Robot-Dependent Maps for Coverage and Perception Task Planning

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Tiago Pereira

M.Sc., Electrical and Computer Engineering,
Faculdade de Engenharia da Universidade do Porto, Portugal

Carnegie Mellon University
Pittsburgh, PA

May, 2019

Acknowledgments

This thesis is the result of people cooperating and working together, even across the Atlantic Ocean. First of all, I would like to thank my advisors, António Moreira and Manuela Veloso, for the constant support and guidance over the past years. The time we spent together was much more than working on research; it was fun, and it was a great learning experience on every level. Without their last push, I would not have completed this thesis.

I also want to thank the members of the thesis committee for their flexibility and valuable feedback, namely Ana Aguiar and Pedro Costa from FEUP, and Maxim Likhachev and Ragunathan Rajkumar from CMU. I am grateful towards the CMU-Portugal program, whose staff was always ready to help, and for the financial support provided by the Fundação para a Ciência e a Tecnologia (Portuguese Foundation of Science and Technology) under grant SFRH/BD/52158/2013.

My doctoral journey started in Portugal, where I had my first contact with robotics, and I was happy to meet my colleagues Héber Sobreira, Miguel Pinto, and Filipe Santos. Our enticing discussions molded the person I am today. I am also grateful to all the people in the CORAL group that let me feel at home when I went to Pittsburgh, and with whom I learned so much. I would like to acknowledge the people that spent the most time with me: Steven Klee, Guglielmo Gemignani, Richard Wang, Vittorio Perera, Philip Cooksey, Juan Pablo Mendoza, and Kim Baraka.

But a fulfilled Ph.D. is not one without fun. I want to recognize the friendship of the CMU-Portugal colleagues that accompanied me in this journey between two countries and became my housemates in Pittsburgh, Damião Rodrigues and Luis Pinto. I would also like to acknowledge all the fun and great times I had with my housemates Evgeny Toropov and Paola Buitrago. And I will never forget the weekly "meetings" with my fellow Portuguese friends in Pittsburgh, Luís Oliveira, Rui Silva, and Sheiliza Carmali, with whom I spent evenings with good food and great company.

A special thanks to my girlfriend Romina, who I was lucky to meet as I was starting my Ph.D., for always being there for me. She was my constant support and motivation, and I want to thank her for the happiest years of my life. Last but not least, I would like to thank my family. To my parents Arminda and Raul, and my brother João, I am eternally grateful. They are my unwavering support and inspiration, encouraging me into ever greater adventures.

Abstract

As different mobile robots may increasingly be used in everyday tasks, it is crucial for path planning to reason about the robots' physical characteristics. This thesis addresses path planning where robots have to navigate to a destination, either for coverage tasks or perception tasks, which we introduce. For coverage tasks, robots target the destination position, whereas for perception tasks a robot has to reach a point from where a target can be perceived.

For complex planning problems with multiple heterogeneous robots, it is essential for planning to run efficiently. This thesis introduces robot-dependent maps, which are map transformations that quickly retrieve information related to the feasibility of coverage and perception tasks. The map transformation depends on properties such as footprint, sensing range and field of view.

When dealing with perception tasks, this thesis introduces the concept of perception planning, where paths are calculated not only to minimize motion cost but also to maximize perception quality. In order to find optimal paths for the perception of targets, this work uses an informed heuristic search to determine paths considering both motion and perception. Robot-dependent maps are then used to improve the efficiency of perception planning, by providing dominant heuristics that reduce the number of ray casting operations and node expansion. The use of robot-dependent maps has a low cost that amortizes over multiple search instances.

This thesis also provides a novel technique for multi-robot coverage task planning, with the new robot-dependent maps used as a pre-processing step. The robot-dependent maps are used to improve the performance of task allocation when splitting tasks among multiple robots. By using a pre-processing phase, the feasibility of coverage tasks is known for each robot beforehand, and estimates on the cost of each robot executing each task can be quickly computed. This thesis contributes an algorithm to calculate paths for multiple heterogeneous robots that need to perceive multiple regions of interest. Clusters of perception waypoints are determined and heuristically allocated to each robot's path to minimize overall robot motion and maximize the quality of measures on the regions of interest.

Overall, this thesis provides techniques that deal with robot heterogeneity mathematically, modeling the individual characteristics, so that path planning takes into account the perception quality as well. Assuming we have multiple robots with different geometries, ranging from different footprints to different sensors, the planner accounts for those differences when coordinating the robots. This thesis thus explores algorithms for a planner to dynamically adapt and be able to generate path solutions based on the advantages of each robot. We provide theoretic proofs for some of our contributions and evaluate our algorithms on simulations with a variety of 2D obstacle maps and robots with different physical characteristics.

List of Publications

T. Pereira, M. Veloso, and A. Moreira. *Multi-robot planning using robot-dependent reachability maps*. In Robot 2015, Second Iberian Robotics Conference. Lisbon, Portugal, 2015

T. Pereira, M. Veloso, and A. Moreira. *Visibility maps for any-shape robots*. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Deajeon, South Korea, 2016

T. Pereira, A. Moreira, and M. Veloso. *Improving heuristics of optimal perception planning using visibility maps*. In IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC). Bragança, Portugal, 2016

T. Pereira, M. Veloso, and A. Moreira. *PA*: Optimal path planning for perception tasks*. In European Conference on Artificial Intelligence (ECAI). The Hague, The Netherlands, 2016

T. Pereira, A. Moreira, and M. Veloso. *Multi-robot planning for perception of multiple regions of interest*. In Robot 2017: Third Iberian Robotics conference. Seville, Spain, 2017

T. Pereira, A. Moreira, and M. Veloso. *Optimal perception planning with informed heuristics constructed from visibility maps*. In Journal of Intelligent & Robotic Systems. 2018

T. Pereira, N. Luis, A. Moreira, D. Borrajo, M. Veloso, and S. Fernandez. *Heterogeneous multi-agent planning using actuation maps*. In IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC). Torres Vedras, Portugal, 2018

N. Luis, T. Pereira, D. Borrajo, A. Moreira, S. Fernandez, and M. Veloso. *Optimal perception planning with informed heuristics constructed from visibility maps*. Accepted to publication in Journal of Intelligent & Robotic Systems. 2019

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Motion planning is one of the core capabilities that any robot must have in order to execute even the most basic tasks. Motion planning is thus a common research topic in robotics, where it is usually associated with goal positions in a map. This thesis addresses two types of tasks: *coverage tasks* where the robot has to navigate to a particular destination from where it can actuate the target; and *perception tasks*, where the robot has to reach a point from where the target can be perceived.

Figure 1.1a shows an example of a coverage task, where the robot moves to a target destination to clean the floor. For the coverage task, the robot needs to move to a target location it can cover, and the robot can either actuate on that location or even take some measurements, such as moving to a place to measure temperature or humidity. On the other hand, Figure 1.1b shows the CoBot robot sensing some objects, which can be done from a distance. For the perception task, the robot does not need to necessarily reach the target, as long as the robot can sense the target. The only constraint is for the target to lie within the robot's "visibility," i.e., the target needs to be within the robot's sensing range. As another example, one can imagine a robot following a human that goes into a narrow corridor. Even if the robot cannot move into the narrow corridor due to a large footprint, it can still perceive the human if the human is within its sensor range. As a consequence, for perception tasks, the robot can perceive objects or humans, and take measures, even in regions of space to where it cannot navigate.

In this work, a 2D grid map is used to represent the space where the robot moves, and all the obstacles it encounters. This thesis considers mobile robots with heterogeneous geometric properties, such as varying size, shape, sensing range, and field of view. This work does not tackle problems regarding height and other 3D related geometric constraints.

Traditional path planning techniques handle only the problem of motion planning that takes the robot from an initial position to a final destination. When dealing with perception-related problems, too often

(a) Coverage Task

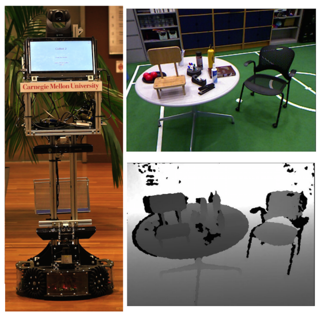

(b) Perception Task

Figure 1.1: a) Example of a coverage task in where the robot moves to positions where it cleans the floor; b) example of a perception task, which shows a robot perceiving a scene and detecting objects from a distance.

the sensing range is considered infinite. Other approaches take into account a maximum sensing range but they overlook the varying uncertainty of measurements depending on distance, considering equally valid perception actions taken from any distance below the range limit. There are also object-oriented planning approaches that tackle mainly the problem of finding the next best view and focus on perception cost and quality but have limited motion considerations.

This thesis examines the path planning problem for a mobile robot both from the motion and perception perspectives, measuring the cost of both and finding solutions that optimize both for motion and perception costs. The thesis also considers the heterogeneity of robots and introduces a map transformation that adapts a grid map to the characteristics of each robot. This map transformation is then used as a pre-processing step for a less time-consuming planning phase.

Although this work has a big focus on mobile robots moving in structured indoor environments, the focus of this thesis is the planning for both coverage and perception tasks and the heterogeneity of agents that execute them. As shown in a later chapter, other mobile vehicles can use the same approach, such as cars moving in a city to deliver something to a person. Instead of CoBot planning on which lab entry to use and navigate to perceive if anyone is inside the lab, one can also apply the perception planning technique to a car moving in a city and deciding where to meet a person for the delivery. In that scenario, the perception range does not represent a sensing operation, but the person walking in a sidewalk (non-navigable by the vehicle) to go to one possible delivery location. Intrinsically, both planning scenarios are similar, and they both consider heterogeneous vehicles with different footprints and different reachability in the world. The algorithm in both cases considers not only motion cost, but also an additional cost associated with either a perception operation or a passenger walking to a delivery location. For the first use case, the planning algorithm finds a solution that optimizes for both motion and perception costs. For the second use case, the delivery scenario, the planning algorithm optimizes for both vehicle and passenger motion.

The framework for motion and perception planning can also be used for multi-robot planning. This thesis shows how the novel map transformation can be used in conjunction with multi-robot classical planning techniques for faster planning in multi-robot problems of coverage tasks. Furthermore, this thesis shows that motion planning of a team of heterogeneous robots that need to perceive targets in a set of predetermined regions can also use our contributed perception planning technique.

The idea of planning for paths that minimize both a motion and perception costs consists in reality of an approach that intrinsically deals with robot heterogeneity. With multiple robots with different physical characteristics, ranging from different footprints to different sensors and perception capabilities, a planner needs to mathematically deal with those differences and produce solutions based on the advantages of each robot. This thesis thus explores and contributes on the techniques necessary for a planner to consider the individual differences of each robot in a team when planning for robot coordination.

1.1 Thesis Question

This thesis addresses the following question:

> *How to efficiently plan for coverage and perception tasks with multiple heterogeneous robots, considering their different geometrical properties, while minimizing motion cost and maximizing perception quality?*

This work argues that both *motion and perception costs* need to be mathematically considered while executing path planning in order to reflect the *heterogeneity* of robots in terms of their motion and sensing capabilities. Measuring the perception quality of measurements and attributing a cost to it provides an immediate way to consider heterogeneity intrinsically in the planning algorithm.

Moreover, while robots are heterogeneous, they commonly use the same map to represent the environment they move in, even if their motion, actuation and perception capabilities in that world are very different. This thesis shows how to adapt maps to the individualities of each robot, resulting in robot-dependent maps. By transforming the map according to the robot's unique *geometrical properties*, it is possible to have a more *efficient path planning*.

Finally, considering the individual capabilities of each robot in the world is essential to create the best coordination and genuinely use the advantage of heterogeneity on a team of *multiple robots*. In particular, in this work there is no coordination that deals with robot variation in a hard-coded way, but instead this thesis aims for a methodology that can dynamically adjust and optimize the multi-robot task allocation to the individual capabilities of each robot.

1.2 Thesis Approach

In order to achieve the goal of having efficient planning for robots executing coverage and perception tasks, the thesis introduces a map transformation that represents the task feasibility regarding the heterogeneity of robots. A brute-force approach can compute the map transformation, but we provide a much faster algorithm. The rationale is that using a structured map transformation allows for easier incremental adaptation to changes in the map, faster computation of feasibility and cost estimates, and provides information that can be used to have a more efficient search in path planning. The maps that are adapted to each robot, the *robot-dependent maps*, are built taking into account its use in more efficient planning techniques.

The thesis approach is to use morphological operations for the map transformation in order to find the overall coverage capabilities of robots in an environment represented by a 2D grid map. From the morphological operations, the map transformation is extended to reason about the perception capabilities of each robot, determining the feasible targets for perception tasks beyond non-traversable regions. The thesis introduces the concept of *critical points*, a set of map positions dependent on the robot footprint, such as those points are the closest to unreachable regions, allowing for fast computation of the visibility map that is very close to the ground-truth. The *critical points* provide the minimum perception distance to some target positions, and the minimum perception distance information can be used to improve the efficiency of path planning for perception tasks.

In terms of path planning, this thesis approach starts from the common A* algorithm but introduces a novel heuristic function and a node expansion methodology that makes the heuristic-based search technique work for the problem of planning for perception tasks. Furthermore, we introduce the concept of perception cost to measure the perception uncertainty and quality, in order to be able to calculate solutions that trade-off the motion and perception costs. This work provides theoretical guarantees of optimality for the cost of paths found, when summing the motion and perception costs, and provides clearly all the underlying assumptions necessary for those guarantees.

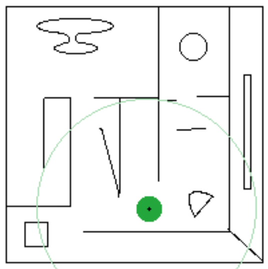
The information from robot-dependent maps is also used to create new heuristics for the problem of perception planning. As an example, the map transformation automatically determines the minimum perception distance to some targets, which can be used to improve the heuristics in informed search planning. Moreover, the robot-dependent maps can also be used to estimate costs for coverage tasks, which coupled with feasibility information can be used in multi-robot coordination to simplify the problem of task assignment.

Finally, the perception planning technique can also be used to generate clusters of waypoints from where multiple target regions can be perceived. With cost estimates for each waypoint calculated from perception planning, it is possible to allocate waypoints to the heterogeneous robots heuristically. By doing so, the algorithm dynamically creates plans for multi-robot teams that take into consideration the robot heterogeneity and both motion and perception cost.

1.3 Contributions

The key contributions of this thesis are the following:

Robot-Dependent Map is a novel 2D map we introduced to represent the actuation and visibility capabilities of robots in terms of their footprint and sensor characteristics. We also contribute an algorithm to quickly generate the robot-dependent maps from grid maps, returning an actuation or visibility map representing the accessible regions of the map as a function of the physical constraints of robots, for coverage and perception tasks. The computation of such maps has a low cost that amortizes over multiple search instances when planning task execution. The contributed technique depends on the initial robot position, the models for the sensor and robot shape and the 2D grid map. The initial positions can be grouped in mutually exclusive sets, such as there is one *Robot-Dependent Map* for each set of initial positions. The contributed algorithm determines the robot-dependent map transformation that does not use brute-force, thus being fast but also returning only an approximation of the true visibility transformation.


(a) Grid Map and Robot

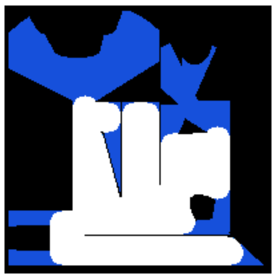

(b) Robot-Dependent Map

Figure 1.2: a) A grid map with a circular robot represented with a small green circle, and its sensing range represented with a light green circumference; b) robot-dependent map with the occluded and non-reachable regions of the grid map in black, the regions for feasible coverage or perception tasks in white, and the non-traversable regions only feasible for perception tasks in blue.

The technique to generate robot-dependent maps is based on the morphological closing operation that is used to first determine the actuation map of each robot, depending on its initial position. The morphological closing operation also gives information about the non-traversable regions of the environment for each robot, where the robot can still perceive targets when executing perception tasks. In order to find the visibility map with all the feasible perception targets, this work contributes the concept of *critical points*, as a set of map positions dependent on the robot footprint, such as those positions are the closest to unreachable regions, allowing for a fast computation of the visibility map that is very close to the ground-truth. Based on the critical points, we introduce a technique to incrementally add regions to the actuation map in a smart way, maintaining overall efficiency while increasing the approximation quality. The described approach with morphological operations is also extended with a multi-layer representation to deal with the any-shape robot and sensor models. Figure 1.2 gives an example of the robot-dependent map.

Informed Search for Perception Planning is a heuristic search for path planning that finds the optimal motion path and perception location for a robot trying to sense a target position in a 2D map while minimizing both motion and perception costs. The overall cost of a path solution is assumed to be the sum of motion cost with perception costs, and a trade-off parameter is used to be able to change the weight of each of these components. Changes in the trade-off parameter shift cost from motion to perception and can generate different solutions, as shown in Figure 1.3.

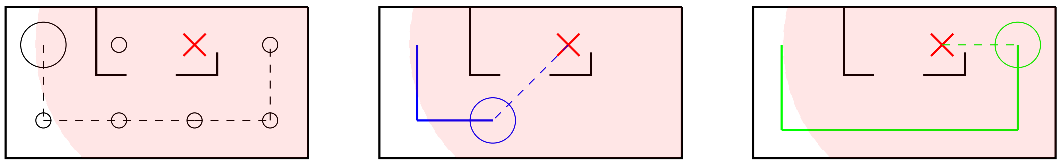

(a) Map, Robot and Target (b) Solution 1 (c) Solution 2

Figure 1.3: a) A simplistic map with a robot starting in the top left grid position, a target represented with a red cross; the perception range is shown with the shaded red region around the target; depending on the trade-off between motion and perception costs, there can be two different path solutions, as shown in b) and c).

The perception cost in this technique accounts for different inaccuracies and uncertainty of different sensors, and models the general decreasing quality of perception measurements with increasing distance. Assuming accurate sensor models, we guarantee the algorithm returns optimal path solutions.

Robot-Dependent Heuristics in Perception Planning is a technique that uses information extracted from robot-dependent maps to improve the efficiency of node expansion in heuristic search for perception planning. When computing the robot-dependent maps, for some targets it is possible to calculate information on the minimum motion and perception distance, which can be used to build dominant heuristics. Dominant heuristics result in fewer node expansions and ray casting operations, and as a result, they can speed up the search process of finding a path for a robot to move to a position that perceives a target location.

Pre-Processing Phase in Multi-Robot Coverage Task Planning introduces the notion of using robot-dependent maps as a pre-processing technique in an automated planning problem, which uses task assignment to distribute covering tasks among multiple robots. In order to do task allocation, a heuristic is used to estimate the cost of each robot executing a task based on the robot-dependent maps. With such pre-processing, the automated planner saves time by only computing the heuristic for feasible tasks. Moreover, the pre-processing also provides a faster cost estimation procedure based on robot-dependent maps, which further improves the efficiency of goal assignment in the multi-agent planning problem.

Multi-Region and Multi-Robot Perception Planning is an algorithm for solving the problem of path planning for multiple heterogeneous robots that need to perceive multiple target regions while minimizing both motion and perception costs. While using the contributed technique for perception planning, this work generates waypoints, i.e., clusters of locations for perception of targets. These waypoints are then used to allocate intermediate goal positions to the paths of robots. This technique using the cost estimates from perception planning as heuristics for the waypoint assignment, creating paths for each robot that minimize the overall motion cost and maximize the perception quality.

