution of the scan and the spin of the rotating LIDAR sensor, respectively.

Now, given $\Delta\theta$ and $\Delta\phi$, the minimal scan and spin resolutions of the rotating LIDAR system, N_l the number of laser pulses per single scan, N_s the number scans per revolution period, and D the maximum scan distance, we define the representation space for both the scan and the map, in spherical coordinates, with

$\rho \in \mathbb{R}^+$, $\theta \in [0, 2\pi]$ and $\phi \in [-\pi/2, \pi/2]$. Accordingly, we specify the space resolution in terms of the following intervals:

$$\begin{array}{lll} P_i = [\rho_{i-1}, \rho_i) & \rho_i - \rho_{i-1} = \overline{\Delta\rho} & i = 1, \ldots, D \\ \Theta_j = [\theta_{j-1}, \theta_j) & \theta_j - \theta_{j-1} = \overline{\Delta\theta} & j = 1, \ldots, N_l \\ \Phi_k = [\phi_{k-1}, \phi_k) & \phi_k - \phi_{k-1} = \overline{\Delta\phi} & k = 1, \ldots, N_s \end{array} \tag{A.2}$$

such that $\overline{\Delta\theta} \geq \Delta\theta$ and $\overline{\Delta\phi} \geq \Delta\phi$.

Then we denote with $\mathbf{V}_{i,j,k}$ the region of space delimited by an interval in Equation A.2. Note that this space region is not cubic (in the space domain $\Omega \subseteq \mathbb{R}^3$) but has the form of a spherical wedge (see Figure A.3.

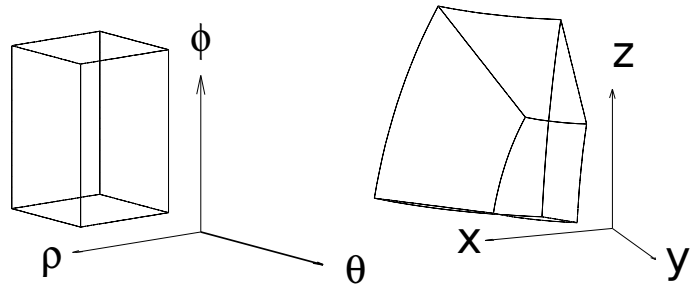

Figure A.3. Spherical wedge $\mathbf{V}_{i,j,k}$ delimited by the intervals P_i, Θ_j and Φ_k in spherical coordinates on the left and in cartesian coordinates on the right.

Typically the laser beam opening angle $\Delta\theta$, the spinning resolution $\Delta\phi$ and the distance $\Delta\rho$, of a 3D depth sensor are subject to errors and noise. However it turns out that the most commercialized sensors for surface inspections have an accuracy of the order of millimeters. Therefore, errors in the sensor model can be considered as negligible.

Note that the chosen spherical coordinate system has poles facing forward and backward. This choice is dictated by the mechanical construction of the rotating LIDAR. However, it introduces an ambiguity for points close to the roll axis $\mathbf{x}_R$ in determining the value of ϕ and consequently the voxel in which they fall. We deal with this ambiguity by merging the voxels $\{V_{i,j,k} : j = 1\}_{k=1}^{N_s}$.

A.4 Dynamic obstacle detection and updating

In this section we describe the detection of dynamic objects and the removal of their spurious trails from the cumulated point cloud S_t. The idea is to resort to the ray-casting technique to test ray-surface intersection for both volume estimation and points removal [Woo+05; Wei06].

A point $\mathbf{p} \in \mathbb{R}^3$, in Cartesian coordinates, belongs to a spherical wedge $\mathbf{V}_{i,j,k}$ and we denote it by $\mathbf{p} \in_f \mathbf{V}_{i,j,k}$ if the following conditions are satisfied:

$$f_\rho(\mathbf{p}) \in P_i \text{ and } f_\theta(\mathbf{p}) \in \Theta_j \text{ and } f_\phi(\mathbf{p}) \in \Phi_k \tag{A.3}$$

Here f_ρ, f_θ, f_ϕ denote the transformation from Cartesian to spherical coordinates (see section A.3).

Let S_t be the current cumulated point cloud, and let $Scan_{t+1}$ be the new point set acquired by the rotating LIDAR system, at time $t+1$. The set of points in $Scan_{t+1}$ satisfying condition Equation A.3 for a spherical wedge $\mathbf{V}_{i,j,k}$ is:

$$w_{i,j,k}^{Scan_{t+1}} = \{\mathbf{p} \in Scan_{t+1} \mid \mathbf{p} \in_f \mathbf{V}_{i,j,k}\} \tag{A.4}$$

Similarly, the set of points in S_t satisfying condition Equation A.3 for a spherical wedge $\mathbf{V}_{i,j,k}$ is:

$$w_{i,j,k}^{S_t} = \{\mathbf{p} \in S_t \mid \mathbf{T}_\Omega^R(\mathbf{p}) \in_f \mathbf{V}_{i,j,k}\} \tag{A.5}$$

Here $\mathbf{T}_\Omega^R$ is the transformation at time $t+1$, mapping a point $\mathbf{p} \in S_t$ from global frame G_Ω to the sensor reference frame G_R. As the indexes i, j and k vary, wedges $\mathbf{V}_{i,j,k}$ specify a spherical voxel space of both the point cloud S_t and $Scan_{t+1}$ [Sto+98]. This spherical voxel space is centered in the origin of G_R, at time $t+1$. The main advantage of this partitioning is to accelerate ray-casting. Note that the proposed approach focuses on the comparison between the scan $Scan_{t+1}$, at time $t+1$, and the point cloud S_t, as cumulated up to time t; this simplification makes detection immediate and can operate in real time.

More precisely, consider the wedges $V_{i,j,k}$ and $V_{i+h,j,k}$, with $\rho_{i+h} > \rho_i$, $h > 0$. By Equation A.4 and Equation A.5, there is one pair of sets of points in $Scan_{t+1}$, namely $w_{i,j,k}^{Scan_{t+1}}$, and $w_{i+h,j,k}^{Scan_{t+1}}$, and one pair of sets of points in S_t, namely $w_{i,j,k}^{S_t}$ and $w_{i+h,j,k}^{S_t}$, which satisfy condition Equation A.3, (see Figure A.5). Then the following facts can take place:

$$\begin{aligned}
&i. \quad w_{i,j,k}^{Scan_{t+1}} \neq \emptyset, \quad w_{i+h,j,k}^{S_t} \neq \emptyset \\
&\qquad\qquad\qquad\quad \text{and } \forall u.\ u < i,\ w_{u,j,k}^{Scan_{t+1}} = \emptyset; \\
&ii. \quad w_{i,j,k}^{S_t} \neq \emptyset, \qquad w_{i+h,j,k}^{Scan_{t+1}} \neq \emptyset, \\
&\qquad\qquad\qquad\quad \text{and } \forall u.\ u < i+h,\ w_{u,j,k}^{Scan_{t+1}} = \emptyset.
\end{aligned} \tag{A.6}$$

When the first fact is verified then points $w_{i,j,k}^{Scan_{t+1}} \subset Scan_{t+1}$ belong to a surface of an object which is occluding another surface object, previously scanned by the sensor,

and suitably accumulated in S_t. Therefore, motion is detected but no points removal is needed. If the second fact is verified then there are wedges $V_{u,j,k}$, $u < i + h$, for which there are sets $w^{S_t}_{u,j,k} \subset S_t$ not empty but the corresponding sets $w^{Scan_{t+1}}_{u,j,k} = \emptyset$, that is why the set of points $w^{Scan_{t+1}}_{i+h,j,k}$ is detected in $Scan_{t+1}$. Therefore, the points in $w^{S_t}_{i,j,k}$ are supposed to belong to the surface of an object that was occluding the surface detected by $w^{Scan_{t+1}}_{i+h,j,k}$, at time t, but is no more occluding it at time $t + 1$. Hence, points in $w^{S_t}_{i,j,k}$ are removed from S_{t+1} (see Figure A.4).

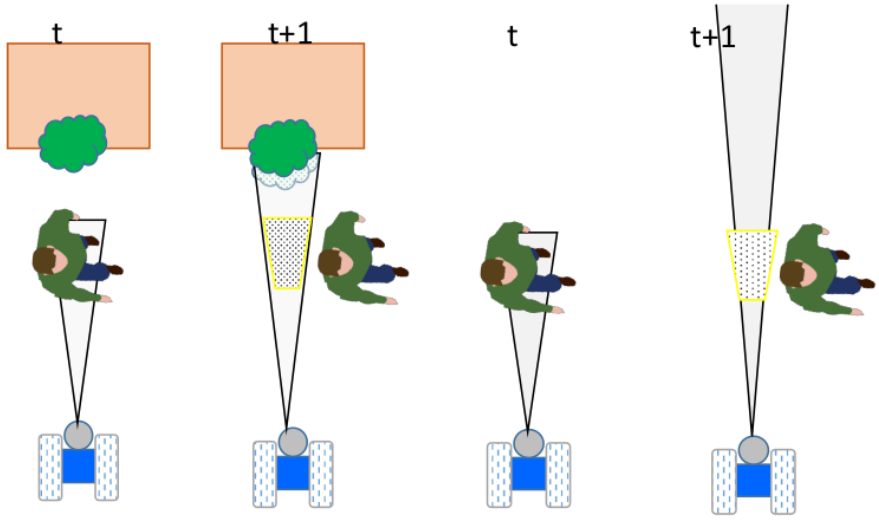

Figure A.4. (left) detecting the points to remove in presence of background and (right) failing to detect points to remove due to lack of background.