Meeting Point Calculation for Delivery Services is an application of our perception planning technique to a more realistic use case of deliveries from vehicles to users in a city. When a vehicle has to make a delivery for a user in a city, it usually drives directly to the passenger location, without ever suggesting the user to walk to some other location. With this contribution, we used the perception planning technique to calculate different rendezvous locations, from which the user can choose according to their preferences. We still consider vehicle heterogeneity with different reachability in the world, and again the algorithm optimizes not only for vehicle motion cost but also for the cost of the user walking to the calculated delivery location.

1.4 Reading Guide to the Thesis

The following outline summarizes each chapter of this thesis.

Chapter 2 - Robot-Dependent Maps introduces a map pre-processing algorithm that transforms generic 2D grid maps on adapted maps to each robot, considering geometric differences of robots, such as size, shape, sensor range and field of view. Those robot-dependent maps can determine the reachability of each robot in the world in terms of coverage and perception. The map transformation also extracts the necessary information from each robot-dependent map that can be used for a more efficient path planning.

Chapter 3 - From A* to Perception Planning introduces an approach for path planning that considers both the motion cost and perception cost, which can represent varying sensing quality with measurements from different distances. The contributed path planning technique is based on heuristic search, and this chapter introduces the heuristic that accounts for perception and guarantees optimal solutions while presenting all the necessary assumptions to support those guarantees.

Chapter 4 - Improving Heuristics of Perception Planning with Robot-Dependent Maps extends the novel perception planning technique with information from Robot-Dependent Maps. For some target locations that need to be perceived, the Robot-Dependent Map can determine additional information on the minimum perception distance and the minimum motion to perceive that target, thus providing valuable information to come up with better heuristic functions closer to the real cost of perceiving a target position.

Chapter 5 - Using Perception Planning as Meeting Point Calculation for Delivery Services introduces an alternative use case for the contributed perception planning technique. When a vehicle has to make a delivery for a user in a city, it usually drives directly to the passenger location, without ever suggesting the passenger to walk to some other location. In this chapter, perception planning is used to calculate different rendezvous locations, from which the user can choose according to their preferences.

Chapter 6 - Heterogeneous Multi-agent Planning Using Actuation Maps introduces a novel combination of robot-dependent maps and classical planning in the context of multi-robot motion planning for coverage tasks. The robot-dependent maps are used as a pre-processing step to speed up the goal assignment phase of a multi-agent planner, and the results are compared with other state-of-the-art planners.

Chapter 7 - Multi-Robot Planning for Perception of Multiple Regions of Interest introduces an approach to multi-robot path planning when multiple regions of interest need to be perceived by any of the robots. Instead of having a single target location that needs to be perceived, this approach uses regions composed of multiple perception targets and calculates waypoints from where various targets can be perceived. A heuristic-based waypoint allocation method that distributes waypoints among the heterogeneous robots is contributed, such that both motion and perception cost is minimized for this team of robots with different capabilities.

Chapter 8 - Related Work reviews the relevant literature. This chapter also presents other motion planning techniques, focusing on heuristic search for path planning. Relevant work on map representation is reviewed, as well as problems related to visibility and perception planning. We also present related literature to multi-robot motion planning.

Chapter 9 - Conclusions and Future Work concludes the thesis with a summary of its contributions, and with a presentation of possible directions for future research.

Chapter 2

Robot-Dependent Maps

Robots have quite often different capabilities, but if they move in the same environment, they all usually share the same map of the environment. Using the same map can bring inefficiencies.

Given a task for a robot to move to an (x, y) position in its map, why should there be any time spent on planning a path to get to that position, if it lies in a region that is not reachable to that specific robot? This question is about a coverage task, but the same applies to perception tasks and other robotic problems. Why should a robot spend any time planning a solution to perceive a target in a room that cannot be perceived by the robot from any location? Furthermore, we can also think about the localization problem. For localization, there is usually a scan matching procedure that finds a position estimate by fitting a laser range finder scan to the map of the environment. Having an unadapted map of the environment means that it is possible for the scan to match against walls and other points in the map that are neither reachable nor visible to a particular robot, introducing inefficiencies in the scan matching process.

In order to achieve our goal of having an adapted map to the geometrical properties of each robot, we created a map transformation, the **robot-dependent map**, that represents the task feasibility regarding the heterogeneity of robots. A brute-force approach can always calculate the map transformation to compute the coverage or perception task feasibility, but that is not necessarily the best option in terms of the information we can extract to improve online planning execution. For example, knowing all the exact location from where a target can be perceived can be opposite to our goal, and burden the computation complexity for path planning.

In this chapter, we describe our contribution of a map transformation that allows for

- Easier incremental adaptation to changes in the map
- Faster computation of task feasibility and cost estimates

- Extraction of information that can be used for a more efficient search in planning.

Our approach is to use morphological operations for the map transformation in order to find the overall actuation capabilities of robots in an environment represented by a 2D grid map. From the morphological operations, we extend the map transformation to reason about the perception capabilities of each robot, determining the feasible targets for perception tasks beyond non-traversable regions. We also introduce the concept of *critical points*, from which we can have a simple sampling of the reachable space with very few points and still build an approximate map of the perception feasibility that is very close to the ground-truth. The *critical points* also provide minimum perception distance to target positions, a piece of information from this map transformation that allows for more efficient planning.

The robot-dependent maps represent the accessible regions of the map in terms of actuation and perception as a function of the initial robot position and its physical constraints, such as robot footprint and sensor field of view and maximum range. The computation of such maps has a low cost that amortizes over multiple search instances when executing path planning.

In our previous work, we introduced the algorithm to build robot-dependent maps [34, 32], which we will explain in detail in this chapter.

In the next section, we review morphological operations, the technical base for the work in this chapter.

2.1 Morphological Operations

Morphological operations are a common technique used in image processing. Here we focus on binary morphology applied to black and white images, which can be obtained by thresholding a normal image, as shown in Figure 2.1. One typical application is noise reduction [16, 39], which can be accomplished with morphological closing, a dilation operation followed by an erosion. In a black and white image where black is the background and white the foreground, dilation will expand one of the color regions (e.g., white points), and the erosion will shrink it. By applying these two operations sequentially, small black clusters inside white regions disappear, successfully eliminating noise from the foreground. The noise reduction works when the noise blobs are smaller than the inflation and deflation radius. On the other hand, the morphological opening can be used to eliminate noise from the background, shown in Figure 2.1.

The basis of mathematical morphology is the description of image regions as sets, where each pixel can be an element of the set. By definition, when we refer to an image set A, it represents the set of all pixels with one color in the corresponding image. Assuming A represents the white pixels, then its complement, A^c, represents the set of black pixels.

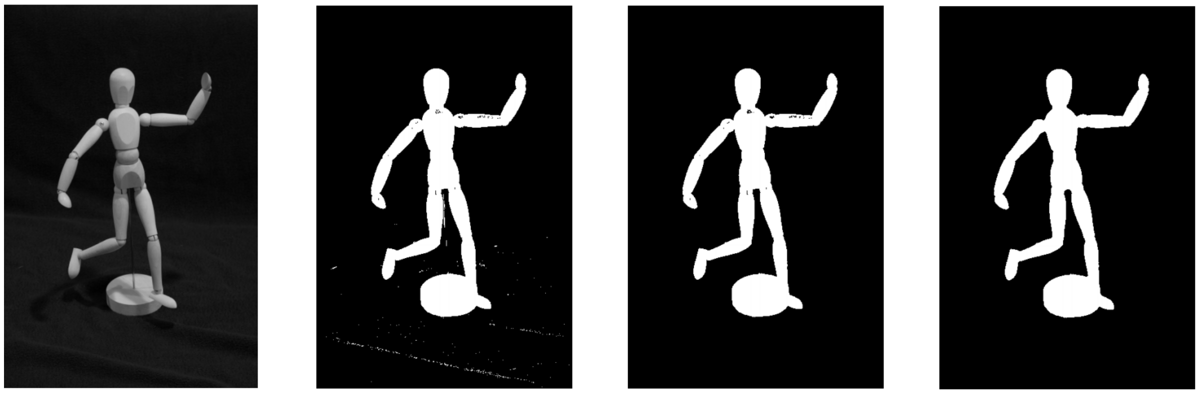

(a) Original Image (b) After Thresholding (c) After Opening (d) After Closing

Figure 2.1: Examples of morphological operations, where opening eliminates white noise from the black background, and closing eliminates black noise from the white foreground; the noise eliminated is smaller than the structuring element used in the morphological operations.

The translated set A_t corresponds to a translation of all the white pixels by t.

$$A_t = \{p \mid \exists a \in A : p = a + t\} \tag{2.1}$$

The reflection of a set A, $\check{A}$, is defined as

$$\check{A} = \{p \mid -p \in A\}. \tag{2.2}$$

The two basic morphological operations are dilation and erosion. Dilation $\oplus$ (also called Minkowski addition [46]) is defined as follows:

$$A \oplus B = \{c \mid \exists a \in A, b \in B : c = a + b\}. \tag{2.3}$$

Alternatively, we can think of dilation as taking multiple copies of A and translating them according to the pixels in B (an origin for B has to be defined, with the center of image B being usually used as origin).

$$A \oplus B = \bigcup_{b \in B} A_b \tag{2.4}$$

Even another interpretation can be taken, by thinking of copies of B at each pixel of A, which is equivalent to using the commutative property on equation 2.4, which is somehow similar to the convolution operation, because the structuring element B is sliding to each position of A and the union for all the positions is taken.

Erosion $\ominus$ (also called Minkowski difference [46]) is defined as

$$A \ominus B = \{c \mid \forall b \in B : (c + b) \in A\}, \tag{2.5}$$

which corresponds to taking copies of A translating them with movement vectors defined by each of the pixels in B. The translation is in the opposite direction, and all copies are intersected.

$$A \ominus B = \bigcap_{b \in B} A_{-b} \tag{2.6}$$

Thus, erosion is equivalent to moving a copy of B to each pixel of A, and only counting the ones where the translated structuring element lies entirely in A. Unlike dilation, erosion is not commutative. Erosion and dilation are dual operations, and their relationship can be written using the complement and reflection operations defined previously:

$$(A \oplus B)^c = A^c \ominus \check{B}, \tag{2.7}$$

$$(A \ominus B)^c = A^c \oplus \check{B}. \tag{2.8}$$

In other words, dilating the foreground is the same as eroding the background with a reflected structuring element.

The opening morphological operation $\circ$ is an erosion followed by a dilation with the same structuring element.

$$A \circ B = (A \ominus B) \oplus B \tag{2.9}$$

The opening morphological operation can be considered to be the union of all translated copies of the structuring element B that can fit inside A. Openings can be used to remove small blobs, protrusions, and connections between larger blobs in images.

The closing morphological operation $\bullet$ works in an opposite fashion from opening, by applying first the dilation and then the erosion.

$$A \bullet B = (A \oplus B) \ominus B \tag{2.10}$$

While opening removes all pixels where the structuring element does not fit inside the image foreground, closing fills all places where the structuring element will not fit in the image background. Closing and opening are also dual operations, but not the inverse.

$$(A \circ B)^c = A^c \bullet \check{B} \tag{2.11}$$

The closing and opening operations are idempotent, as when applying more than once the same operation, nothing changes after the first application: $A \circ B \circ B = A \circ B$ and $A \bullet B \bullet B = A \bullet B$.

Opening and closing are the basic operations in morphological noise removal.

Morphological operations have already been used as a map transformation, by automatically extracting topology from an occupancy grid [12]. The morphological operations can thus robustly find the big spaces in the environment like humans would, separating it into regions.

These image processing techniques have also been used in motion planning algorithms, inflating obstacles to determine the configuration space in order to find a path that minimizes a cost function while avoiding collisions. Indeed, inflation is the solution used, for example, in the ROS navigation package [25].

2.2 Robot-Dependent Maps for Circular Footprints

We first assume robots have a circular shape and a sensor with a maximum sensing range, with 360 degrees of field of view. We assume a full field of view because that is equivalent to considering a robot with a circular footprint and limited field of view. For example, for a robot with a beam sensor with a maximum range, if the robot has either an omnidirectional or differential motion, it can rotate in place and simulate a robot with a full field of view, and the same maximum perception range.

In this section we will show how to build both Robot-Dependent Actuation Maps and Visibility maps to efficiently determine the feasibility of coverage and perception tasks, respectively, assuming robots with a circular footprint.

2.2.1 Robot-Dependent Actuation Map

The goal of robot-dependent actuation maps is to efficiently determine the actuation capabilities of a robot in a particular environment, i.e., determine what regions can be covered from any point that is reachable from the initial robot position.

Robots move along the environment, which we represent as a grid map. Given the robot's geometric properties, there will be some regions that are accessible to the robot and some regions that will be inaccessible. Our algorithm is a function of robot size and its actuation range. We show in Figure 2.2 a simulated environment with obstacles and the resulting actuation map.

(a) Map

(b) Actuation Map

Figure 2.2: a) A simulated map with obstacles in black and a circular robot in green; b) the resulting actuation map, with regions that can be covered by the robot in white and regions that cannot be covered by the robot in black, assuming the actuation range is equal to the robot size.

Considering maps are discrete representations of the environment, there is a duality between images and maps because both of them are a discrete sampling of the world. The first step of our algorithm is to transform an occupancy grid map (with probabilities of occupation in each cell) into a binary map of free and obstacle cells, using a threshold. The occupancy grid can be obtained through SLAM methods.

To determine the robot-dependent map, we use the partial morphological closing operation, which can be applied on images using a structuring element with a shape that represents the robot. The domain is a grid of positions G. The input is a black and white binary image representing the map, in Figure 2.3a. M is the set of obstacle positions, where each pixel corresponds to a grid position. The structuring element, R, represents the robot. The morphological operation dilation on the obstacle set M by R is

$$M \oplus R = \bigcup_{z \in R} M_z, \tag{2.12}$$

where $M_z = \{p \in G \mid p = m + z, m \in M\}$. By applying the dilation operation to the obstacles in the map (black pixels in the image), the algorithm inflates the obstacles by the robot radius, which can be used to find the free configuration space, in Figure 2.3b.

$$C^{free} = \{p \in G \mid p \notin M \oplus R\} \tag{2.13}$$

The configuration space shows where the robot center can be, representing the feasible positions for the robot center, but not giving any information on which regions can be actuated or perceived by the robot. From C^{free}, it is possible to find the points in the free configuration space where the robot can be by moving from the initial robot position S, which we call the set of navigable points $Nav(S)$, in Figure 2.3e.

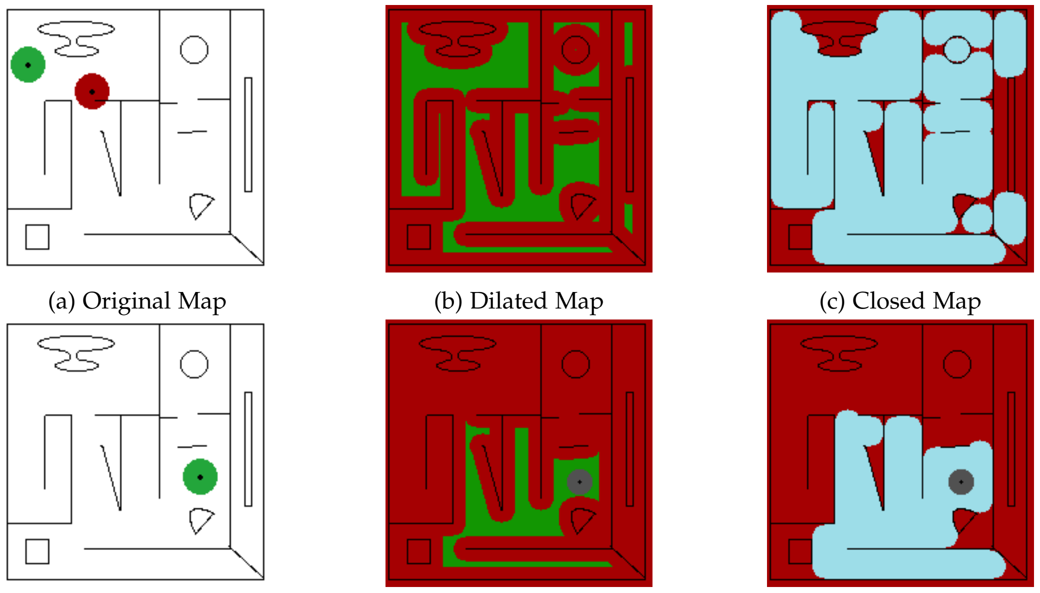
$$Nav(S) = \{p \in C^{free} \mid p \text{ connected to } S\} \tag{2.14}$$

Because dilation and erosion are dual operations, the morphological closing is computed by dilating the free configuration space. The partial morphological closing applies the second morphological operation to a subset of C^{free}, dilating $Nav(S)$ instead.

$$A(S) = Nav(S) \oplus R \tag{2.15}$$

We assume that the actuation radius is the same as the robot size. In case the actuation range is smaller, in the second dilation operation, instead of R we would have a structuring element representing a circle with a smaller radius, equal to the actuation range.

Using the partial morphological operation, we determined the actuation space, $A(S)$, which represents the regions a circular robot can touch with its body, given its radius and an initial position. The corresponding *Actuation Map* is in Figure 2.3f, which represents the regions of the environment that can be


(a) Original Map (b) Dilated Map (c) Closed Map

(d) Initial Robot Position (e) Reachable Space (f) Actuation Space

Figure 2.3: a) Map with two possible positions for the robot, the green one is feasible, while in the red the robot overlaps with obstacles; b) the configuration space is obtained with the morphological dilation (C^{free} is the set of green regions); c) the morphological closing operation; d) the initial robot position; e) the navigable space, with initial position represented by gray circle; f) the partial morphological operation applies the second dilation operation only to the navigable space, resulting in the actuation space.

actuated by a robot. The regions outside the actuation space, $U(S)$, cannot be reached by the robot body, and thus cannot be actuated.

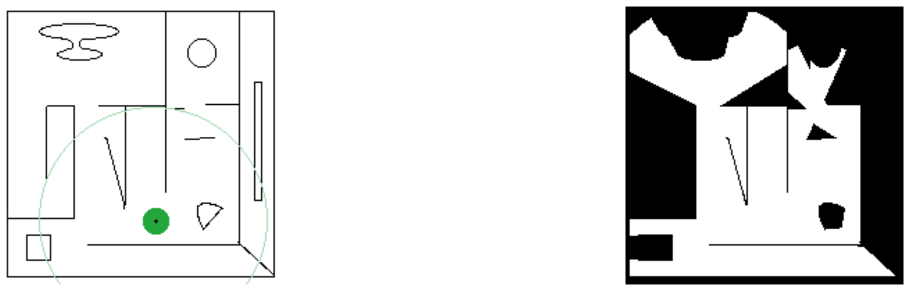
$$U(S) = \{p \in G \mid p \notin A(S) \wedge p \notin M\} \tag{2.16}$$

As an example, we can consider a vacuum cleaning robot. The configuration space represents the possible center positions for the circular robot, and $A(S)$ represents the regions the robot can clean. Finally, the unreachable regions $U(S)$ are the parts of the environment the robot cannot clean. For example, a circular vacuum cleaner can never reach and clean corners.

2.2.2 Robot-Dependent Visibility Map

The goal of robot-dependent visibility maps is to efficiently determine the perception capabilities of a robot in a specific environment, i.e., determine which regions can be perceived by the robot's sensor from any point that is reachable from the initial robot position. We show in Figure 2.4 an example of a resulting visibility map, given an initial robot position, footprint, and sensor maximum range, r_p.

If we consider the problem of perception, a *Visibility Map* represents the regions of the input map that are visible by the robot from some reachable position. One approach to finding the visibility map is to use a brute-force algorithm. Each position in the map can be tested for visibility, by finding at least one



(a) Map (b) Visibility Map

Figure 2.4: a) A simulated map of obstacles in black, a circular robot in green, and the robot sensing range in light green; b) the Visibility Map, with regions that can be sensed by the robot in white and occluded regions in black.

feasible robot position in $Nav(S)$ that has line-of-sight to the target point. For that purpose, we use the ray casting technique. This approach is however very computationally expensive, especially if the robot moves in an environment with unexpected and dynamic obstacles.

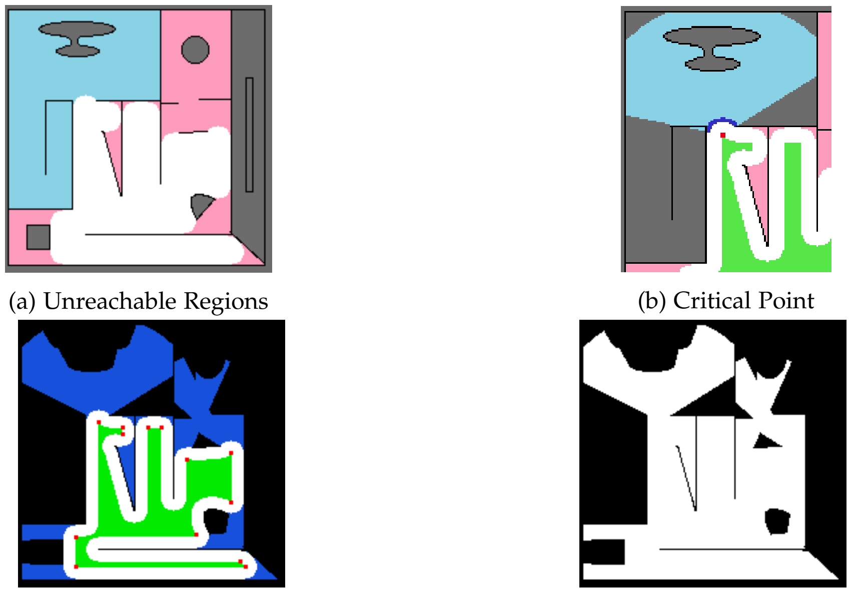
Given the robot's geometric properties, there will be some regions that are accessible to the robot and some regions that will be inaccessible. Furthermore, robots can use their sensors to perceive inside inaccessible regions. Our algorithm is a function of the robot size and sensing range.

As an alternative to using the brute-force approach, we can take the actuation space $A(S)$ and consider it as a first approximation of the visibility map. The visibility map is then built incrementally from $A(S)$. The unreachable regions $U(S)$ are divided into a set of different disconnected components $U^l(S)$, which is useful because it allows determining additional visibility inside each one independently. Each region $U^l(S)$ has its unique openings to the actuation space, from where visibility inside $U^l(S)$ is possible. These openings are the frontiers, defined as the points of the unreachable space that are adjacent to $A(S)$, as shown in Figure 2.5a.

$$F^l(S) = \{p \in U^l(S) \mid \exists p' : p' \text{ is adjacent to } p \wedge p' \in A(S)\} \tag{2.17}$$

The frontier set can be composed of multiple disjoint segments $F^{li}(S)$, and visibility inside the unreachable region should be determined for each segment independently. The additional visibility in each U^l always comes from points with line-of-sight through $F^{li}(S)$. Therefore, the algorithm automatically discards unreachable regions without frontiers.

Multiple candidate positions can sense inside $U^l(S)$, and all of them have to be in $Nav(S)$, the feasible positions for the robot center. In order to have the true visibility map, all points from $Nav(S)$ should be considered. However, the complete solution is computationally expensive, so we propose an alternative, where the visibility inside unreachable regions through each frontier segment is determined only from one point of the navigable space.



(a) Unreachable Regions (b) Critical Point

(c) Visibilities from Critical Points (d) Visibility Map

Figure 2.5: a) A map with $A(S)$ in white, the unreachable regions that connect with $A(S)$ in pink, and an example of a disconnected unreachable region in light blue; b) highlighting a disconnected unreachable region, with the frontier segment points $F^{li}(S)$ in dark blue, the critical point $c^*_{li}(S)$ in red, and the expected visibility $V_e^{cli}(x)$ in light blue; c) $Nav(S)$ in green, all critical points in red and respective extended visibility regions in dark blue; d) the final visibility map.

As the algorithm only uses one point, the output of this algorithm is an approximate visibility map. To obtain a good approximation, the point chosen has to maximize the expected visibility inside the unreachable region. Given a point p, it is possible to determine the expected visibility $V_e^{pli}(S)$ as the area of an annulus sector in U^l defined by the robot sensing radius, and the frontier extremes. A point closer to the frontier is chosen to maximize the expected visibility, as it has a deeper and wider view inside $U^l(S)$.

$$c^*_{li}(S) = \underset{p \in Nav(S)}{\operatorname{argmin}} \sum_{\zeta \in F^{li}(S)} \|p - \zeta\|^2 \tag{2.18}$$

We define the point $c^*_{li}(S)$ as a *critical point*, shown in Figure 2.5b. As explained before, in order to reduce the computation needed to calculate the visibility map, we choose only one critical point per frontier $F^{li}(S)$. For each pair of frontier $F^{li}(S)$ and critical point $c^*_{li}(S)$, the algorithm defines an annulus sector of expected visibility inside the unreachable regions, $V_e^{cli}(S)$, as illustrated in Figure 2.5b. In order to deal with occlusions, we consider the points in $V_e^{cli}(S)$ and determine the true visibility of $V_t^{cli}(S)$ from critical points $c^*_{li}(S)$ using ray casting, considering the maximum sensing range. The algorithm determines the true visibility from the critical point and through the corresponding frontier inside the corresponding unreachable regions. Those points of true visibility define the set $V_t^{cli}(S) \subseteq V_e^{cli}(S)$.

$$V(S) = A(S) \bigcup_{li} V_t^{cli}(S) \tag{2.19}$$

After analyzing unreachable regions $U^l(S)$ independently, we were able to determine the visibility inside each one. The overall visibility for the whole map is then given by the union of the actuation space with the individual visibilities $V_t^{cli}(S)$ obtained for each region $U^l(S)$.

The complexity of determining the position of each critical point depends on the robot size and the size of the frontier. As we know the distance between the critical point and the frontier is the robot radius, we define a rectangular search box such as its boundaries have a distance to the frontier extremes equal to the robot radius. Then the algorithm only looks for the critical point inside that rectangular search area.

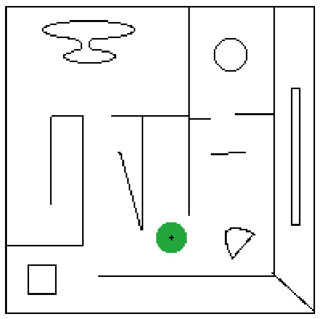
2.2.3 Results

Our solution is an approximation of the real visibility, as the robot-dependent visibility map is not the same as the yielded by the ground-truth visibility, as shown in Figure 2.6. The approximation is the result of considering only visibility from critical points. However, the final visibility is generally very close to the true visibility given by the ground-truth. Figure 2.6 shows false negatives in blue and correct visibility in white.

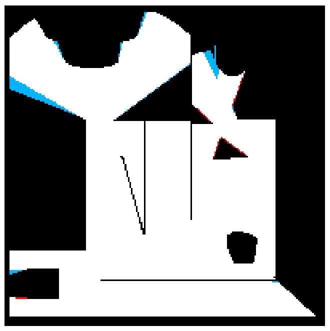
The precision of our algorithm is always 100% because there are no false positives. As an approximation algorithm, there can be some visibility that is not detected. However, all the visibility determined from the critical points is always correct, thus resulting in perfect precision. On the other hand, recall is the amount of visibility not accounted in our approximate algorithm, which is always due to considering one critical point per frontier. The effect of using only one point varies with the topology of the environment, being worse for larger unreachable regions and sensing ranges, shown in Figure 2.6c, as the error propagates and increases with distance from the critical point. While error increases, the efficiency of our algorithm also increases with larger sensing ranges, as seen in Figure 2.6d. Finally, the time efficiency of our contributed algorithm increases linearly with the increase of image size.

2.3 Robot-Dependent Maps for Any-Shape Robots

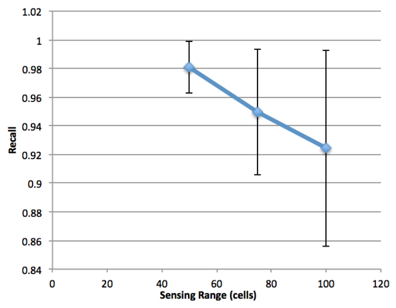
For the case of non-circular robot footprints (Figure 2.7), given that the robot model is not rotation invariant, we need to discretize orientation as well. We introduce a world representation that is composed of multiple layers, using the partial morphological closing operation to each layer, and as such determining individually for each orientation the corresponding actuation space [32].



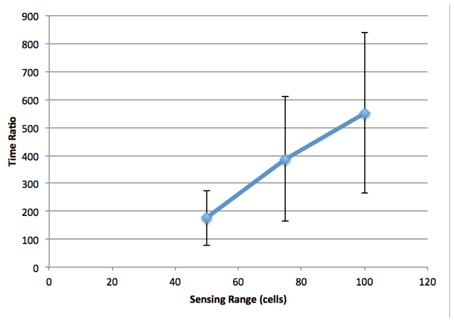
(a) Map and Initial Robot Position



(b) Visibility Map and Ground-Truth



(c) Recall



(d) Time Ratio

Figure 2.6: a) The simulated map; b) comparison of approximate visibility map and ground-truth in a 200 by 200 cells map, robot radius of 9 cells, and sensor radius of 80 cells; visibility true positives in white, true negatives in black, and visibility false negatives in blue; c) changes in recall as a function of maximum sensing range; d) changes in the ratio of ground-truth computation time divided by approximate visibility computation time, as a function of maximum sensing range.

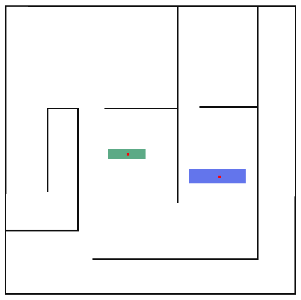
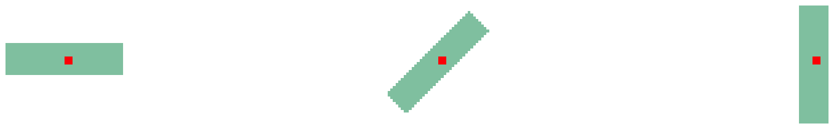


Figure 2.7: Environment and robot models used to test the extended approach to any-shape robots.

To extend the approach described in the previous section to any-shape robots with any sensor model as well, we used a multi-layered representation to discretize orientation and a new method to choose the critical points. We parametrized both the robot shape and sensor model with images that can be rotated and scaled. It is also possible to define the sensor and robot centers, and their relative positions. First, the algorithm needs images to model both the robot and its actuation capabilities. Both are parametrized by images that can be rotated and scaled to represent any robot. As input, it is also necessary to give the center of the robot and actuation in terms of their model images and their relative positions.

Here we assume a quantization in the θ dimension (i.e., orientation), where n_θ is the number of layers in the quantization. After the initial parametrization, the robot and sensor models (structuring elements) are rotated by $2k\pi/n_\theta$, where $0 \leq k < n_\theta$, as shown in Figure 2.8. The rotated $R(k)$ and $Sens(k)$ represent the robot and sensor models for each possible discrete orientation k.



(a) Robot Model R for $\theta = 0^{\text{o}}$ (b) Robot Model R for $\theta = 45^{\text{o}}$ (c) Robot Model R for $\theta = 90^{\text{o}}$

Figure 2.8: Example of an image representing the robot footprint, rotated for three different angles, and used as structuring element in the morphological operations applied to the respective orientation layers; robot center shown with red dot.

The morphological operations can now be used to determine the free configuration space for each layer k, by dilating the map with the corresponding robot shape $R(k)$.

$$C^{free}(k) = \{p \in G \mid p \notin M \oplus R(k)\} \quad \forall 0 \leq k < n_\theta \tag{2.20}$$

We use a circular representation for the layered orientation, where the next layer after $\theta_j = n_\theta - 1$ is layer $\theta = 0$.

In order to model a robot that navigates through the grid map, we need to establish the type of connectivity between points in different layers, such as it is equivalent to the real motion model of the robot. As an example, the connectivity graph from Figure 2.9, where one point is connected to all its neighbors in the same layer and the respective positions in adjacent layers, is equivalent to considering an omnidirectional model of navigation.

Given the connectivity mode, it is then possible to find all points in each layer of the configuration space that connect with the starting robot location S, obtaining the navigable set $Nav(S,k)$. We can then use a second dilation operation to the navigable space in each layer to get the actuation space for each orientation. The structuring element for this second operation is the one that models the actuation capabilities, T, which dilates the space according to the actuation model. If instead the structuring element R is used again, that would be equivalent to assuming an actuating ability utterly coincident with the entire footprint.

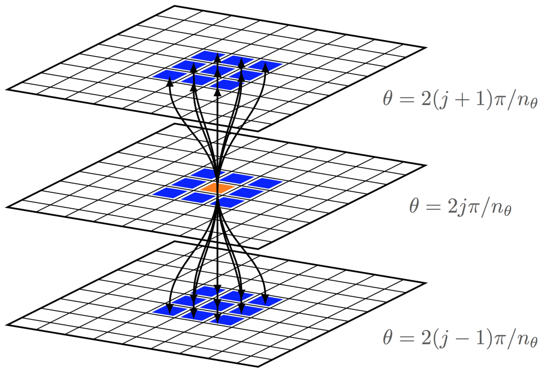

$\theta = 2(j+1)\pi/n_\theta$
$\theta = 2j\pi/n_\theta$
$\theta = 2(j-1)\pi/n_\theta$

Figure 2.9: Three adjacent layers of the discretized orientation, showing in blue the neighbor points of a central orange dot, representing the connectivity of an omnidirectional motion model.

Then, the actuation space for each layer would be given by

$$A(S,k) = Nav(S,k) \oplus R(k). \tag{2.21}$$

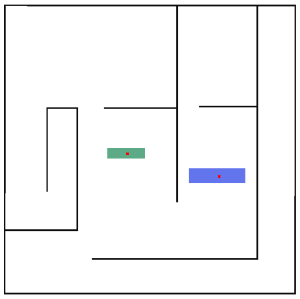
This actuation space represents the points in each layer that can be touch by the robot with the corresponding orientation. After determining the actuation space for each layer, the multiple layers are projected into one single 2D image to compute the overall actuation capabilities for any orientation.

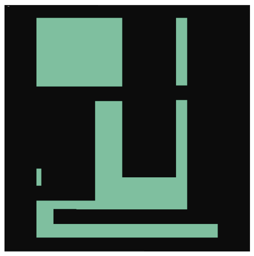
$$PA(S) = \bigcup_k A(S,k) \tag{2.22}$$

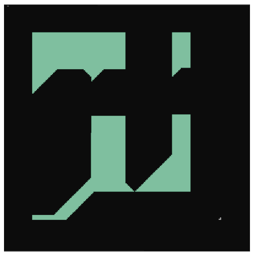
The set $PA(S)$ is equivalent to the feasible points in the *Actuation Map*.

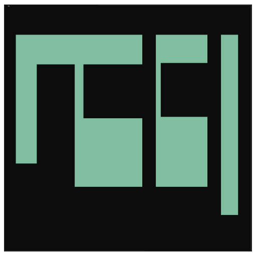
The actuation space gives the actuation capabilities for each orientation for a given robot shape and starting position. So, if a point belongs to $PA(S)$, then it can be actuated by the robot. We show in Figure 2.10 the navigable and actuation spaces for different layers.

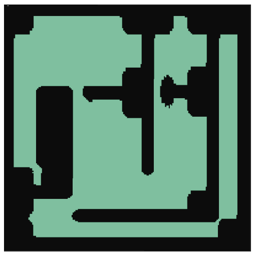
From this figure, it may look that navigability and actuation capabilities are calculated with morphological operations independently based on orientation. If that were the case, projecting the navigability and actuation onto a 2D image would be an incorrect transformation, as paths could become feasible even if adjacent 2D grid positions would come from non-adjacent orientations. However, the only operation that is taken independently in terms of orientation is building the configuration space.

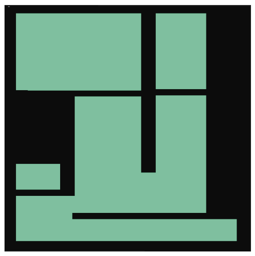

(a) Environment Layout and Initial Positions

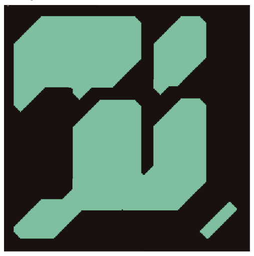

(b) Navigable Space 0º Layer, Robot 1

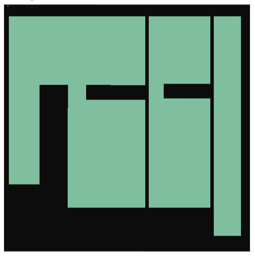

(c) Navigable Space 45º Layer, Robot 1

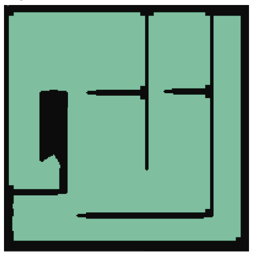

(d) Navigable Space 90º Layer, Robot 1

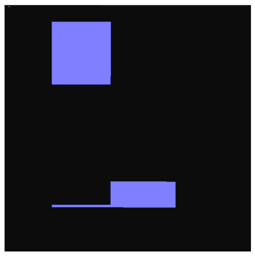

(e) 2D Projected Navigability, Robot 1

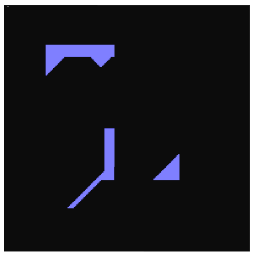

(f) Actuation Space 0º Layer, Robot 1

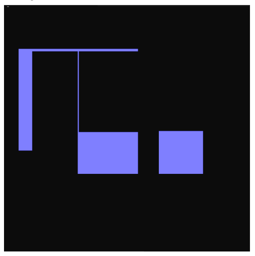

(g) Actuation Space 45º Layer, Robot 1

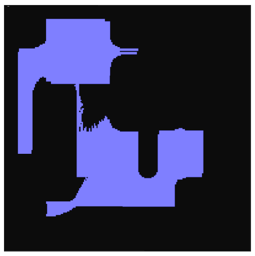

(h) Actuation Space 90º Layer, Robot 1

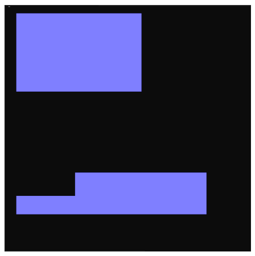

(i) 2D Projected Actuation Map, Robot 1

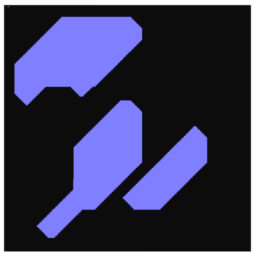

(j) Navigable Space 0º Layer, Robot 2

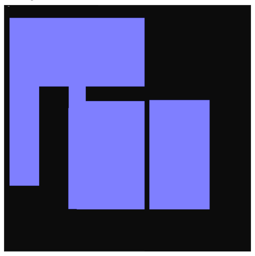

(k) Navigable Space 45º Layer, Robot 2


(l) Navigable Space 90º Layer, Robot 2


(m) 2D Projected Navigability, Robot 2


(n) Actuation Space 0º Layer, Robot 2


(o) Actuation Space 45º Layer, Robot 2


(p) Actuation Space 90º Layer, Robot 2

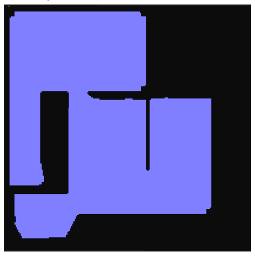

(q) 2D Projected Actuation Map, Robot 2

Figure 2.10: Navigable and actuation space for two non-circular robots with different sizes, for the scenario shown in a).