Note that the approach for points removal implicitly takes into account the robot ego-motion. Indeed, points in $w^{S_t}_{i,j,k}$ are transformed according to the transformation $\mathbf{T}^R_\Omega$ provided by the ICP algorithm at time $t + 1$ (see Equation A.5), and thus expressed in the frame of $w^{Scan_{t+1}}_{i,j,k}$, namely G_R; then, are partitioned into the spherical voxel space. Consequently, the ego-motion is compensated.

Trails of points belonging to a moving object cannot be removed when the distance between the robot and the background exceeds the maximum range of the sensor (see Figure A.4). In this case, the following fact holds:

$$iii. \quad w^{S_t}_{i,j,k} \neq \emptyset \text{ and } \forall u.\ w^{Scan_{t+1}}_{u,j,k} = \emptyset. \tag{A.7}$$

The lack of the background makes it impossible to identify the motion, this is a typical effect of point at infinity. To avoid this case it is enough to fix a maximal distance and assume that the wedge at that distance identifies a set of points at any time t. Under this condition the above item iii) reduces to the item ii) of Equation A.6.

However, the presence of noise in the measurements can lead to errors in the accumulated point cloud S_{t+1}. For example, an erroneous measurement can be

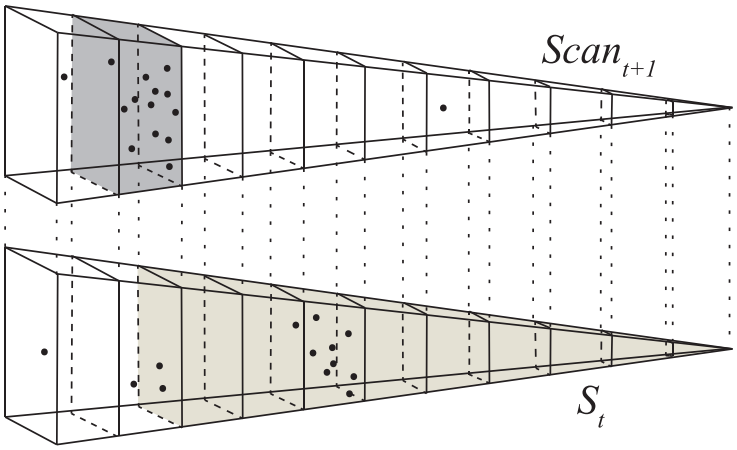

Figure A.5. Set of spherical wedges and $w_{i,j,k}$ of $Scan_{t+1}$ and S_t for fixed j and k. Highlighted in $Scan_{t+1}$ the occupied wedge, and highlighted in S_t the wedge and the points, which will not be merged into S_{t+1}. Observe that in $Scan_{t+1}$ some outliers appear, which could have affected the result if not treated correctly.

produced by the scan of a surface, when a part of that is reflective. This results in few points located very far from the scanned surface (see Figure A.5). A possible solution to this problem is to look at the cardinality $n_{i,j,k}^{Scan_{t+1}} = \left| w_{i,j,k}^{Scan_{t+1}} \right|$ of a set of points $w_{i,j,k}^{Scan_{t+1}}$ identified by the wedge $\mathbf{V}_{i,j,k}$. Therefore, let $i^\star$ be the index of the wedge $\mathbf{V}_{i^\star,j,k}$, such that

$$i^\star = \arg\max_i \left\{ n_{i,j,k}^{Scan_{t+1}} - i\epsilon \right\} \quad \text{with } \epsilon > 0 \tag{A.8}$$

If $i^\star > 0$, then $\mathbf{V}_{i^\star,j,k}$ corresponds to the first not empty voxel such that $w_{i^\star,j,k}^{Scan_{t+1}}$ has maximal cardinality. Otherwise, the region of space subtended by the intervals Θ_j and Φ_k is completely empty. The selection of $\mathbf{V}_{i^\star,j,k}$, on the basis of the above criteria, ensures that the voxels $\mathbf{V}_{i,j,k}$ occupied by very few points of $Scan_{t+1}$, including noisy measurements, are discarded. Therefore, S_{t+1} will contain all the points of $w_{i,j,k}^{Scan_{t+1}}$, for all i, and those of $w_{i,j,k}^{S_t}$ such that $i > i^\star$.

Bibliography

[Abb+06] E. Abbena, S. Salamon, and A. Gray. *Modern Differential Geometry of Curves and Surfaces with Mathematica, Third Edition*. 2006.

[Aga+05] G. Agam and Xiaojing Tang. "A sampling framework for accurate curvature estimation in discrete surfaces". In: *Visualiz. and Comp. Graph.* 5 (2005).

[And+10] Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. "Particle markov chain monte carlo methods". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 72.3 (2010), pp. 269–342.

[And+11] P. Anderson-Sprecher, R. Simmons, and D. Huber. "Background subtraction and accessibility analysis in evidence grids". In: *ICRA*. May 2011, pp. 3104–3110.

[Azi+12] A. Azim and O. Aycard. "Detection, classification and tracking of moving objects in a 3D environment". In: *IVS*. 2012, pp. 802–807.

[Bar+97] Jérôme Barraquand, Lydia E. Kavraki, Jean-Claude Latombe, Tsai-Yen Li, Rajeev Motwani, and Prabhakar Raghavan. "A Random Sampling Scheme for Robot Path Planning". In: *International Journal of Robotics Research* 16.6 (1997), pp. 759–774.

[Bar01] David Baraff. "Physically based modeling: Rigid body simulation". In: *SIGGRAPH Course Notes, ACM SIGGRAPH* 2.1 (2001), pp. 1–2.

[Ber+94] J. Berkmann and T. Caelli. "Computation of surface geometry and segmentation using covariance techniques". In: *PAMI* 11 (1994).

[Bib+04] P. Biber, H. Andreasson, T. Duckett, and A. Schilling. "3D modeling of indoor environments by a mobile robot with a laser scanner and panoramic camera". In: *IROS*. Vol. 4. 2004, pp. 3430–3435.

[Blo+11] Nico Blodow, Lucian Cosmin Goron, Zoltan-Csaba Marton, Dejan Pangercic, Thomas Ruhr, Moritz Tenorth, and Michael Beetz. "Autonomous semantic mapping for robots performing everyday manipulation tasks in kitchen environments". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2011), pp. 4263–4270.

[Bro+85] Rodney A Brooks and Tomas Lozano-Perez. "A subdivision algorithm in configuration space for findpath with rotation". In: *IEEE Transactions on Systems, Man, and Cybernetics* 2 (1985), pp. 224–233.

[Can88] John Canny. "Some algebraic and geometric computations in PSPACE". In: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM. 1988, pp. 460–467.

[Car+06] Joseph Carsten, Dave Ferguson, and Anthony Stentz. "3D Field D*: Improved Path Planning and Replanning in Three Dimensions". In: *International Conference on Intelligent Robots and Systems*. 2006, pp. 3381–3386.

[Che+94] Moez Cherif and Christian Laugier. "Dealing with Vehicle/Terrain Interactions when Planning the Motions of a Rover". In: (1994).

[Che99] Moez Cherif. "Motion planning for all-terrain vehicles: a physical modeling approach for coping with dynamic and contact interaction constraints". In: *Robotics and Automation, IEEE Transactions on* 15.2 (1999), pp. 202–218.

[Cho05] Howie M. Choset. *Principles of Robot Motion: Theory, Algorithms, and Implementation*. 2005, pp. 1–161.

[Col+13] Francis Colas, Srivatsa Mahesh, François Pomerleau, and Roland Siegwart. "3D path planning and execution for search and rescue ground robots". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Nov. 2013, pp. 722–727.

[Dai+88] Mike Daily, J Harris, D Keirsey, D Olin, D Payton, K Reiser, J Rosenblatt, D Tseng, and V Wong. "Autonomous cross-country navigation with the ALV". In: *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*. IEEE. 1988, pp. 718–726.

[Dev+11] Didier Devaurs, Thierry Siméon, and Juan Cortés. "Parallelizing RRT on distributed-memory architectures". In: *Robotics and automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2261–2266.

[Die+06] James Diebel and Sebastian Thrun. "An Application of Markov Random Fields to Range Sensing". In: *Advances in neural information processing systems*. MIT Press, 2006, pp. 291–298.

[Dij59] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs". In: *Numerische Mathematik* 1 (1959), pp. 269–271.

[Don+93] Bruce Donald, Patrick Xavier, John Canny, and John H. Reif. "Kinodynamic motion planning". In: *Journal of the ACM (JACM)* 40.5 (1993), pp. 1048–1066.

[Dor+13] Christian Dornhege and Alexander Kleiner. "A frontier-void-based approach for autonomous exploration in 3D". In: *Advanced Robotics* 27.6 (Apr. 2013), pp. 459–468.

[Dub57] Lester Eli Dubins. "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents". In: *American Journal of Mathematics* 79 (1957), pp. 497–516.

[ElH+13] Haitham El-Hussieny, Samy FM Assal, and Mohamed Abdellatif. "Improved backtracking algorithm for efficient sensor-based random tree exploration". In: *Computational Intelligence, Communication Systems and Networks (CICSyN), 2013 Fifth International Conference on*. IEEE. 2013, pp. 19–24.