Afterward, we determine the navigable space from an initial position using the layered structure and considering the robot motion model. Therefore, paths cannot be feasible if only their 2D positions are grid neighbors. To determine navigability, we also consider if two points are neighbors in terms of orientation (being in the same or an adjacent orientation layer). So, after projecting navigability and actuation onto a 2D image, neighbor points will only be connected if there is a path between them that is feasible in terms of rotation and orientation in the layered structure. Thus we guarantee that a corner in the environment will only be navigable by a rectangular robot if the passage is not too narrow for the robot footprint, allowing it to rotate while moving along the corner.

$PA(S)$ has the same kind of representation we had with the circular robot, where the actuation space is a single 2D image not depending on the orientation.

Similarly, the *Visibility Map* can also be computed incrementally from the overall actuation space, as we did in the circular robot case.

However, we need to use a structuring element to represent the sensing capabilities bounded by the robot shape: $Sens_B(k) = R(k) \cap Sens(k)$. Replacing $R(k)$ by $Sens_B(k)$ in equation 2.21 results in $PV_B(S)$, which now can be used as a starting point to determine the Visibility Map for a robot with a sensor model $Sens$. The set $PV_B(S)$ gives the projected visibility bounded by the robot shape, and it has the same kind of representation we had with the circular robot, with a single 2D image that does not depend on orientation.

Therefore, the unreachable regions $U^l(S)$ and respective frontiers $F^l(S)$ can be determined in the same way as we did with the circular robot.

2.3.1 Critical Points for Any-Sensor Model

In the any-shape robot case with general sensor models, the previous definition of critical point is not ideal. If the critical point was chosen again as the closest position to the frontier points, the result could be inferior as a position with a non-optimal orientation towards the frontier might be chosen. Therefore, we do not use the center of the robot as a critical point, but the sensor center instead, because it is the sensor position relative to the frontier that controls the amount of visibility determined inside the unreachable region. The critical point is still the position that maximizes the expected visibility, but now we redefine the calculation of expected visibility V_e^{cli}.

For that purpose, we build a histogram of the sensor model that represents the visibility of the sensor in each direction. If ϕ is the angle of the vector from the sensor center to another point in the sensor model, $b_s(\phi)$ is the correspondent bin in the histogram. Therefore, b_s returns values between 0 and $n_s - 1$, with n_s being the number of bins.

$$b_s(\phi) = \underset{0 \leq n < n_s}{\operatorname{argmin}} |\text{angleNorm}(\phi - 2n\pi/n_s)| \tag{2.23}$$

The function *angleNorm* normalizes the angle between π and $-\pi$. The histogram is built by iterating over all points in the sensor model *Sens*, determining the angle ϕ for each point, and adding the point to the respective histogram bin $b_s(\phi)$. With the histogram, it is possible to estimate the expected visibility inside the unreachable regions using only the frontier extremes.

In order to search for the critical point, we use the angle of an annulus sector defined by the candidate point c_{li} and the frontier extremes, with angles ϕ_1^{cli} and ϕ_2^{cli}. The expected visible area inside the unreachable region is given by the sensor histogram:

$$V_e^{cli} = \Big(\sum_{n=b_s(\phi_1^{cli})}^{b_s(\phi_2^{cli})} \text{hist}(n) \Big) - ||c_{li} - C_f^{li}||^2 (\phi_2^{cli} - \phi_1^{cli})/2, \tag{2.24}$$

where C_f^{li} is the frontier center of mass, and $||c_{li} - C_f^{li}||^2(\phi_2^{cli} - \phi_1^{cli})/2$ accounts for the area from the candidate point to the frontier, already counted in the space $PV_B(S)$.

$$c_{li}^*(S) = \underset{c_{li}}{\operatorname{argmax}} \, V_e^{cli} \tag{2.25}$$

Finally, the critical point is given by searching in the layered image representation. Then, visibility inside U is determined using ray casting, as with the circular shape assumption.

2.4 Summary

In this chapter, we reviewed the morphological operations that are commonly used for noise reduction in image processing. Then, we introduced our contribution of *robot-dependent maps*, which transforms maps of the environment according to the robot's physical characteristics. The goal of this transformation is to be able to adapt grid maps to each robot, representing efficiently in the form of maps the robot-dependent feasibility for coverage and perception tasks. For the visibility map to efficiently represent perception feasibility, we introduced the concept of *critical points* to have a smart sampling of the navigable space in order to quickly obtain a good quality approximation to the visibility map. We showed that the approximation error depends on the environment topology, being greater for larger sensing radius and larger unreachable regions, while the time efficiency compared to the brute-force approach also increases for larger maps. Finally, we presented a multi-layer representation that allows us to extend the robot-dependent maps to any-shape robots, removing the restriction of using only circular sensor models and robot footprints.

Chapter 3

From A* to Perception Planning

Path planning usually deals with the problem of finding a route for a robot to move from an initial position to a destination point. The path can be calculated and optimized according to many different metrics, but those metrics are usually related to the path size. However, if the goal is not moving to a destination point, but to perceive a target location, then sensing the target can usually be performed from many different positions. Thus it becomes necessary to account not only for the motion cost when calculating the optimal path, but also to consider the perception quality.

In this chapter, we assume the perception quality decreases with distance and depends on the specific sensor the robot uses to perceive the target. In this work, we consider perception quality by translating it into a cost function. We introduce a heuristic search for path planning that finds the optimal motion path and perception location for a robot trying to sense a target position in a 2D map while minimizing both motion and perception costs. The overall cost of a path solution is assumed to be the sum of motion cost with perception costs, and a trade-off parameter is used to be able to change the weight of each of these components. As different robots might have different characteristics and different weights for the motion and perception costs, changes in the trade-off parameter can generate different solutions, with each solution being optimal for a specific robot configuration.

The perception cost in this technique is designed to represent the inaccuracies and uncertainty of each sensor, and as such, it measures perception quality. Assuming accurate sensor models, we can guarantee that the algorithm returns optimal path solutions.

In our previous work, we developed the search framework for perception planning that we present in this chapter [35].

3.1 Perception Planning Formulation

In our perception scenario, we have to find a path ρ that minimizes not only the distance traveled but also the perception cost. The path ρ is a sequence of adjacent positions in a grid, $\{s_0, s_1, ..., s_N\}$. The robot starts from the initial position $s_0 = S$ and moves through connected cells of the discretized configuration space to a final robot position, $s_N = F$, such as the sensing target T is perceived from some position in the path ρ. The total cost of path ρ is given by

$$\text{cost}(\rho) = \text{cost}_m(\rho) + \lambda \text{cost}_p(\rho, T), \tag{3.1}$$

where λ is a weight parameter that trades-off the motion cost $\text{cost}_m(\rho)$, and the perception cost $\text{cost}_p(\rho, T)$ in the overall cost function.

Usually, the motion cost cost_m is proportional to the distance traveled from S to F, where $F = s_N$:

$$\text{cost}_m(\rho) = \sum_{i=1}^{N} ||s_i - s_{i-1}||. \tag{3.2}$$

Here we assume that $\text{cost}_p(\rho, T)$ is a function of the minimum distance between the ρ and T, making the perception cost a function of the minimum sensing distance from the path to the target:

$$\text{cost}_p(\rho, T) = c_p\Big(\min_{\substack{s_i \in \rho, \\ s_i \text{ with line-of-sight to } T}} ||s_i - T|| \Big), \tag{3.3}$$

with c_p being a continuous function that depends on the sensor model. In the perception tasks, one of the goals is to increase the accuracy of perception. In our work, we represent the accuracy of perception as a cost function, where a low sensing accuracy corresponds to a high perception cost, and vice-versa. Therefore, the cost of perception is a function of the sensing distance. We assume that the further away the robot is from the target, the lower the sensing accuracy, because of the increased probability of inaccurate measurements. Thus the cost function for perception, c_p, increases with distance.

The trade-off parameter λ could be included in the c_p function, resulting in one less parameter in our model. However, we chose to have it as a separate parameter, in order to have a more flexible model, where λ is only a weight parameter that trades-off motion and perception costs, while the c_p function deals specifically with the modeling of the perception cost function, which depends on the type of sensor used. For example, the perception cost can increase linearly with distance, quadratically, or in many other different ways. The c_p function models the type of perception cost (e.g., linear or quadratic), and we can solve the heuristic for a specific perception cost model, and maintain it as a function of λ. By using this approach, later on, we can reuse the same heuristic, and in case we only want to change the weights of motion and perception costs, we can change only the λ parameter without having to recalculate the heuristic equation for a new type of perception cost function.

The optimal path is given by

$$\begin{aligned}\rho^* &= \operatorname*{argmin}_{\rho\in\mathcal{P}} \text{cost}(\rho)\\ &= \operatorname*{argmin}_{\rho\in\mathcal{P}} \text{cost}_m(\rho) + \lambda \text{cost}_p(\rho, T), \end{aligned} \tag{3.4}$$

where $\mathcal{P}$ is the space of all possible paths.

Theorem 3.1. *For the optimal path ρ^*, the position that minimizes the distance from the path to sensing target T is the final position of the path, F.*

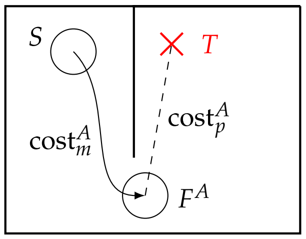
Proof. If there were another position s_i in the middle of the path that had the smallest distance to the target, then there would be a different path ending in s_i with minimal cost, contradicting the hypotheses that the path ρ^* from S to F is the one that minimizes the overall cost. □

The space of all possible paths $\mathcal{P}$ is a general notation for representing the minimization problem to find the optimal path, but our proposed algorithm does not implement any search in the path space. We are going to show in the following sections how to use the A* architecture to do an informed search that builds the optimal path by moving between neighbor cells from the initial position S to the final goal position F. We can use heuristic search for this problem because the first theorem states that the perception cost is only a function of the distance from the last point of the path to the target. Thus the perception cost in this problem can be rewritten as

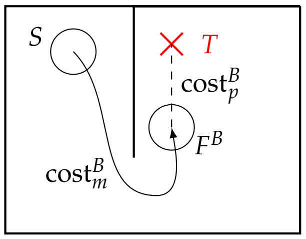
$$\text{cost}_p(\rho, T) = c_p\big(||s_N - T||\big). \tag{3.5}$$

Because the perception cost depends only on the last path position, it is possible to estimate the perception cost in the heuristic of a search algorithm.

Finally, due to the existence of a trade-off parameter λ, the optimal solution depends on the value of λ, as shown in Figure 3.1.



(a) Solution A with Lower λ



(b) Solution B with Higher λ

Figure 3.1: The cost of a path is given by the sum of the motion and perception costs, $cost_m$ and $cost_p$ respectively, and the optimal solution depends on the trade-off parameter λ.

3.2 PA*: Informed Search for Perception Planning

A* is a graph search algorithm that finds the lowest cost path from a given initial node to a goal node. It also works with discretized representations of the environments such as grid maps, where each grid position represents a node, and the graph connectivity can be either a 4 or 8-neighborhood. A* explores the environment by computing a heuristic cost function for each visited node that estimates the cost to reach the target. It moves through the search space by selecting the nodes with a lower overall cost.

In traditional motion planning, a node is a goal position if its coordinates are the same as the target coordinates. Moreover, the total cost estimate is given by the cost of the path from the starting position, S, to the current node n, plus a heuristic of the cost from n to the target position T.

$$f(n) = g(S, n) + h(n, T) \tag{3.6}$$

If the heuristic used is admissible, i.e., always less or equal than the true value, then the path returned is guaranteed to be optimal. Therefore, the usual choice for the heuristic is just the Euclidean distance between the current node and the target, without considering any obstacles.

$$h_{mp}(n, T) = ||n - T|| \tag{3.7}$$

In this section, we introduce PA* (perception A*), a heuristic search for motion planning that returns the optimal path to perceive a target, considering both motion and perception cost.

As in the A* algorithm, in PA* the total cost estimate is also given by the sum of $g(S, n)$, the path distance from the starting position S to the current node n, and $h(n, T)$, which here is a heuristic of both the motion and perception costs from n to T. In order for the heuristic to be admissible, it is based on the Euclidean distance between the current node and the target, without considering any obstacles.

The new heuristic in PA* still uses the straight line between the current node and sensing target, without considering obstacles, as A* does. Nevertheless, it now considers the expected cost of both for approaching the target and sensing from a smaller distance.

$$h_{pp}(n, T) = \min_{q} \Big(||n - q|| + \lambda c_p(||q - T||) \Big) \tag{3.8}$$

We assume that from position n the robot can still approach the target by moving to other location q, from where it senses the target. There is a trade-off between the possible increase of motion cost and the decrease in perception cost. We take the distance between points n and q as the approaching cost.

3.2.1 Optimal Heuristics for PA*

With the previously presented problem formulation, it is possible to solve the perception planning problem with A*, introducing a new heuristic that takes into account the perception cost. For that purpose, we will consider the problem of finding the optimal sensing position in a continuous straight line scenario without obstacles, as presented in Figure 3.2. The idea is to use the solution for the straight line problem without obstacles as an admissible heuristic for the perception problem.

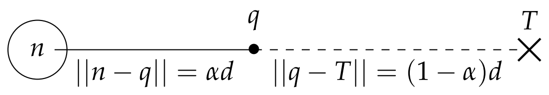


Figure 3.2: Given a robot at node n and a perception target T at distance d in a scenario without obstacles, the optimal sensing goal position lies in the straight line connecting those two points; the image shows a solution with a motion of αd and sensing distance equal to $(1-\alpha)d$.

With $||n-T|| = ||n-q|| + ||q-T|| = d$, the overall cost from node n to target T can be expressed as a sum of motion cost $|\alpha d|$ and perception cost $c_p\big(|(1-\alpha)d|\big)$:

$$\text{cost}(\alpha, d) = |\alpha d| + \lambda c_p\big(|(1-\alpha)d|\big), \tag{3.9}$$

where $\text{cost}(\alpha, d)$ is a continuous function of the overall cost when the robot is at a distance d to the target. The variable α represents the percentage of the distance the robot can approach the target, and $1-\alpha$ the percentage of the distance d that is sensed. In order to have the optimal solution, we need to find α that minimizes cost.

$$\alpha^* = \underset{\alpha}{\operatorname{argmin}}\, |\alpha d| + \lambda c_p(|(1-\alpha)d|) \tag{3.10}$$

Therefore, the first step is to find the percentage that minimizes the cost of approaching and sensing the target in a straight line. This percentage can be calculated analytically or numerically, depending on the function c_p. Later we will show optimal solutions for specific polynomial functions.

Lemma 3.1. *If c_p is a positive and monotonically increasing function, then $0 \leq \alpha^* \leq 1$, i.e., the optimal solution for the sensing goal position in the straight line perception task lies between the current node n and T.*

Proof. The cost of motion is always positive and increases with traveled distance. Using the previous definition for perception cost, it is a function of sensing distance; thus it is positive and increases monotonically. Thus, the assumption for the perception cost function holds. Assuming a straight line distance with size d, the optimal decision of approaching the target by $\alpha^* d$ and sensing from a distance $(1-\alpha^*)d$

has a minimal cost. By definition,

$$\forall \alpha, d \in \mathbb{R} \quad \text{cost}(\alpha, d) = |\alpha d| + \lambda c_p(|(1-\alpha)d|) \leq$$
$$|\alpha^* d| + \lambda c_p(|(1-\alpha^*)d|) = \text{cost}(\alpha^*, d). \tag{3.11}$$

If we take $\alpha > 1$, with a positive and monotonically increasing function c_p, then we have:

$$\forall \alpha > 1 \quad \text{cost}(\alpha, d) = \alpha d + \lambda c_p(d\alpha - d) > d + \lambda c_p(0) = \text{cost}(1, d). \tag{3.12}$$

Using equations 3.11 and 3.12, we prove that approaching the target by more than d always has a greater cost than approaching just by a distance d, showing that $\alpha^* \leq 1$. In other words, it is always preferable to approach the target by d and sense from that position than approach by a bigger distance while increasing the perception distance as well, by sensing the target from further away. The same analysis can be done for $\alpha < 0$,

$$\forall \alpha < 0 \quad \text{cost}(\alpha, d) = -\alpha d + \lambda c_p(d - \alpha d) > 0 + \lambda c_p(d) = \text{cost}(0, d), \tag{3.13}$$

proving that $0 \leq \alpha^* \leq 1$. □

There is an intuitive explanation for the previous lemma. If $\alpha^* > 1$, then the robot would move past the target and perceive it from behind, increasing both motion and perception costs, which contradicts the initial assumptions. If that were the case, there would be other solutions where the robot could sense from the same distance, but with a smaller cost of approaching the target, thus having an overall lower cost. That solution would also contradict the first theorem, as there would be a point in the middle of the path with a smaller distance to the target. The same reasoning can be applied when $\alpha^* < 0$, which also represents a counter-intuitive situation, because the robot would be moving away from the target, increasing both the motion and perception cost and again contradicting the first theorem.

The previous results are expected if we use positive and monotonically increasing cost functions, showing that the optimal solution to sense a target in a straight line is to approach the target while moving to a position in between the current node and the target. Moving beyond the target or moving back always yields solutions with a higher cost.

To find the solution for the straight line problem, we have to solve a minimization problem:

$$\alpha^* = \underset{\alpha}{\text{argmin}}\, \text{cost}(\alpha, d), \tag{3.14}$$

for $0 \leq \alpha^* \leq 1$. This minimization can be done by setting the gradient to zero in order to find extrema for the interior region, and if any point exists, comparing it to the cost values on the boundary to find the point with minimum cost (for $\alpha = 0$ the cost is $\lambda c_p(d)$ and for $\alpha = 1$ the cost is $\lambda c_p(0) + d$). If the function c_p has a gradient which is strictly increasing, then the overall cost is a convex function with only one minimum, and in that case, the optimal solution is either $\alpha^* = 0 \vee \alpha^* = 1 \vee \alpha^* = \alpha_c$, where α_c is given by

$$\frac{\mathrm{d}\big(|\alpha_c d| + \lambda c_p(|(1-\alpha_c)d|)\big)}{\mathrm{d}\alpha_c} = 0. \tag{3.15}$$

After finding α^* as the optimal solution for the straight line problem, the heuristic in PA* is

$$h_{pp}(n,T) = \alpha^*||n-T|| + \lambda c_p\big((1-\alpha^*)||n-T||\big). \tag{3.16}$$

If we find the optimal approach point q^* such as $||q^* - T|| = \alpha^*||n-T||$ (where q^* has to be in the straight line distance between n and T), then the heuristic can also be defined as

$$h_{pp}(n,T) = ||n-q^*|| + \lambda c_p(||q^*-T||). \tag{3.17}$$

In order to use the straight line solution as the best heuristic for PA* when not considering obstacles, as in the common A* formulation, we need to prove that the straight line solution yields the smallest cost.

Considering the scenario in Figure 3.3, the direct distance between robot position and target is d, as in the example before. However, because the approaching is not in a straight line with the target, $d' = ||n-q|| + ||q-T|| > d = ||n-T||$. Given the non-straight line assumption, $d' = d + \epsilon$, with $\epsilon \geq 0$. The robot moves a percentage of this path, $\alpha d'$, and senses the reamining distance, $(1-\alpha)d'$, where $\alpha = \frac{||n-q||}{||n-q||+||q-T||}$.

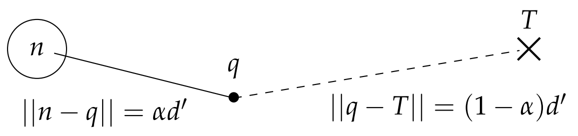


Figure 3.3: Robot approaches the target in a non-straight line to position q, with approach distance of $\alpha d'$, and the sensing distance of $(1-\alpha)d'$, with $d' > d$, where d is the straight line distance without obstacles.

Theorem 3.2. *If the heuristic in PA* uses the straight line solution, the heuristic is admissible and consistent iff the perception cost function $c_p(d)$ is zero for $d = 0$.*

Proof. The cost of the solution with $0 \leq \alpha \leq 1$ and $d' > 0$ is

$$\mathrm{cost}(\alpha, d') = \alpha d' + \lambda c_p((1-\alpha)d') = \alpha(d+\epsilon) + \lambda c_p((1-\alpha)(d+\epsilon)). \tag{3.18}$$

Again, if the perception cost function is monotonically increasing

$$\mathrm{cost}(\alpha, d') = \alpha(d+\epsilon) + \lambda c_p((1-\alpha)(d+\epsilon)) \geq \alpha d + \lambda c_p(d - \alpha d) \geq \mathrm{cost}(\alpha^*, d), \tag{3.19}$$

proving moving out of the straight line always has a higher cost than the optimal solution for the straight line, and as such, the straight line solution can be used as a heuristic for the informed search in perception planning.

As we assumed that $c_p(0) = 0$, then it is trivial to show that $h(T,T) = 0$.

Furthermore, if we consider a successor node n' with an optimal approach point q'^* and a cost to move from n to n' as $c(n,n') = ||n - n'||$, then

$$c(n,n') + h(n',T) = ||n - n'|| + ||n' - q'^*|| + \lambda c_p(||q'^* - T||). \tag{3.20}$$

Using first the geometric triangle inequality and then the definition of the heuristic,

$$c(n,n') + h(n',T) \geq ||n - q'^*|| + \lambda c_p(||q'^* - T||) \geq ||n - q^*|| + \lambda c_p(||q^* - T||) = h(n,T). \tag{3.21}$$

Because $h(T,T) = 0$, this heuristic is consistent. □

We use $||.||$ as the Euclidean distance. As we have shown here, the heuristic for PA*, h_{pp}, depends only on the distance to the target $d = ||n - T||$ and the parameter α^*, which depends on the perception cost function c_p and the distance d as well. Regarding the cost function $g(S,n)$, it is the same as standard A* is used, with g being the minimum cost for a robot to move from the initial position S to node n.

We consider the perception quality depends only on the distance to the target because we assume robots can rotate in place and sense the target from the most favorable direction. Furthermore, the perception cost function c_p can be any monotonically increasing function, allowing flexibility to represent the cost of multiple perception models.

3.2.2 Perception Cost Function Examples

For any specific perception cost, it is possible to find the optimal sensing position q^* and the parameter α^* as a function of the distance $||n - T||$ and the function c_p. With α^* known before-hand, the heuristic h_{pp} becomes only a function of n and T, and easy to compute during the search execution.

We give in this section two examples for the function c_p, where the perception cost is either a linear or quadratic function of the distance to the target. In our model, we assume circular omnidirectional sensing with a limited range r_p. We then formulate the heuristic $h_{pp}(n,T)$ in a way that it can be used for multiple perception cost functions while being easily computed for each specific perception function.

First, we define the optimal sensing distance as $d_s^* = ||q^* - T|| = (1 - \alpha^*)||n - T||$, when $||n - T|| \to \infty$. This definition is useful when the overall cost functions are convex, with a perception cost whose gradient is strictly increasing, which is true for the functions we exemplify in this section. In that case, the optimal sensing distance is a constant, and a unique solution that only depends on the function c_p, not depending on the distance $||n - T||$. In this scenario, the real sensing distance used in the heuristic h_{pp} is either d_s^* or the boundary solution $||n - T||$ if $||n - T|| < d_s^*$.

In the linear case, the function c_p is linear, apart from a limit related to the maximum perception range.

$$c_p(d) = \begin{cases} |d| & |d| \leq r_p \\ \infty & |d| > r_p \end{cases} \tag{3.22}$$

There are two cases for the optimal sensing distance d_s^* in the linear case:

- $\lambda < 1$: The cost of motion is greater than the cost of sensing, so the robot minimizes motion by sensing form as far apart as possible (limited by maximum sensing range r_p);
- $\lambda \geq 1$: The cost of sensing is higher than the cost of motion, so the robot moves as close to the target as possible.

Given the analysis before, it is possible to write the optimal sensing distance d_s^* for the linear perception cost function.

$$d_s^* = \begin{cases} r_p & \lambda < 1 \\ 0 & \lambda \geq 1 \end{cases} \tag{3.23}$$

With a quadratic sensing cost, $c_p(||q - T||) = ||q - T||^2$, we can solve for α^* using equation 3.15. In order to find the minimum, we find the point α_c with zero derivative.

$$\frac{\mathrm{d}\left(\alpha^* d + \lambda(d - \alpha^* d)^2\right)}{\mathrm{d}\alpha^*} = 0 \Leftrightarrow \alpha^* = 1 - \frac{1}{2d\lambda} \tag{3.24}$$

Ideally, the robot would move to a fixed distance $1/(2\lambda)$ of the target to sense it optimally.

The optimal sensing distance d_s^* for the quadratic perception cost function also depends on r_p.

$$d_s^* = \begin{cases} \frac{1}{2\lambda} & 1/(2\lambda) \leq r_p \\ r_p & 1/(2\lambda) > r_p \end{cases} \tag{3.25}$$

We can interpret the quadratic c_p as a function that represents a cost that changes little with close distances. When the robot is already close enough to the target, changes in the distance to the target have a small impact on the perception cost. On the other hand, the further away the robot is from the target, the greater the impact of the distance in the perception cost. Therefore, it is reasonable to think of a fixed optimal sensing distance in that case, which is a function of λ, the parameter that weights motion and perception cost.

Having calculated the optimal sensing distance d_s^* and checking the boundary condition, we can write the heuristic as a function of n, T and d_s^*.

$$h_{pp}(n,T) = \begin{cases} (||n-T|| - d_s^*) + \lambda c_p(d_s^*) & ||n-T|| \geq d_s^* \\ \lambda c_p(||n-T||) & ||n-T|| < d_s^* \end{cases} \tag{3.26}$$

For nodes from which the target cannot the perceived, the heuristic value becomes infinite.

Other perception cost functions with similar properties (convex functions with unique solutions) can use the same heuristic equation, with the only requirement of finding d_s^*.

The solution to the perception planning problems depends significantly on the sensor used. That is the reason we approached this problem in a more general perspective first, introducing a heuristic that works for any perception cost function, as long as it is an increasing function with distance. The specific cases presented in this section are examples of what needs to be done to determine the heuristics for specific cost functions. It is possible to adapt the cost functions to the problem in hand (e.g., sensor properties), and then determine the heuristic for that specific problem using our approach.

3.3 Underlying Graph and Node Expansion

In the solution presented until now, we assume there is a grid-like discretization of the world which serves as the underlying graph for search. Every grid point is a node in the graph, and two nodes are connected if their respective grid positions are neighbors, assuming 8-connectivity as the adjacency rule. Due to the existence of obstacles between grid positions, not all of them are reachable by the robot.

As explained before, nodes are expanded using a heuristic that estimates how much the robot should approach the target to have an optimal path. If there were no obstacles in the PA* problem, the search would expand nodes until reaching the final grid position F. In that case, unless $F = T$, the heuristic for node F has an approaching distance equal to zero, i.e., $\alpha^* = 0$, which can be a stopping condition. However, in that case, the heuristic in the final expanded state would be non-zero if $F \neq T$.

Furthermore, if we also consider the existence of obstacles, another problem arises. As the search progresses, assuming in current node n the optimal α^* is greater than zero, the algorithm has to continue the search because the possibility of approaching the target yields a solution with lower cost. The meaning of this situation is that at the current node n the solution is not guaranteed to be optimal, and there is the change to find another node with a lower overall cost for both motion and perception. As the search progresses, the nodes that could in principle have a lower cost might be blocked by obstacles, or not have line-of-sight with the target.

Therefore, it is essential that the algorithm can go back to the previous node n if it has a global lower cost after exploring other alternatives. While for this node n there will always be a positive optimal approaching distance, after exploring the rest of the graph and increasing the lower bound for the overall cost, this will be the node minimizing the cost as all the other possibly better alternatives have already been tested and are not valid final positions. However, the capability of backtracking to previous nodes is not possible in the graph described so far.

We thus propose an extension to the original underlying graph that is built from the grid discretization, as shown in Figure 3.4. On this extension, we reuse a solution from other robotic problems, where one more node, T_e, is added to the original graph. The cost between nodes in the original graph is $c(n, n') = ||n - n'||$, and they stay the same in the extended graph. However, the extended graph also has new connections between every node n and the new node T_e such as their cost is $c(n, n' = T_e) = \lambda c_p(||n - T_e||)$. For the search to finish, the additional node T_e has to be expanded, and in that final expanded node the heuristic value is zero, i.e., $h_{pp}(n = T_e, T) = 0$. Therefore, when expanding node n, not only its grid neighbors are added to the priority queue with the respective priority $f(n')$, but also the node T_e is added with priority $f(T_e) = g(S, T_e) = g(S, n) + \lambda c_p(||n - T_e||)$, where n is the last grid position expanded.

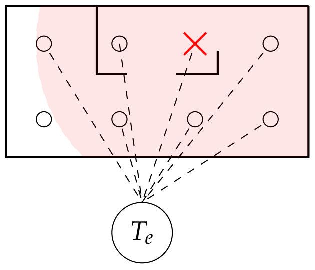


Figure 3.4: Extending the underlying graph with an additional node T_e, which must be reached to finish the search.

3.3.1 Stopping Condition

The stopping condition becomes trivial if we use the extended graph described before. For the search to stop with a feasible path for the robot, the final step is to expand node T_e. In that case, the search ends after expanding a node with heuristic equal to zero. Moreover, until now obstacles which occlude the target have not been considered. So, when expanding the node T_e, it is necessary to test with ray casting if there is line-of-sight between the last position of the path in the grid and the target. If the evaluation is successful, the algorithm found the optimal path and the search stops.

As there can be multiple entries in the priority queue for the node T_e which come from connections with different grid positions, it is necessary to memorize for each one what was the last grid position m

visited. When the ray casting test results in a non-obstructed line-of-sight to the target, the position m becomes F, and the g values are used to find that optimal path from S to F, exactly as done in A* for motion planning.

Furthermore, it is possible that the ray casting fails and the search has to continue. In that case, even though the heuristic is consistent, the node T_e can be expanded multiple times. All the other nodes from the original graph built from the grid can never be expanded more than once, and as such, they go into a closed list of nodes that is never expanded again.

Finally, in the new extended graph, the heuristic in the *goal state* is always zero, so we can drop the constraint $c_p(0) = 0$ in order to have a consistent heuristic. The multiple expansion of T_e may seem contradictory to the fact that the heuristic is consistent, but we can explain it as an *error* in the cost of connections to the new node T_e. When ray casting fails from node n to T, the cost of the connection from node n to node T_e should have been infinite instead of $\lambda c_p(||n - T||)$. Therefore, the expansion of T_e from occluded nodes should not have existed, and if that were the case, the first valid expansion would have terminated search.

As we illustrate in Figure 3.5, there are candidate and feasible final path positions. For all the nodes outside of the circle, the connection to node T_e is infinity. For the grid positions inside the circle, the cost of the connection to T_e depends on the distance to the target, which is known and used when building the extended graph. When the distance to the target is more than the maximum sensing range r_p, the connection to T_e is infinity. Therefore unfeasible sensing distances are never considered during search.

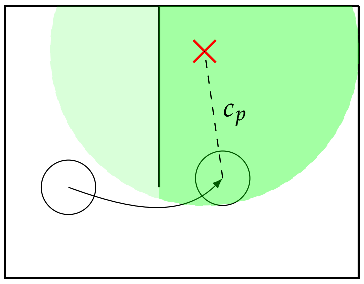


Figure 3.5: Given a sensing target and a maximum perception range r_p, there is a set of candidate goal positions (green circle with radius r_p); from those positions, not all are feasible because obstacles can block line-of-sight to the target; the darker green represents feasible goal positions.

The feasible positions are determined using ray casting. That operation could be done in the beginning too in order to have a correct cost in the connections to T_e (again, the cost is infinity to non-feasible final path locations). However, ray casting is an expensive operation, so we only test for feasibility after expanding T_e, updating the cost to infinity in case ray casting fails during search, meaning there is no line-of-sight from that position to the target.

In conclusion, the stopping criteria are 1) expansion of node T_e (with a connection from node n) and 2) feasible line-of-sight from n to T (determined with ray casting). After successfully testing the stopping criteria, the last considered grid position n with a connection to the target becomes the final position F.

3.4 Perception Planning Assumptions

In the algorithm for perception planning presented until now in this chapter, we model some assumptions both in terms of the robot motion and its sensors. In this section, we clarify the list of assumptions we make and their consequences in terms of applicability of this planning technique.

First, we discretize the world in a 2D grid map, but robots move in a continuous space. Therefore, the optimal solutions we compute are only optimal solutions in the discretized world, but not necessarily optimal in the continuous world where robots do actually move. However, we believe that using a fine-grained discretization mitigates that problem, especially when considering robots with an omnidirectional drive. For other types of drives, a carefully designed grid neighbor connectivity, together with a fine enough resolution, is also enough to have a discretization that is good enough, while greatly facilitating the planning effort.

Second, in this planning framework, we consider sensors with 360 degrees of field of view, which is not always realistic. Many sensors have a limited field of view, or they may even be mounted in a way such as they have blind spots. In this planning framework, for the sake of simplicity, we consider omnidirectional robots with a footprint close to circular, where this assumption is not relevant because robots can rotate and sense from a different angle in order to mitigate the possible blind spots of their sensors. However, even if our assumptions for the robot footprint and motion model do not hold, there is nothing in our proposed planning algorithm that makes it impossible to deal with sensors with blind spots and different fields of view. We can cope with those cases by taking some considerations both in the perception cost function c_p and in the perception estimate of the heuristic.

As an example, let's assume we have a sensor with different maximum ranges in different directions, a $c_p(\theta, distance)$ perception model that is a function not only of distance but also perception angle θ, and a blind spot in the $\theta = 45^\circ$ direction relative to the front of the robot. To deal with this specific sensor, our heuristic would still be admissible as long as we create a new perception cost function such as $c'_p(distance) = min_\theta\big(c_p(\theta, distance)\big)$. By taking the minimum function, we guarantee that the heuristic estimate is always less than the real cost, thus guaranteeing optimal solutions, at the expense of search not being very efficient. In terms of blind spots, before testing the feasibility of perception with ray casting, we would consider the angle of perception, and fail the feasibility of perception if the angle of perception is $\theta = 45°$. Another approach to deal with these types of sensor models is to extend the graph of the PA* algorithm from an xy grid to an $xy\theta$ graph that also discretizes orientation. If we were to consider orientation, a traditional solution is to first run easier searches in the xy 2D grid and use the results in heuristics of the final problem that considers orientation as well. This approach would also justify the need of an efficient search method in the more straightforward xy space, and that is also why we optimize as much as possible the PA* variants in a later chapter that uses robot-dependent maps as a pre-processing step.

Third, we assume that there is only one sensor. However, most robots usually use multiple sensors, and in our contributed planning framework we did not address that possibility. Nevertheless, we can discuss assumption in the same way we did for complex sensors. By combining all the sensor characteristics into a c_p function to model perception, we can take the minimum cost of each sensor in each direction, and come up with a function that underestimates the true perception cost such as it can still be used in PA* and guarantee optimal solutions. However, it is trivial to underestimate the cost greatly, but that results in inefficiencies and slower searches. Therefore, the goal would be to find a c_p function that underestimates the cost, but still is as close as possible to the real cost in order to have an efficient search algorithm.

Finally, we assume there are no explicit constraints in terms of battery life in our optimization problem. However, by giving cost to motion, and considering the battery life is correlated with the amount of motion of a robot, we indirectly maximize battery by minimizing the robot motion in a perception task. But given our cost function also tries to reduce the perception cost, and that we have no constraint on the maximum motion of the vehicle, we cannot guarantee that solutions from PA* will not make the robot run out of battery while executing a perception task.

3.5 Running Example of PA* Search

We show in Figure 3.6 two PA* searches with different values for the trade-off parameter λ. In Figure 3.6a we have a simple environment where black lines represent obstacles. We assume the robot can only be in eight positions (represented with circles, where the big circle in the upper left corner is the robot in its initial position). The vertical and horizontal distance between grid points is one unit. From those eight grid positions, only six are reachable, which are connected two the initial position with the dashed lines. The red cross shows the target position, and the red region represents a circle with radius r_p, the maximum sensing range. Therefore, all the points inside the red region could perceive the target if there were no obstacles. The bottom left grid point is the only position from where the target cannot be sensed.