[Eul68] Leonhard Euler. *Institutionum calculi integralis*. 1768.

[Far+04] H. Farid and E.P. Simoncelli. "Differentiation of discrete multidimensional signals". In: *Image Processing, IEEE Transactions on* 13.4 (2004), pp. 496–508.

[Far+98] Shane Farritor, Hervé Hacot, and Steven Dubowsky. "Physics-based planning for planetary exploration". In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)* 1.May (1998).

[Fer+05] Dave Ferguson and Anthony Stentz. "The delayed D* algorithm for efficient path replanning". In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE. 2005, pp. 2045–2050.

[Fer+14] Federico Ferri, Mario Gianni, Matteo Menna, and Fiora Pirri. "Point Cloud Segmentation and 3D Path Planning for Tracked Vehicles in Cluttered and Dynamic Environments". In: *3rd Workshop on Robots in Clutter - Perception and Interaction in Clutter*. 2014.

[Fok+10] Amalia F. Foka and Panos E. Trahanias. "Probabilistic Autonomous Robot Navigation in Dynamic Environments with Human Motion Prediction". In: *Int. J. of Soc. Rob.* 2.1 (2010), pp. 79–94.

[Gen99] Donald B. Gennery. "Traversability Analysis and Path Planning for a Planetary Rover". en. In: *Autonomous Robots* 6.2 (Apr. 1999), pp. 131–146.

[Gil+88] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. "A fast procedure for computing the distance between complex objects in three-dimensional space". In: *IEEE Journal on Robotics and Automation* 4.2 (1988), pp. 193–203.

[Gon+02] H. H Gonzalez-Banos and Jean-Claude Latombe. "Navigation Strategies for Exploring Indoor Environments". In: *The International Journal of Robotics Research* 21.10-11 (2002), pp. 829–848.

[Gra97] A Gray. "A Monge patch". In: *Modern Differential Geometry of Curves and Surfaces With Mathematica* (1997).

[Gun+13] Martin Gunther, Thomas Wiemann, Sven Albrecht, and Joachim Hertzberg. "Building semantic object maps from sparse and noisy 3d data". In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE. 2013, pp. 2228–2233.

[Hah+03] Dirk Hahnel, Rudolph Triebel, Wolfram Burgard, and Sebastian Thrun. "Map building with mobile robots in dynamic environments". In: *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. Vol. 2. IEEE. 2003, pp. 1557–1563.

[Häh+03] Dirk Hähnel, Wolfram Burgard, and Sebastian Thrun. "Learning compact 3D models of indoor and outdoor environments with a mobile robot". In: *Robotics and Autonomous Systems* 44.1 (2003), pp. 15–27.

[Har+68] P.E. Hart, N.J. Nilsson, and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4 (1968).

[Her+14] Evan Herbst, Peter Henry, and Dieter Fox. "Toward online 3-d object segmentation and mapping". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 3193–3200.

[Hol+08] Dirk Holz, Christopher Lorken, and Hartmut Surmann. "Continuous 3D sensing for navigation and SLAM in cluttered and dynamic environments". In: *Information Fusion, 2008 11th International Conference on*. IEEE. 2008, pp. 1–7.

[Hop+92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. "Surface Reconstruction from Unorganized Points". In: *SIGGRAPH Comp. Graph.* (1992).

[Hsu+97] D. Hsu, Jean-Claude Latombe, and R. Motwani. "Path planning in expansive configuration spaces". In: *Proceedings of International Conference on Robotics and Automation* 3 (1997).

[Ich+17] Brian Ichter, Edward Schmerling, Ali-akbar Agha-mohammadi, and Marco Pavone. "Real-time stochastic kinodynamic motion planning via multiobjective search on GPUs". In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 5019–5026.

[Jos+05] David Joslin, Jeremy Frank, Ari K Jónsson, and David E Smith. "Simulation-based planning for planetary rover experiments". In: *Simulation Conference, 2005 Proceedings of the Winter*. IEEE. 2005, 10–pp.

[Kae+12] R. Kaestner, J. Maye, Y. Pilat, and R. Siegwart. "Generative object detection and tracking in 3D range data". In: *ICRA*. May 2012, pp. 3075–3081.

[Kal+07] E. Kalogerakis, P. D. Simari, D. Nowrouzezahrai, and K. Singh. "Robust statistical estimation of curvature on discretized surfaces". In: *Symp. on Geom. Process.* 2007.

[Kav+05] Lydia E. Kavraki and Andrew M. Ladd. “Fast Tree-Based Exploration of State Space for Robots with Dynamics”. In: *Algorithmic Foundations of Robotics VI, Springer* (2005), pp. 97–312.