The perception function is quadratic, with $c_p(||n - T||) = ||n - T||^2$ when $||n - T|| < r_p$, and infinity

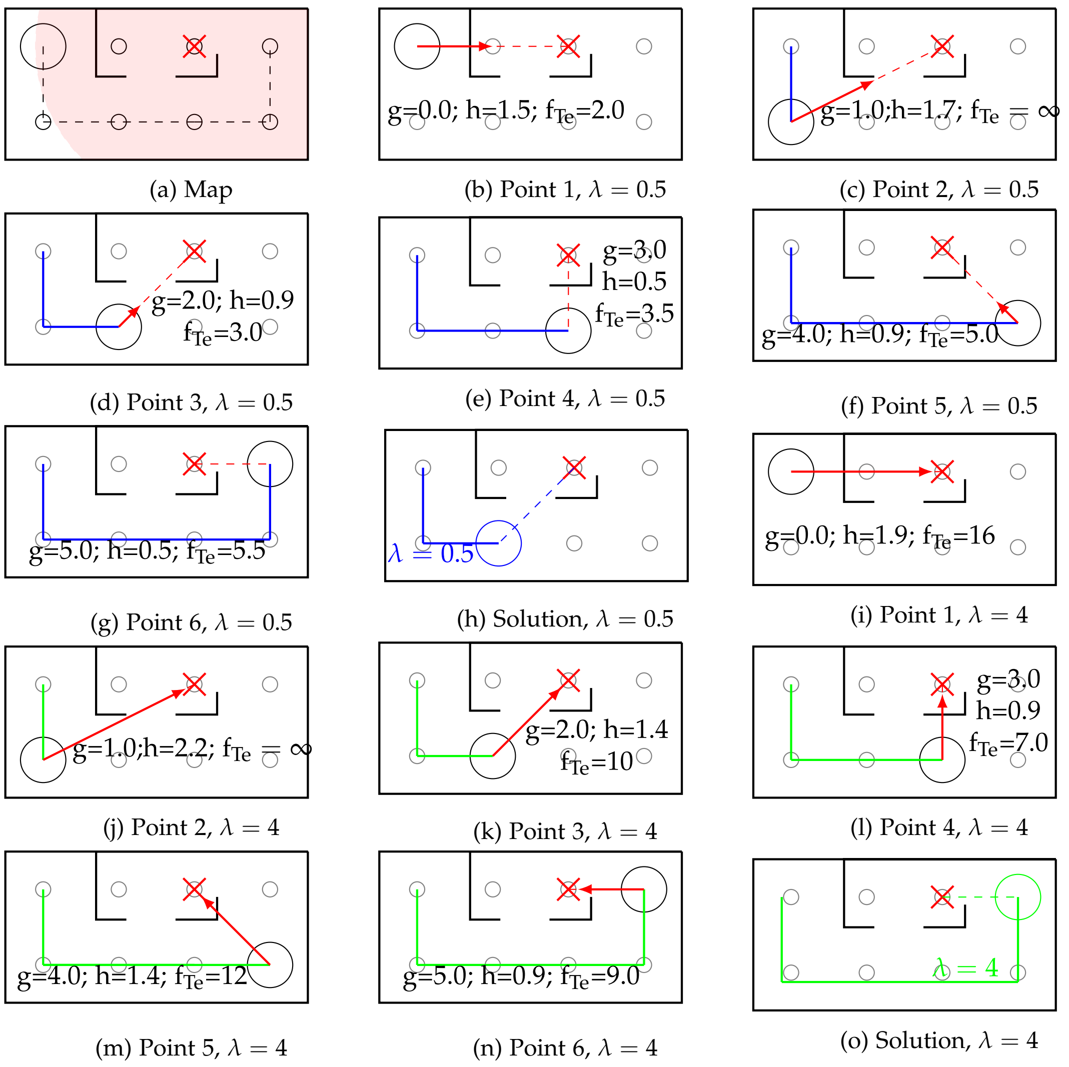

(a) Map
g=0.0; h=1.5; f$_{Te}$=2.0
(b) Point 1, $\lambda = 0.5$
g=1.0;h=1.7; f$_{Te}$ = ∞
(c) Point 2, $\lambda = 0.5$
g=2.0; h=0.9
f$_{Te}$=3.0
(d) Point 3, $\lambda = 0.5$
g=3.0
h=0.5
f$_{Te}$=3.5
(e) Point 4, $\lambda = 0.5$
g=4.0; h=0.9; f$_{Te}$=5.0
(f) Point 5, $\lambda = 0.5$
g=5.0; h=0.5; f$_{Te}$=5.5
(g) Point 6, $\lambda = 0.5$
$\lambda = 0.5$
(h) Solution, $\lambda = 0.5$
g=0.0; h=1.9; f$_{Te}$=16
(i) Point 1, $\lambda = 4$
g=1.0;h=2.2; f$_{Te}$ = ∞
(j) Point 2, $\lambda = 4$
g=2.0; h=1.4
f$_{Te}$=10
(k) Point 3, $\lambda = 4$
g=3.0
h=0.9
f$_{Te}$=7.0
(l) Point 4, $\lambda = 4$
g=4.0; h=1.4; f$_{Te}$=12
(m) Point 5, $\lambda = 4$
g=5.0; h=0.9; f$_{Te}$=9.0
(n) Point 6, $\lambda = 4$
$\lambda = 4$
(o) Solution, $\lambda = 4$

Figure 3.6: PA* search for two values of λ and quadratic perception cost: map has eight grid positions, from which six are reachable by the robot; dashed black lines represent connectivity between feasible robot positions; the maximum sensing range is shown with the red circle around the target (red cross), so all grid points except one are close enough to perceive the target; each grid point, which is a graph node, has a $g(S, n)$ cost and a heuristic estimate $h(n, T)$; each grid point is connected to the node T_e of the extended graph; the priority value of T_e with a connection to each node is shown as f_{T_e} in the respective image; red arrows represent the optimal approach distance for the heuristic in each node, and the red dashed lines represent the optimal perception distance in the heuristic estimate for each node.

otherwise. In the first example, the weight parameter $\lambda = 0.5$, so the optimal sensing distance is $d_s^* = 1$. For each point, the heuristic cost represents the minimal weighted sum of an approach distance and the cost of a perception distance. The approach distance, $\alpha^* d$, is represented with a red arrow, and the perception distance for the heuristic, $(1 - \alpha^*)d$, is represented with a red dashed line.

The reachable points are all connected in a line, and as shown in Figure 3.6 they are Point 1 to Point 6, which are nodes n_1 to n_6 in the graph representation. To find the optimal path from n_1 to perceive the target, the algorithm starts by expanding n_1 in Figure 3.6b. This node is not the goal because the search only stops after expanding the node T_e in the extended graph. The successors of n_1 are node n_2 with priority $f_2 = 2.7$ $(= g_2 + h_2 = 1 + 1.7)$, as shown in Figure 3.6c, and node T_e with priority $f_{T_e} = 2$, as shown in Figure 3.6b. These two nodes are added to the priority queue. Because T_e has a lower priority, it is the next expanded node, and the stopping criteria are tested with ray casting but fail because there is no line-of-sight to the target from n_1 as shown in Figure 3.6b.

In the next iteration, the only node in the priority queue is n_2, which is expanded. The next node added to the priority queue is n_3 with priority $f_3 = g_3 + h_3 = 2 + 0.9 = 2.9$, because it is the only grid neighbor to n_2 that was not expanded before. The connection to T_e from n_2 has infinite cost because it is beyond the maximum sensing range to the target. As such, T_e is not added to the priority queue this time.

The next and only node in the priority queue is n_3, shown in Figure 3.6d, which adds T_e to the priority queue again, now with priority (3.0). In this iteration, the node n_4 is also added with priority (3.5). There are now two nodes in the priority queue, and the first expanded node is again T_e because it has the lowest priority. The algorithm uses ray casting and finds that n_3 has no obstructions and can perceive the target. Therefore, the stopping criteria are met and search stops. Node n_3 is the final position F, and the optimal path to perceive this target with $\lambda = 0.5$ is $\rho = \{n_1, n_2, n_3\}$, as shown in Figure 3.6h, with cost of 3 units. Due to the specific parameters used, nodes n_4 to n_6 shown in Figures 3.6e, 3.6f and 3.6g are never expanded.

As a side note, it is possible to see in Figures 3.6e and 3.6g that the heuristic distance for perception takes the total distance from the node to the target, and as such $f = g + n = f_{T_e}$.

In the second example, $\lambda = 4$, making the perception cost bigger in comparison to the motion cost, and as such it is expected for the optimal solution to move closer to the target to reduce the perception distance. Again, in order to find the optimal path from n_1 to perceive the target, the algorithm starts by expanding n_1 in Figure 3.6i. This node has successors node n_2 with priority $f_2 = 3.2$ $(= g_2 + h_2 = 1 + 2.2)$, as shown in Figure 3.6j, and node T_e with priority $f_{T_e} = 16$, as shown in Figure 3.6i. These two nodes are added two the priority queue.

Because n_2 has a lower priority, it is the next expanded node. The next node added to the priority queue is n_3 with priority $f_3 = g_3 + h_3 = 2 + 1.4 = 3.4$, because it is the only grid neighbor to n_2 that was not expanded before. The connection to T_e from n_2 has infinite cost again, not being added to the priority queue.

The next node in the priority queue with the lowest priority is n_3, shown in Figure 3.6k, which is expanded and adds T_e to the priority queue again, now with priority 10. In this iteration, the node n_4, the only not expanded neighbor, is also added to the queue with priority 3.9. There are now three nodes in the priority queue, n_4 with priority 3.9 and T_e twice with priorities 16 and 10. The next expanded is n_4, shown in Figure 3.6l, which adds again T_e with priority 7.0, and n_5 with priority 5.4, as shown in Fig. 3.6m.

In the next iteration n_5 is expanded, adding T_e to the priority queue with priority 12 and node n_6 with priority $f_6 = 5.0 + 0.9 = 5.9$. Again all the nodes T_e added to the priority queue have a higher cost, so the next expanded node is n_6, from Figure 3.6n, and T_e is added one last time, with priority 9. At this point the priority queue has five nodes, all of them being T_e, with priorities 7, 9, 10, 12 and 16. The first expanded node now is T_e with a connection from n_4, shown in Figure 3.6l, but the stopping criteria fail with ray casting showing that obstacles block line-of-sight to the target. Finally, the node T_e with a connection from n_6 and priority 9 is expanded, and search stops as n_6 has line-of-sight with the target.

Having met the stopping criteria, node n_6 is the final position F, and the optimal path to perceive this target with $\lambda = 4$ is $\rho = \{n_1, n_2, n_3, n_4, n_5, n_6\}$, as shown in Figure 3.6o, with cost of 9 units. As we can see, the priority of expanded nodes always increases, which is a characteristic of consistent heuristics.

For $\lambda = 3$, both solutions with paths stopping in n_3 and n_6 have the same cost, with eight units.

3.6 Summary

In this chapter we introduced the problem of motion planning for perception tasks, considering both motion and perception costs. We proposed PA*, an approach extended from A*, and contributed heuristics to solve the planning problem, proving their admissibility. The heuristics use the minimal cost of motion and perception in a straight line, not considering obstacles. The methods proposed are general for any perception and motion costs, as long as they are monotonically increasing functions. The main novelty of this contribution is the computation of heuristics that take into account the perception cost. Moreover, the perception cost itself comes from a model of the specific sensor of each robot.

The algorithm determines the optimal path using an informed search strategy. It looks for the most promising solution first and directs search towards the target, reducing node expansion as much as possible. Moreover, the algorithm only expands goal nodes when they have the least cost compared to all other possible nodes, which reduces the number of ray casting operations as well.

We proved the presented perception planning algorithm yields optimal solutions, by showing that the contributed heuristics are consistent and thus admissible. We also provided two examples of perception cost functions, (1) a cost function that increases linearly with distance, and (2) a perception function whose cost increases quadratically with perception distance.

Chapter 4

Perception Planning with Visibility Maps

In the previous chapters, we introduce robot-dependent maps and PA*, a heuristic search for perception path planning. The robot-dependent maps provide a transformation that quickly gives information on the feasibility of coverage and perception tasks. Specifically for the robot-dependent visibility maps, we introduced the concept of critical points, from where visibility inside unreachable regions can be determined.

In this chapter we combine those two techniques, using information extracted from robot-dependent maps to improve the efficiency of node expansion in heuristic search for perception planning. When computing the robot-dependent maps, we determine the position of critical points, and we show here how to use the critical point position to obtain information on the minimum motion and perception distance. The goal of using additional information is to build better heuristics that result in fewer node expansion and ray casting operations, enabling faster computation of solutions for path planning problems that deal with the perception of a target position.

Using the information on the minimum motion and perception distance, we construct new heuristics that are still admissible, but we can also prove that they are always more informative than the base PA* heuristic. Thus, the new heuristics produced in this chapter also result in optimal paths. Moreover, we improve the goal stopping condition as well with information from critical points, to reduce the number of ray casting operations. In our previous work we show that with an initial fixed cost of building the visibility map, it is possible to use the critical points from that transformation to improve the search heuristic of PA* for multiple search instances [33, 38].

4.1 Regions from Robot-Dependent Maps

The visibility map gives information on the feasibility of perception, while not giving any information about the positions from where targets can be perceived. Nevertheless, the transformation provides structured information about the environment, and it is possible to separate grid points into three categories:

1. **Navigable Space**: points that can be reached by the robot center, $Nav(S)$;
2. **Actuation Space**: points that can be "touched" by the robot footprint, $A(S)$;
3. **Unreachable Regions**: points the robot cannot cover with its footprint and motion only, because they lie in positions not traversable by the robot, $U(S)$.

The second category is a superset of the first. While there is no perception information for targets in the first category, it is possible to gain information about points in the second category only, i.e., $T \in$ (Actuation Space) $\setminus$ (Navigable Space). We know these targets T have a distance to the navigable set not larger than the robot size. Therefore, with a small search bounded by the robot size, it is possible to find the closest point p in the navigable set minimizing the distance to the target t. The distance $||p - T||$ is a lower bound for the perception distance. However, in this specific case, the gained information will probably have only negligible effects on the search efficiency.

The third category of points, those that belong to the unreachable regions, is the category with the most considerable benefits in terms of information gain from the Visibility Map transform. Targets in the unreachable regions have associated a critical point, which gives information about a possible position from where targets can be sensed. Furthermore, we can have a better estimate of the perception distance in the heuristic for perception planning if we consider the distance between a target in region $U^l(S)$ and its corresponding critical point $c^*_{li}(S)$. Therefore, we will focus our discussion only on positions that belong to the third category, *Unreachable Regions*.

4.2 Improved Heuristics

Considering the base heuristic of PA*, independently of the perception cost function, we know it is associated with the cost of approaching the target to sense it from a better sensing position, reducing the perception cost. Moreover, the heuristic does not consider obstacles, and the best sensing position lies in the straight line between the current node n and the target T. We assume we can solve the heuristic minimization problem (equation 3.8) for a specific cost function c_p, and find the optimal sensing distance

d_s^*. Assuming $0 \leq d_s^* \leq ||n - T||$, the heuristic becomes

$$h_{pp}(n, T) = ||n - T|| - d_s^* + \lambda c_p(d_s^*). \tag{4.1}$$

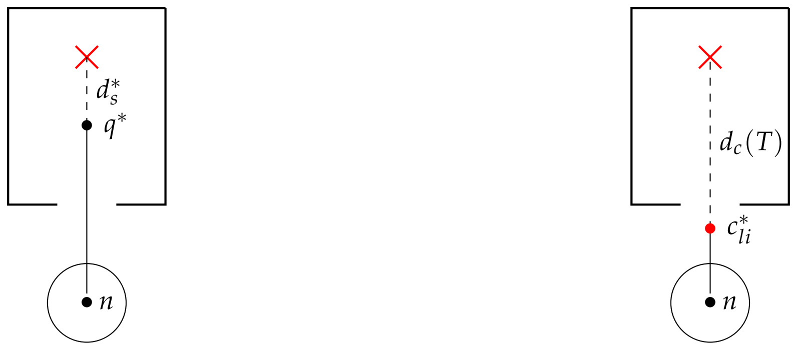
4.2.1 Using Perception Distance from Critical Points

As presented in a previous chapter, critical points are feasible robot positions that maximize the visibility inside each unreachable region.

From the visibility map, the critical points can provide information about the minimum sensing distance from any point in the navigable space to a point in the unreachable region. The heuristic can be updated and use the minimum sensing distance from the critical point instead of the optimal sensing distance d_s^* from the straight line solution, as shown in Figure 4.1. When using the visibility map, and being the distance from the critical point to $T \in U^l(S)$ ($\equiv T^l$) equal to $d_{li}^c(T) = ||T - c_{li}^*(S)||$, the heuristic becomes a function of $d_l^c(T^l) = \min_i d_{li}^c(T^l)$.

$$h^1(n, T^l) = \begin{cases} ||n - T^l|| - d_l^c(T^l) + \lambda c_p(d_l^c(T^l)) & \forall ||n - T^l|| \geq d_l^c(T^l) \\ \lambda c_p\left(d_l^c(T^l)\right) & ||n - T^l|| < d_l^c(T^l) \end{cases} \tag{4.2}$$

The equation for h^1 applies to the case $d_l^c(T^l) \geq d^*$; otherwise, the original heuristic h_{pp} is used. Here we are not considering the possibility that d_l^c is bigger than r_p, but in that case the target would not be visible. In order to use this heuristic as admissible and guarantee an optimal path, we only need to prove the distance from unreachable T^l to any other point in the navigable space is larger than $d_l^c(T^l)$.



(a) PA* Base Heuristic (b) h_1 Improvement

Figure 4.1: Impact of using the distance to the critical point, $d_c(T)$, as the guaranteed minimum perception distance on the improved heuristic h_1.

In the unreachable regions, the minimum sensing distance is the smallest distance from the target to the corresponding critical point. Again, that distance can be used as the minimum sensing distance in the PA* heuristic to improve the search speed.

Theorem 4.1. *The distance of points inside unreachable regions to the critical point is minimal in comparison to distance to any other point in the navigable space.*

Proof. We assume only one critical point and frontier, for the sake of simplicity. As shown in Figure 4.2, we consider only the frontier extremes, the two obstacles at points $O_1(0, -\zeta)$ and $O_2(0, \zeta)$, with $\zeta < R$, being R the robot radius. The frontier is between those two obstacles. If the robot starts at some point with $x > 0$, then the unreachable region consists of points with $x \leq 0$. If there were other obstacles besides the points O_1 and O_2, there might be unreachable points with $x > 0$, but those are not relevant to this proof, and as such, we kept the minimum number of obstacles.

If we use the same reasoning, we can conclude that only points of the navigable space with $x > 0$ are relevant because those are the only ones that can be used to have visibility inside the unreachable region.

Following this description, the critical point results as the point that is at R distance from both obstacles, $(\sqrt{R^2 - \zeta^2}, 0)$. For any point (a, b), with $a < 0$, the distance to the critical point has to be the minimum distance between (a, b) and any position in the navigable space, (α', β'), with $\alpha' > 0$. As we can see in Figure 4.2, for any point (α', β') there is a point (α, β) in the border of reachability that has a lower distance to (a, b). The distance between (a, b) and (α, β) is d.

$$\begin{aligned} d^2 &= (\gamma + R\cos\theta)^2 + (R\sin\theta)^2 \\ &= \gamma^2 + R^2\cos^2\theta + 2\gamma R\cos\theta + R^2\sin^2\theta \\ &= \gamma^2 + R^2 + 2\gamma R\cos\theta \end{aligned} \tag{4.3}$$

As we can see from the equation, increasing the angle θ minimizes the distance, and at the critical point, the angle θ reaches its maximum value. Thus, we prove the critical point minimizes the distance to any unreachable target. □

Only minimal errors exist due to discretization. While in the continuum space there is only one point that minimizes the distance to all frontiers, in the grid map, there might be two points that minimize the distance, with the same cost. If we take into account the discretization error when determining the minimal distance, then it is possible to use the distance to the critical point to still obtain an admissible heuristic. Therefore, we only have to subtract the quantization error from the distance to the critical point to get an admissible heuristic.

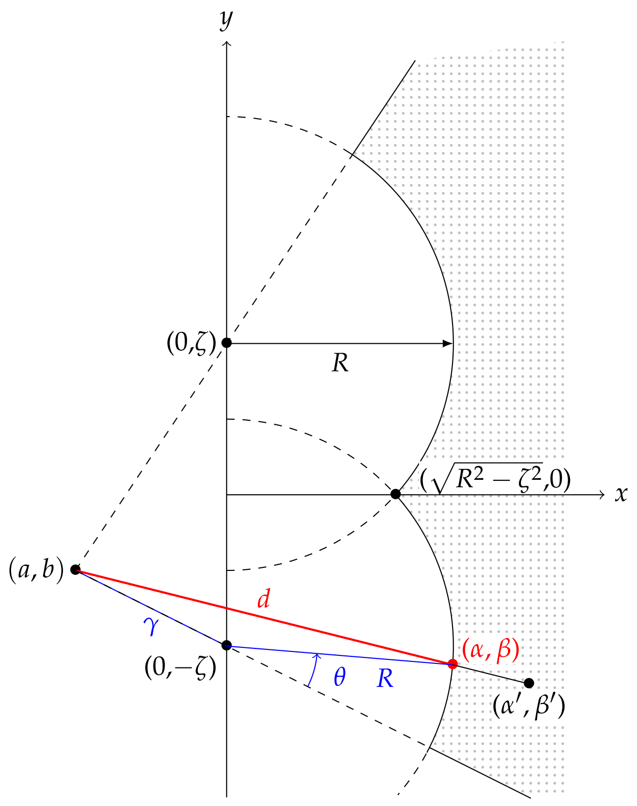


Figure 4.2: The filled regions represents the set of points in the navigable space that can sense the point (a,b), given two obstacles at positions $(0,\zeta)$ and $(0,-\zeta)$; the critical point $(\sqrt{R^2-\zeta^2},0)$ is the point with the minimum distance to any (a,b) in the unreachable region.

4.2.2 Using Critical Point Perception Distance in Goal Stopping Condition

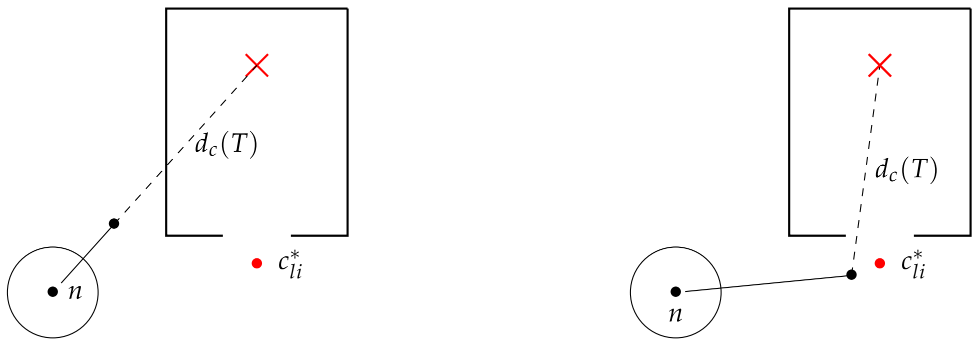
With heuristic h_1 we can also use the distance to the critical point, d^c, for the stopping condition. Given that d^c is proved to be minimal, we can change the cost of connections to node T_e if their distance is greater than the critical point distance to the target, reducing the number of points to be tested with ray casting.

$$c(n,T_e^l)=\infty \quad , \quad ||n-T^l||<d_l^c(T^l) \tag{4.4}$$

4.2.3 Using Critical Point Position to Estimate Minimum Motion Cost

However, it is still possible to improve the proposed heuristic. Instead of using the critical point to have only a lower bound estimate on the perception distance, we can use it to estimate a lower bound for motion cost as well, as shown in Figure 4.3.

At first, we assume the optimal sensing distance d_s^* is lower than the distance to the critical point, d_{li}^c. Thus, from any position with line-of-sight to the target (robot at point $x>0$ in Figure 4.4), the robot will move to a point as close as possible to the unreachable target, i.e., a border of the navigable space.



(a) h_1 Inefficiency (b) h_2 Improvement

Figure 4.3: Impact of using the critical point location for a better estimate of the motion cost on the improved heuristic h_2.

Therefore, we know that the minimum motion cost will be the distance between the current node n and the closest point in the border of the navigable space. From Figure 4.4, we can see that in the worst case scenario, the distance between the critical point and any other point of the navigable space border, with line-of-sight to the target, is $2R$. Thus, the new admissible heuristic becomes

$$h^2(n, T^l) = \min_i \Big(\max(||n - c^*_{li}(S)|| - 2R, 0) + \lambda c_p(d^c_{li}(T^l)) \Big). \tag{4.5}$$

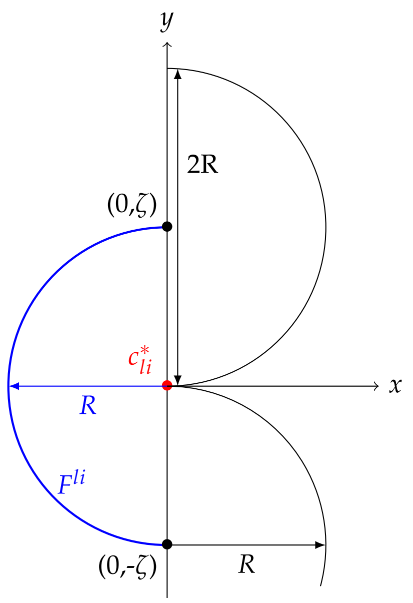


Figure 4.4: Worst case scenario for the distance between critical point and the border of navigable space (the two half circumferences with $x > 0$), from Figure 4.2; for the worst case, the distance between obstacles points is precisely the diameter of the robot, and the further point in the border of the navigable space is at distance $2R$ from the critical point.

We can also update the heuristic to consider $d^*_s > d^c_{li}(T^l)$, using $\delta = \max(d^*_s - d^c_{li}(T^l), 0)$.

$$h^2(n, T^l) = \min_i \Big(\max(||n - c^*_{li}(S)|| - 2R - \delta, 0) + \lambda c_p(d^c_{li}(T^l)) \Big) \tag{4.6}$$

Finally, the first heuristic h^1 might be a better estimate in cases there is line-of-sight between n and T, so to always use the best heuristic, we choose the one closest to the true value, considering they are both admissible.

$$h^{AVM}(n, T^l) = \max(h^1(n, T^l), h^2(n, T^l)) \tag{4.7}$$

Theorem 4.2. *Heuristic using critical points dominates original heuristic in PA*.*

Proof. The original heuristic $h_{pp}(n, T)$ in PA* is always less than the real cost, because it uses the optimal solution for the Euclidean distance without any obstacles, assuming optimal motion and perception distances. The heuristics using the visibility map replace the perception and motion costs by better estimates, but still underestimating the real cost as shown in Figures 4.2 and 4.4 and Theorem 4.1. Thus the estimates $h^1(n, T)$ and $h^2(n, T)$ are always greater or equal than $h_{pp}(n, T)$, because we proved the sensing distance to the critical point is the minimal perception distance. Moreover, if both $h^1(n, T)$ and $h^2(n, T)$ are admissible heuristics, the maximum operation keeps that property. Therefore, $h_{pp}(n, T) \leq h^{AVM}(n, T)$. Also, because $h^{AVM}(n, T)$ is admissible, it is also dominant over $h_{pp}(n, T)$. □

4.2.4 Using Critical Point Position to Further Improve Goal Stopping Condition

Like we did in equation 4.4, we can use the added information of the critical point location to update the cost of connecting nodes to T_e, filtering the clearly unfeasible positions. That allows not only to reduce the size of the priority queue that manages the PA* search, but also the number of ray casting operations. Nodes n from where perception is not feasible are updated such as $c(n, T_e) = \infty$.

Given a target T^l in an unreachable region, we consider the distance from the target to the critical points, $d^c_{li}(T^l)$. We know the robot footprint, a circle with radius R, generates the frontiers F^{li}. Therefore, the distance between frontier points and the critical point c^*_{li} is R. Using this information, we can determine an annulus sector from the target to the possible frontier points with a distance R around the critical point, and guarantee that any feasible position with line-of-sight to the target has to be in that annulus sector. So, determining the maximum possible angle range between target and frontier points allows us to filter the feasible points for perception.

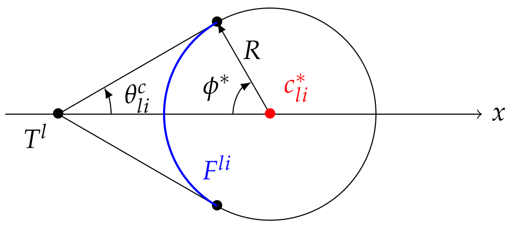


Figure 4.5: The maximum angle range from the target to the frontier points, considering the distance to the critical point and the robot size R.

Assuming a critical point c^*_{li} and target aligned with the x axis, as in Figure 4.5, the maximum angle to the frontier is given by:

$$\theta^f_{li}(T^l) = \max_{\phi} \operatorname{atan}\left(\frac{R\sin(\phi)}{d^c_{li}(T^l) - R\cos(\phi)}\right). \tag{4.8}$$

Because $d^c_{li}(T^l) > R$, we can solve the equation and find the optimal ϕ^*,

$$\cos(\phi^*) = \frac{R}{d^c_{li}(T^l)}, \tag{4.9}$$

and the maximum angle to the frontier θ^f becomes

$$\theta^f_{li}(T^l) = \operatorname{atan}\left(\frac{R}{\sqrt{\left(d^c_{li}(T^l)\right)^2 - R^2}}\right). \tag{4.10}$$

Then, we can filter the nodes n that are feasible in terms of perception of the target. For the filtering operation, we use the angle between n and the target T^l, θ^n, and the angle between the target and the critical points c^*_{li}, θ^c_{li}.

$$c(n, T_e) = \begin{cases} \lambda c_p(||n - T^l||) & ||n - T^l|| < d^c_l(T^l) \wedge \\ & \exists i : \theta^c_{li} - \theta^f_{li} \le \theta^n \le \theta^c_{li} + \theta^f_{li} \\ \infty & \text{otherwise} \end{cases} \tag{4.11}$$

4.3 Node Expansion Analysis for Variants of Perception Planning Heuristic

In this section, we present several experiments that show the benefits of each improvement proposed for the PA* heuristics. We only consider in this analysis the target points that are located in unreachable regions. Those are the only ones associated with critical points, thus being the regions where it is possible to use structured information from visibility maps to help the performance of PA* by improving its heuristics. For targets in the other regions, the algorithm uses just the base PA* heuristic, resulting in no negative impacts on efficiency. We consider 5 variants of PA*: base PA*, PA* with improved heuristic h_1 (PA*-1), PA* with h_1 and equation 4.4 for the stopping condition (PA*-1S), PA* with both improved heuristics h_1 and h_2 and equation 4.4 (PA*-2S), and finally PA* with both h_1 and h_2 and extended with equation 4.11 for the stopping condition (PA*-2SE). Table 4.1 explains the color meaning of the figures in this section. For the map representation, black represents obstacles, while white represents the free configuration space for the base PA* or the navigable space for the improved versions of PA*. Dark gray represents the unreachable space for base PA* or the non-visible parts of the unreachable space for improved versions of PA*, and light gray the visible parts of the unreachable space (for improved PA* versions only). Then, for the point representation, cyan represents the starting position, green the target location, and red the final path position for perception. Finally, for the PA* color representation, light blue represents the nodes on the open list, dark blue the expanded nodes on the closed list, purple represents the expanded nodes that are considered feasible locations for perception ($c(n, T_e)$ has a finite cost), and orange represents the nodes tested as goal position using ray casting.

Table 4.1: Color meaning for map, search and points.

		Base PA*	PA* variants
Map Colors	⬛	Obstacles	
	⬜	C^{free}	$Nav(S)$
	▩	Free Space $\setminus C^{free}$	$U(S) \setminus V(S)$
	▨	-	$V(S) \setminus Nav(S)$
Search Colors	🟦	Nodes on Open List	
	🟦	Expanded Nodes on Closed List	
	🟪	Expanded Nodes Feasible for Perception	
	🟧	Nodes Tested as Goal Position	
Point Colors	🟦	Starting Position	
	🟩	Target Position	
	🟥	Final Path Position for Perception	

Overall, the base PA* version has much worse performance in all criteria, because it does not use any additional information to guide search when targets are inside unreachable regions. As a result, and given the requirement of always finding optimal solutions, the base PA* variant might have to search the entire state space to guarantee the optimality of the solution. However, for the new variants introduced in this chapter, the heuristic has a better cost estimate, which results in search stopping earlier, and guaranteeing the optimality of the solution without having to explore the entire state space.

Moving into the analysis of our results, we first start evaluating the impact of the first heuristic improvement h_1, from equation 4.2. In this heuristic, the distance to the critical point d^c is used as an indication of what is the minimum perception distance to the target. If the distance to the critical point is greater than the optimal sensing distance from the base PA*, d_s^*, the heuristic can use this distance to make a more realistic estimate of the cost to perceive the goal. As expected, the impact of this heuristic improvement is higher when the difference to the base heuristic is more significant, i.e., when the λ parameter is higher (lower optimal sensing distance of PA*, d_s^*), and the distance to the critical point greater. In those cases, for targets in unreachable regions whose real minimum perception distance is large, the base PA* algorithm will reach the critical point but not consider it as the final position, and search will continue expanding nodes with lower f values, assuming it might be possible to perceive the target from a smaller and better distance. Then, only after having explored a large portion of the space, the algorithm will find the critical point to be the right perception position and test it with ray casting in the stopping condition. However, using the information from the visibility maps, i.e., the distance to the corresponding critical point (or minimum distance in case of multiple critical points from where the target can be perceived), the improved heuristics does not consider an unrealistic perception distance. Therefore, it converges much faster to the solution, because it knows shortly after reaching the critical point that no other nodes can have lower cost, significantly minimizing node expansion. Figures 4.6b to 4.8b, 4.11 and 4.12b show the great impact of this heuristic for small d_s^*, while figures 4.13b to 4.17b show the lesser impact of this heuristic for lower λ.

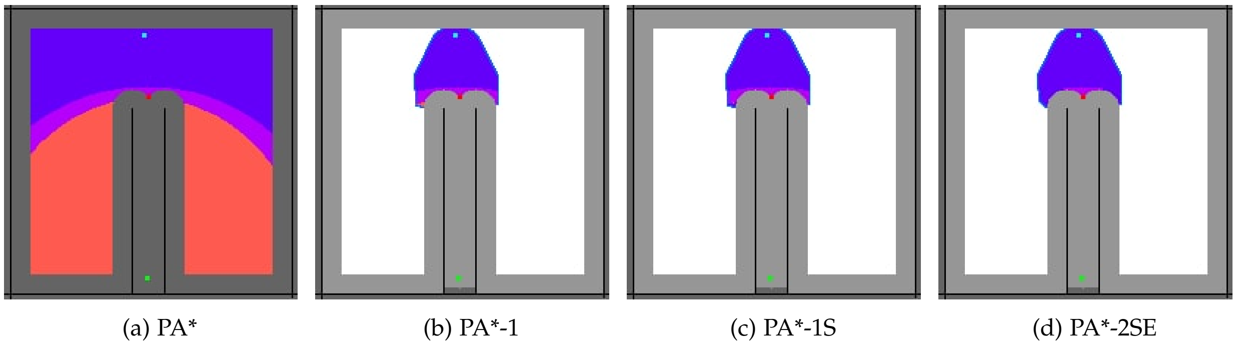

(a) PA* (b) PA*-1 (c) PA*-1S (d) PA*-2SE

Figure 4.6: S1: Node expansion and ray casting results for search on 200x200 grid map, $R = 13$, $r_p = 130$, $\lambda = 0.04$, quadratic perception cost function, $d_s^* = 12.5$, and target inside unreachable region with large distance to critical point.

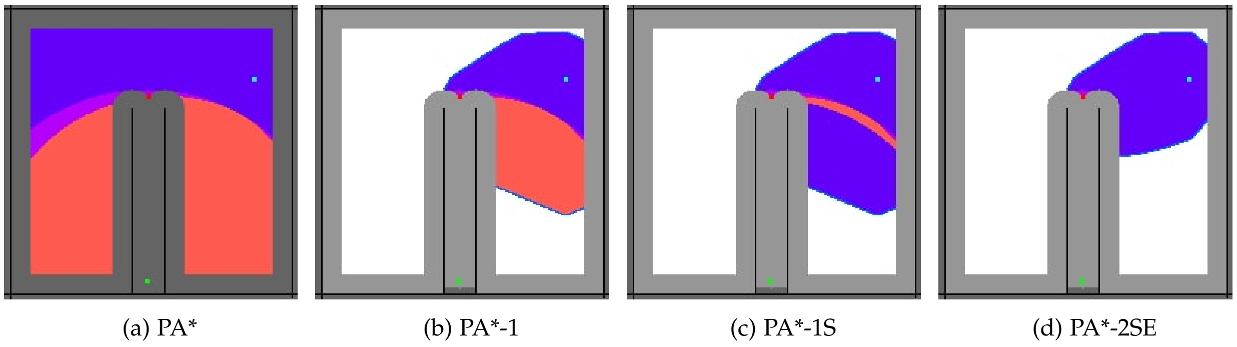

(a) PA* (b) PA*-1 (c) PA*-1S (d) PA*-2SE

Figure 4.7: S2: Node expansion and ray casting results for search on 200x200 grid map, $R = 13$, $r_p = 130$, $\lambda = 0.04$, quadratic perception cost function, $d_s^* = 12.5$, and target inside unreachable region with large distance to critical point.

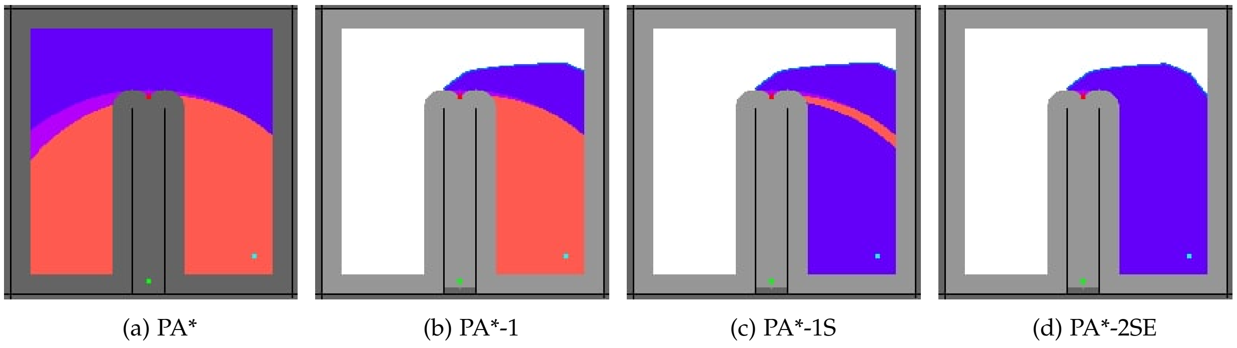


Figure 4.8: S3: Node expansion and ray casting results for search on 200x200 grid map, $R = 13$, $r_p = 130$, $\lambda = 0.04$, quadratic perception cost function, $d_s^* = 12.5$, and target inside unreachable region with large distance to critical point.

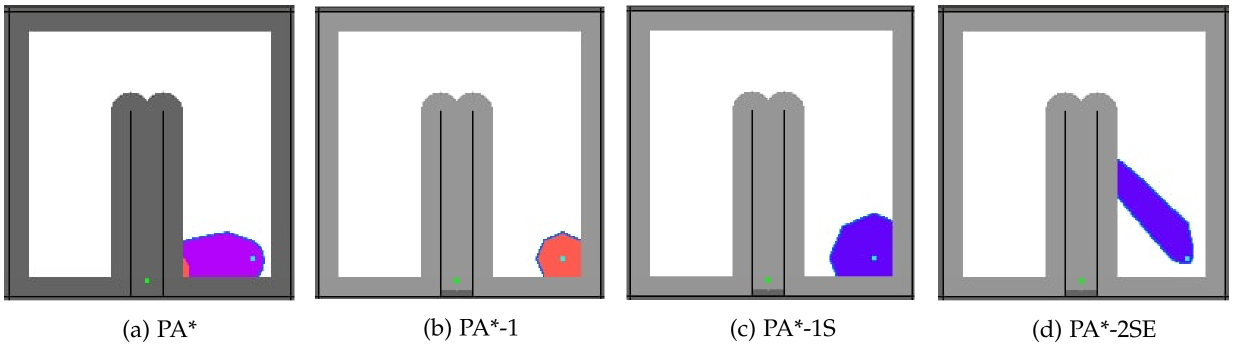


Figure 4.9: Initial stage of search for problem S3, figure 4.8.