[Kav+12] Lydia E. Kavraki and Ioan A. Şucan. “A sampling-based tree planner for systems with complex dynamics”. In: *IEEE Transactions on Robotics* 28.1 (2012), pp. 116–131.

[Kav+96] Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[Kla+12] J. Klaess, J. Stueckler, and S. Behnke. “Efficient Mobile Robot Navigation using 3D Surfel Grid Maps”. In: *German Conf. on Robotics*. May 2012.

[Koe+02] Sven Koenig and Maxim Likhachev. “Improved fast replanning for robot navigation in unknown terrain”. In: *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. Vol. 1. IEEE. 2002, pp. 968–975.

[Kon+08] Kurt Konolige, Motilal Agrawal, Robert C. Bolles, Cregg Cowan, Martin Fischler, and Brian Paul Gerkey. “Outdoor Mapping and Navigation Using Stereo Vision”. In: *Experimental Robotics* 39 (2008), pp. 179–190.

[Kum+13] R. Kummerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. “A navigation system for robots operating in crowded urban environments”. In: *ICRA*. 2013, pp. 3225–3232.

[Kut01] Wilhelm Kutta. “Beitrag zur näherungweisen Integration totaler Differentialgleichungen”. In: (1901).

[Kwe+92] In-So Kweon and Takeo Kanade. “High-resolution terrain map from multiple sensor data”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 278–292.

[Lal+06] J. Lalonde, N. Vandapel, D. Huber, and M. Hebert. “Natural terrain classification using three-dimensional ladar data for ground robot mobility”. In: *JFR* 23 (2006).

[Lan+94] Dirk Langer, JK Rosenblatt, and Martial Hebert. “A behavior-based system for off-road navigation”. In: *IEEE Transactions on Robotics and Automation* 10.6 (1994), pp. 776–783.

[Lat91] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, July 1991.

[LaV+00] Steven M. LaValle and James J. Kuffner. “Rapidly-exploring random trees: Progress and prospects”. In: *4th Workshop on Algorithmic and Computational Robotics: New Directions*. Citeseer, 2000, pp. 293–308.

[LaV+99] Steven M. LaValle and James J. Kuffner. “Randomized kinodynamic planning”. In: *Proceedings 1999 IEEE International Conference on*

Robotics and Automation (Cat. No.99CH36288C) 1.5 (1999), pp. 378–400.

[LaV06] Steven M. LaValle. *Planning Algorithms*. 2006.

[LaV11a] Steven M. LaValle. *Motion planning: The Essentials*. 2011.

[LaV11b] Steven M. LaValle. *Motion Planning: Wild Frontiers*. 2011.

[LaV98] Steven M. LaValle. "Rapidly-Exploring Random Trees: A New Tool for Path Planning". In: (1998).

[Lin+05] Stephen R Lindemann and Steven M. LaValle. "Current Issues in Sampling-Based Motion Planning". In: *Robotics Research, The Eleventh International Symposium*. 2005, pp. 36–54.

[Loz83] Tomas Lozano-Perez. "Spatial planning: A configuration space approach". In: *Computers, IEEE Transactions on* 100.2 (1983), pp. 108–120.

[Men+14] Matteo Menna, Mario Gianni, Federico Ferri, and Fiora Pirri. "Real-time Autonomous 3D Navigation for Tracked Vehicles in Rescue Environments". In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. 2014, pp. 696–702.

[Mol+12] Mark Moll, Lydia E. Kavraki, and Ioan A. Şucan. "The Open Motion Planning Library". English. In: *IEEE Robotics and Automation Magazine* 19.4 (Dec. 2012), pp. 72–82.

[Mon+06] Michael Montemerlo, Sebastian Thrun, Hendrik Dahlkamp, and David Stavens. "Winning the DARPA Grand Challenge with an AI robot". In: *AAAI*. 2006, pp. 17–20.

[Mor+09] Benoit Morisset, Radu Bogdan Rusu, Aravind Sundaresan, Kris Hauser, Motilal Agrawal, Jean-Claude Latombe, and Michael Beetz. "Leaving Flatland: Toward real-time 3D navigation". In: *ICRA*. 2009, pp. 3786–3793.

[Mor+11] H. Morioka, S. Yi, and O. Hasegawa. "Vision-based mobile robot's SLAM and navigation in crowded environments". In: *IROS*. 2011, pp. 3998–4005.

[Nik] Robert B. Fisher Nikolaos A. Massios. "A Best Next View Selection Algorithm incorporating a Quality Criterion". In: ().

[Nuc+03] A. Nuchter, H. Surmann, and J. Hertzberg. "Automatic model refinement for 3D reconstruction with mobile robots". In: *3-D Digital Imaging and Modeling*. 2003.

[Nüc+08] Andreas Nüchter and Joachim Hertzberg. "Towards semantic maps for mobile robots". In: *Robotics and Autonomous Systems* 56.11 (2008), pp. 915–926.

[Oli+91] Karen E Olin and David Y Tseng. "Autonomous cross-country navigation: an integrated perception and planning system". In: *IEEE expert* 6.4 (1991), pp. 16–30.

[ORo+79] Joseph O'Rourke and Norman Badler. "Decomposition of three-dimensional objects into spheres." In: *IEEE transactions on pattern analysis and machine intelligence* 1.3 (1979), pp. 295–305.

[Pau+02] Mark Pauly, Markus H. Gross, and Leif Kobbelt. "Efficient Simplification of Point-Sampled Surfaces". In: *Visualization*. 2002.

[Pau+03] M. Pauly, R. Keiser, L. P. Kobbelt, and M. Gross. "Shape Modeling with Point-sampled Geometry". In: *ACM Trans. Graph.* 3 (2003).

[Phi+04] Jeff M Phillips, Nazareth Bedrossian, and Lydia E Kavraki. "Guided expansive spaces trees: A search strategy for motion-and cost-constrained state spaces". In: *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*. Vol. 4. IEEE. 2004, pp. 3968–3973.

[Pla+10] Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. "Motion planning with dynamics by a synergistic combination of layers of planning". In: *Robotics, IEEE Transactions on* 26.3 (2010), pp. 469–482.

[Pom+13] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat. "Comparing ICP variants on real-world data sets". In: *Auton. Rob.* 34 (2013), pp. 133–148.

[Pom+14] F. Pomerleau, P. Krusi, F. Colas, P. Furgale, and R. Siegwart. "Long-term 3D map maintenance in dynamic environments". In: *ICRA*. May 2014, pp. 3712–3719.

[Pot+13] Christian Potthast and Gaurav S. Sukhatme. "A probabilistic framework for next best view estimation in a cluttered environment". In: *Journal of Visual Communication and Image Representation* 25.1 (Jan. 2013), pp. 148–164.

[Rei79] John H. Reif. "Complexity of the mover's problem and generalizations". In: *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)* (1979).

[Rim+92] E. Rimon and D.E. Koditschek. "Exact robot navigation using artificial potential functions". In: *Rob. Aut.* 5 (1992).

[Roh+08] Florian Rohrmuller, Matthias Althoff, Dirk Wollherr, and Martin Buss. "Probabilistic mapping of dynamic obstacles using markov chains for replanning in dynamic environments". In: *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE. 2008, pp. 2504–2510.

[Rui+17] JR Ruiz-Sarmiento, Cipriano Galindo, and J Gonzalez-Jimenez. "Robot@ home, a robotic dataset for semantic mapping of home environments". In: *The International Journal of Robotics Research* 36.2 (2017), pp. 131–141.

[Run95] Carl Runge. "Über die numerische Auflösung von Differentialgleichungen". In: *Mathematische Annalen* 46.2 (1895), pp. 167–178.

[Rus+08] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. "Towards 3D point cloud based object maps for household environments". In: *Robotics and Autonomous Systems* 56.11 (2008), pp. 927–941.

[Ryd+11] J. Ryde and N. Hillier. "Alignment and 3D scene change detection for segmentation in autonomous earth moving". In: *ICRA*. May 2011, pp. 1484–1490.

[Sch+11] Falko Schindler and Wolfgang Förstner. "Fast Marching for Robust Surface Segmentation". In: *PIA*. 2011, pp. 147–158.

[Sch+12] K. Schueler, T. Weiherer, E. Bouzouraa, and U. Hofmann. "360 Degree multi sensor fusion for static and dynamic obstacles". In: *IVS*. 2012, pp. 692–697.

[Sch+83a] Jacob T Schwartz and Micha Sharir. "On the "piano movers"' problem I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers". In: *Communications on pure and applied mathematics* 36.3 (1983), pp. 345–398.

[Sch+83b] Jacob T Schwartz and Micha Sharir. *On the "piano movers" problem. II. General techniques for computing topological properties of real algebraic manifolds*. 1983.

[Sch+83c] Jacob T Schwartz and Micha Sharir. "On the piano movers' problem: III. Coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers". In: *The International Journal of Robotics Research* 2.3 (1983), pp. 46–75.

[Sch+88] Jacob T Schwartz and Micha Sharir. "A survey of motion planning and related geometric algorithms". In: *Artificial Intelligence* 37.1-3 (1988), pp. 157–169.

[Ser+05] Thomas Serre, Lior Wolf, and Tomaso Poggio. "Object recognition with features inspired by visual cortex". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 2. Ieee. 2005, pp. 994–1000.

[Sha+84] Micha Sharir and Elka Ariel-Sheffi. "On the Piano Movers' problem: IV. Various decomposable two-dimensional motion-planning problems". In: *Communications on Pure and Applied Mathematics* 37.4 (1984), pp. 479–493.

[Sim+93] T. Siméon and B. Dacre-Wright. "A practical motion planner for all-terrain mobile robots." In: *IROS*. 1993.

[Sin+00] Sanjiv Singh, Reid Simmons, Trey Smith, Anthony Stentz, Vandi Verma, Alex Yahja, and Kurt Schwehr. "Recent progress in local and global traversability for planetary rovers". In: *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. Vol. 2. IEEE. 2000, pp. 1194–1200.

[Smia] Russell Smith. *Open Dynamics Engine*.

[Smib] Russell Smith. *Open Dynamics Engine Manual*. URL: `https://www.ode-wiki.org/wiki/index.php?title=Manual:_All&printable=yes` (visited on 10/10/2017).

[Smi+93] Adrian FM Smith and Gareth O Roberts. "Bayesian computation via the Gibbs sampler and related Markov chain Monte Carlo methods". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1993), pp. 3–23.

[Ste95] A. Stentz. "The Focussed D* Algorithm for Real-time Replanning". In: *Proc. Int. Conf. Art. Intel.* 1995.

[Sto+98] Nilo Stolte and Arie Kaufman. "Robust hierarchical voxel models for representation and interactive visualization of implicit surfaces in spherical coordinates". In: *Implicit Surfaces' 98*. 1998, pp. 81–88.

[Str+08] M. Strand and R. Dillmann. "Using an attributed 2D-grid for next-best-view planning on 3D environment data for an autonomous robot". In: *2008 International Conference on Information and Automation* (2008).

[Şuc+08] Ioan A. Şucan, Jonathan F. Kruse, Mark Yim, and Lydia E. Kavraki. "Kinodynamic motion planning with hardware demonstrations". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. 2008, pp. 1661–1666.

[Şuc+09] Ioan A. Şucan and Lydia E. Kavraki. "Kinodynamic motion planning by interior-exterior cell exploration". In: *Algorithmic Foundation of Robotics VIII*. Springer, 2009, pp. 449–464.

[Sur+03] Hartmut Surmann, Andreas Nüchter, and Joachim Hertzberg. "An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments". In: *Robotics and Autonomous Systems* 45.3-4 (2003), pp. 181–198.

[Tal+02] A Talukder, R Manduchi, R Castano, K Owens, L Matthies, A Castano, and R Hogg. "Autonomous terrain characterisation and modelling for dynamic control of unmanned vehicles". In: *IEEERSJ International Conference on Intelligent Robots and System* 1 (2002), pp. 708–713.

[Ten+10] Moritz Tenorth, Lars Kunze, Dominik Jain, and Michael Beetz. "Knowrob-map-knowledge-linked semantic object maps". In: *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*. IEEE. 2010, pp. 430–435.

[Udu77] SM Udupa. "Collision detection and avoidance in computer controlled manipulators". In: (1977), pp. 737–748.

[Vas+10] Narunas Vaskevicius, Kaustubh Pathak, Razvan Pascanu, and Andreas Birk. "Extraction of quadrics from noisy point-clouds using a sensor noise model". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 3466–3471.

[Ver67] Loup Verlet. “Computer" experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules”. In: *Physical review* 159.1 (1967), p. 98.

[Vin+08] Regis Vincent, Dieter Fox, Jonathan Ko, Kurt Konolige, Benson Limketkai, Benoit Morisset, Charles Ortiz, Dirk Schulz, and Benjamin Stewart. “Distributed multirobot exploration, mapping, and task allocation”. In: *Annals of Mathematics and Artificial Intelligence* 52.2-4 (2008), pp. 229–255.

[Wan+03] Chieh-Chih Wang, Charles Thorpe, and Sebastian Thrun. “Online simultaneous localization and mapping with detection and tracking of moving objects: Theory and results from a ground vehicle in crowded urban areas”. In: *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. Vol. 1. IEEE. 2003, pp. 842–849.

[Wei06] Daniel Weiskopf. *GPU-Based Interactive Visualization Techniques (Mathematics and Visualization)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[Wen+06] S. Wenhardt, B. Deutsch, J. Hornegger, H. Niemann, and J. Denzler. “An Information Theoretic Approach for Next Best View Planning in 3-D Reconstruction”. In: *18th International Conference on Pattern Recognition (ICPR'06)* 1 (2006).

[Woo+05] Sven Woop, Jörg Schmittler, and Philipp Slusallek. “RPU: A Programmable Ray Processing Unit for Realtime Ray Tracing”. In: *SIGGRAPH*. Los Angeles, California: ACM, 2005, pp. 434–444.

[Yam97] Brian Yamauchi. “A frontier-based approach for autonomous exploration”. In: *... Intelligence in Robotics and Automation, 1997. ...* (1997), pp. 146–151.