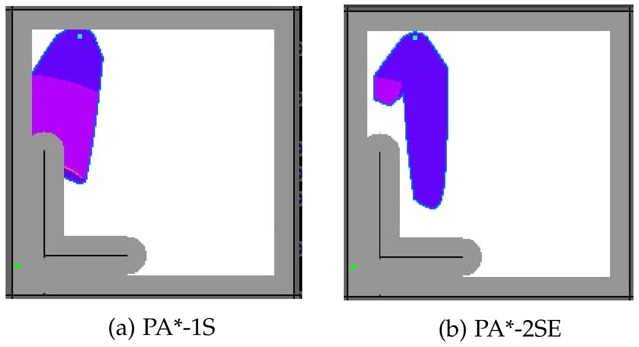


Figure 4.10: Progression of node expansion with multiple critical points.

Moreover, when using the improved heuristic h_1, it is also possible to use the distance to the critical point, d^c, as an additional improvement on efficiency by filtering the elements that the algorithm tests with the stopping condition (PA*-1R). Given the distance to the critical point being the minimum perception distance possible, we know that nodes with lower distances to the target cannot be the last position of the path, and obstacles are guaranteed to be in between, not allowing line-of-sight. In the base PA* algorithm, all expanded nodes are connected to the final node T_e if their distance is less than the maximum perception range. However, in our improved heuristic, using equation 4.4, only nodes with a distance to the target

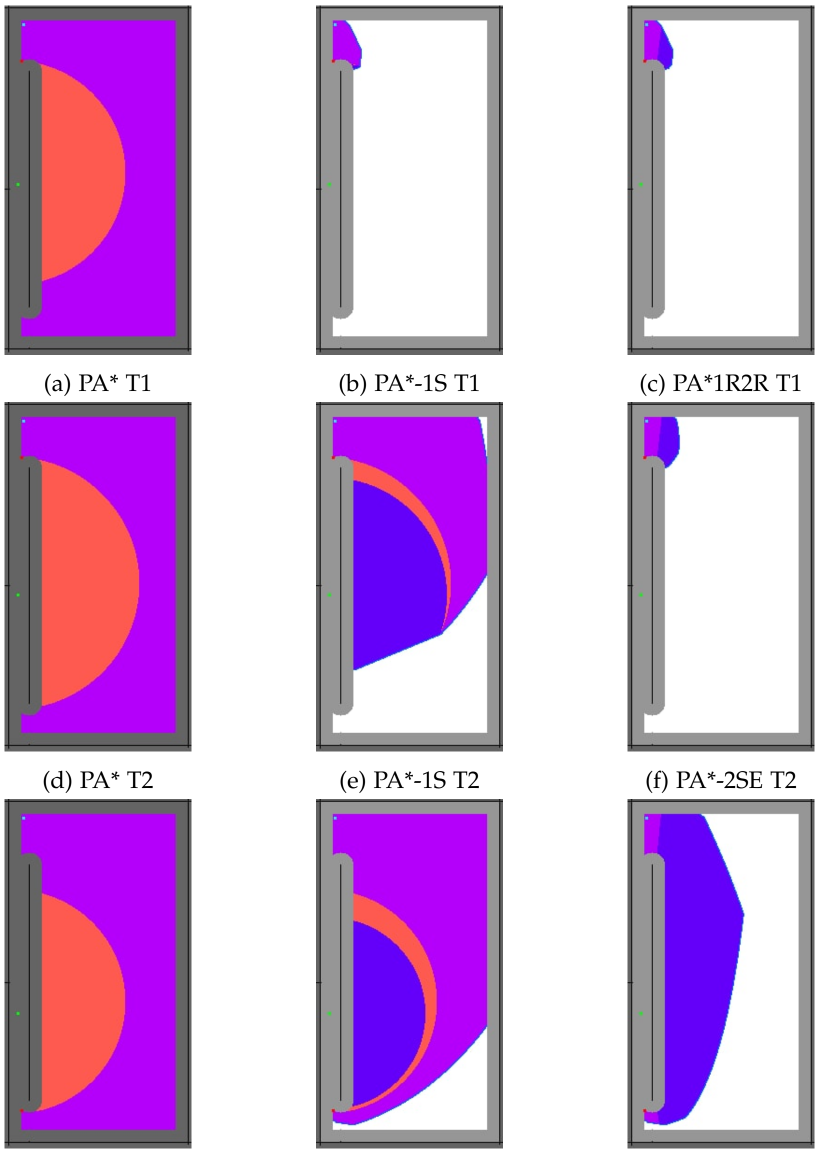

(a) PA* T1 (b) PA*-1S T1 (c) PA*1R2R T1
(d) PA* T2 (e) PA*-1S T2 (f) PA*-2SE T2
(g) PA* T3 (h) PA*-1S T3 (i) PA*-2SE T3

Figure 4.11: S4, S5 and S6: Node expansion and ray casting results for search on 375x200 grid map, $R = 13$, $r_p = 130$, $\lambda = 0.04$, quadratic perception cost function, $d_s^* = 12.5$, and target inside unreachable region with large distance to critical points, for 3 different target positions, T1, T2 and T3.

of at least d^c are connected to T_e, thus reducing the number of nodes tested with ray casting as possible final positions of the path. Reducing the number of times we check the stopping criteria is an important contribution, because ray casting is expensive to compute, and enables us to improve the search time. As seen in all figures, the cloud of points that expand during search remains the same as before, but this feature reduces considerably the number of nodes considered feasible, and as such it reduces the amount of ray casting operations to test for line-of-sight (in the figures, orange represents points tested with ray casting). While the h_1 heuristic has a small effect in scenarios with low λ, the PA*-1R variant has a good

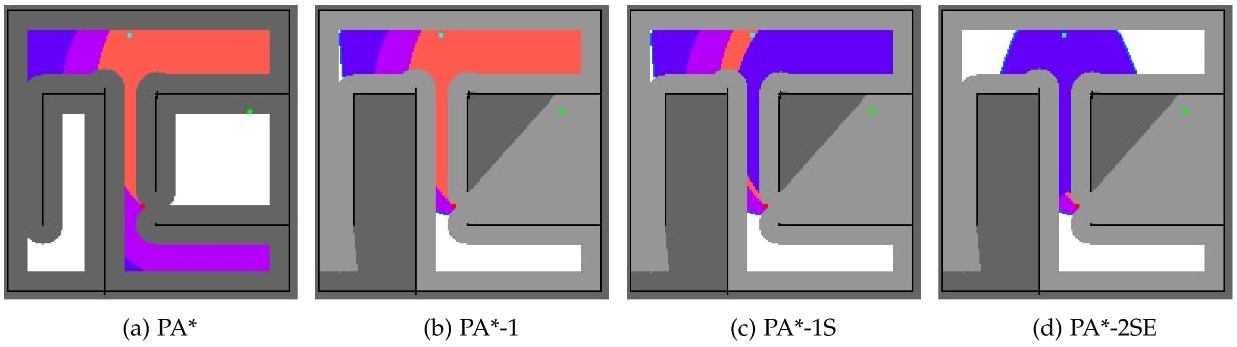

(a) PA* (b) PA*-1 (c) PA*-1S (d) PA*-2SE

Figure 4.12: S7: Node expansion and ray casting results for search on 200x200 grid map, $R = 13$, $r_p = 130$, $\lambda = 0.04$, quadratic perception cost function, $d_s^* = 12.5$, and target inside unreachable region with large distance to critical point.

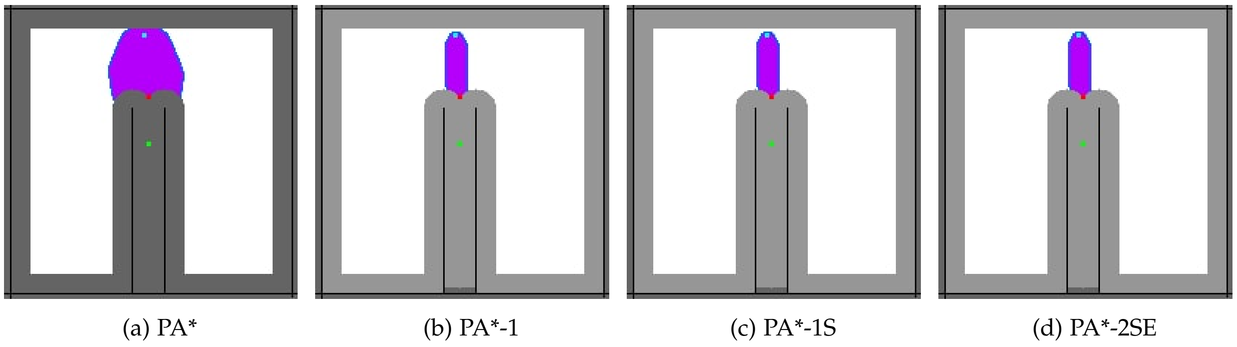

(a) PA* (b) PA*-1 (c) PA*-1S (d) PA*-2SE

Figure 4.13: S8: Node expansion and ray casting results for search on 200x200 grid map, $R = 13$, $r_p = 130$, $\lambda = 0.04$, quadratic perception cost function, $d_s^* = 12.5$, similar to S1, but target inside unreachable region with small distance to critical point.

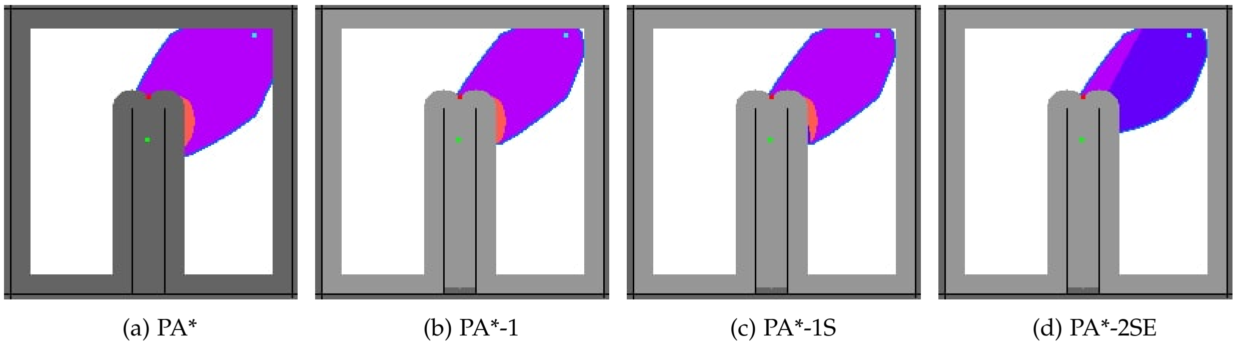
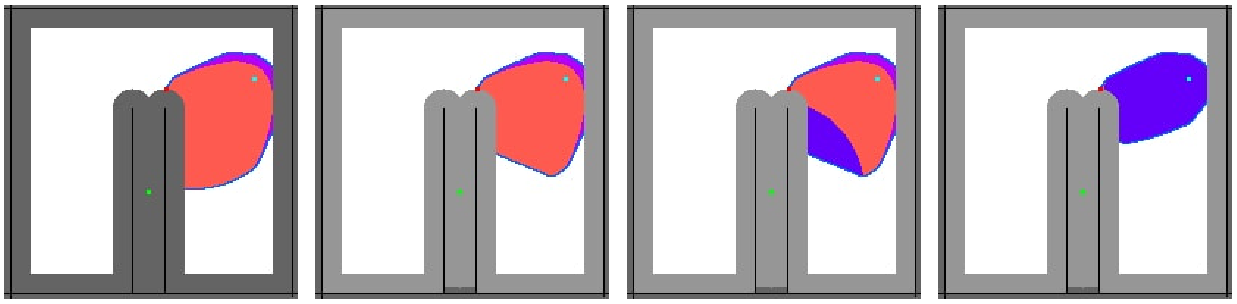

(a) PA* (b) PA*-1 (c) PA*-1S (d) PA*-2SE

Figure 4.14: S9: Node expansion and ray casting results for search on 200x200 grid map, $R = 13$, $r_p = 130$, $\lambda = 0.04$, quadratic perception cost function, $d_s^* = 12.5$, similar to S2, but target inside unreachable region with small distance to critical point.

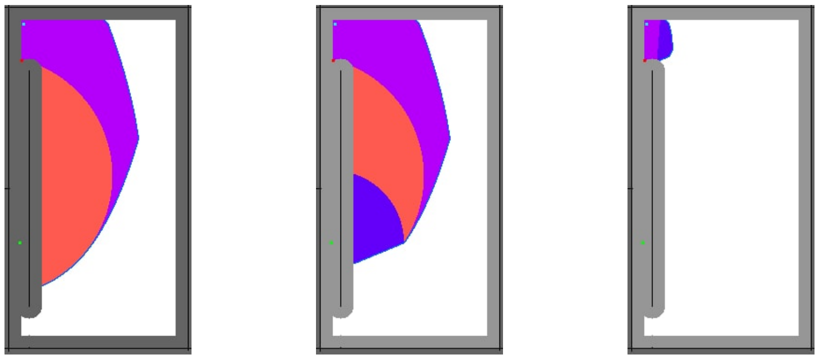
impact independently of λ, as confirmed in Figures 4.15c to 4.17c.

As shown in the previous section, h_1 only updates the perception distance estimate, being agnostic to the critical point positions. Therefore, many times h_1 makes the search expand nodes in undesirable directions, possibly contrary to the critical points, not directing search into the only regions from where the



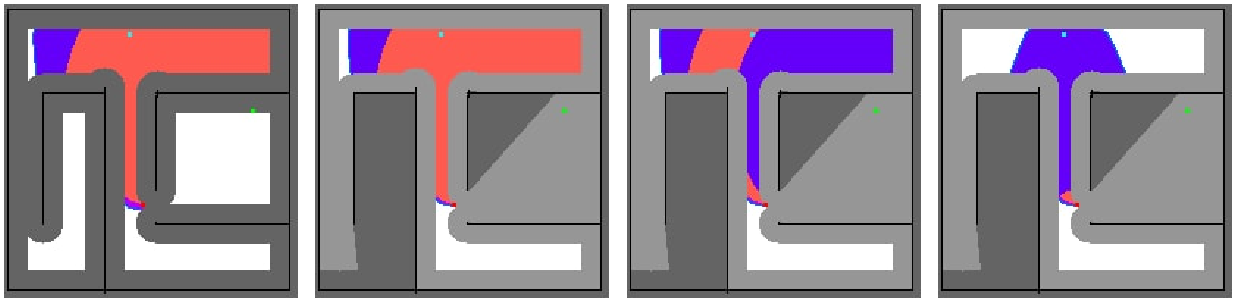
(a) PA* (b) PA*-1 (c) PA*-1S (d) PA*-2SE

Figure 4.15: S10: Node expansion and ray casting results for search on 200x200 grid map, $R = 13$, $r_p = 130$, $\lambda = 0.007$, quadratic perception cost function, $d_s^* = 71.4$, and target in unreachable region with $d^c(T) \approx d_s^*$.



(a) PA* (b) PA*-1S (c) PA*-2SE

Figure 4.16: S11: Node expansion and ray casting results for search on 375x200 grid map, $R = 13$, $r_p = 130$, $\lambda = 0.007$, quadratic perception cost function, $d_s^* = 71.4$, and target in unreachable region with $d^c(T) \approx d_s^*$.



(a) PA* (b) PA*-1 (c) PA*-1S (d) PA*-2SE

Figure 4.17: S12: Node expansion and ray casting results for search on 200x200 grid map, $R = 13$, $r_p = 130$, $\lambda = 0.007$, quadratic perception cost function, $d_s^* = 71.4$, and target inside unreachable region with $d^c(T) > d_s^*$.

target is observable. Again, the expansion with h_1 might still result in a waste of computation resources, even though it is better than the original heuristic. For that purpose, we contributed h_2, in equation 4.6, which directs the node expansion towards the points from where targets are observable, i.e., the critical points. As shown before, this new improvement can result in a great search efficiency boost. Besides being useful for expanding directly towards the best critical point if there are multiple ones (Figures 4.10 and 4.11), this heuristic is also helpful to direct the search towards the feasible regions for perception of the target, i.e., the critical point, even when there is only one. For the particular case where the target, initial position and critical point are aligned, as shown in Figure 4.6, the h_1 heuristic is dominant, and h_2 has minimal impact. However, as shown in many other figures, the improvements from h_2 can have a significant effect on search efficiency. Even though for Figure 4.8 the PA*-1S and PA*-2S variants have a similar cloud of expanded nodes at the end of search, an analysis of the initial expansion behavior, from Figure 4.9, is useful for understanding what differentiates each variant. While PA* starts expanding towards the target position, even though a wall blocks it, PA*-1 expands more uniformly, using the information that the target cannot be observed from a small distance. The PA*-1S variant expands exactly like PA*-1, but it does not use unnecessary ray casting operations, resulting in faster expansion. Finally, PA*-2S and PA*-2SE use the critical point location in the heuristic, and as such start expanding in the direction of the critical point. In Figure 4.12 example, the impact of h_2 is visible even at the final search step.

This new heuristic is also capable of being of great help for lower λ, where the d_s^* and d^c might be similar and where h_1 introduces less gain, as shown in Figures 4.16 and 4.17. Heuristic h_2 improvements are less dependent on λ, because the heuristic does not improve only the estimate for the perception distance, but also the motion cost estimate, using the critical points.

When there are multiple critical points, h_2 can produce vast differences. As shown in Figure 4.11, for the T1 target positions, the target, optimal and closest critical point and initial position are aligned, so h_1 and h_2 produce similar results. On the other hand, for T3, the farthest away critical point is optimal, and h_2 is better at directing search towards it. However, more interesting here is the case of T2, where the optimal critical point is the closest, while the minimum perception distance to the target is the distance to the farthest critical point. In that case, h_1 will expand nodes using in the distance between the target and the bottom critical point in the heuristic, resulting in a lot of unnecessary expansions while the search tries to find a solution with a smaller perception distance. On the other hand, h_2 uses the trade-off between perception and motion cost, and immediately stops the search at the top critical point, knowing it is not worth to expand more and thus significantly reducing the number of node expansions.

Finally, we consider the last variant, PA*-2SE, using the rule from equation 4.11. This case does not need much explanation, as the results speak for themselves. This variant considers the geometry of the environment and limits the feasible perception positions to the ones in front of a frontier, resulting in a considerable reduction of the nodes tested with ray casting (orange in the figures), with great benefits in terms of computation time. Before, PA*-1 and PA*-1S had the same node expansion cloud and were only different in the number of expanded nodes tested with ray casting. Now, PA*-2S and PA*-2SE have the same expansion cloud, and again they are only different in terms of the number of ray casting operations, also having a significant impact on scenarios with low λ and large optimal perception distance d_s^*.

Table 4.2 summarizes all the tested scenarios. From the base PA* version to the variant PA*-2SE, the number of expanded nodes (Exp.), stopping condition tests (SC) and ray casting operations should always decrease, as the variants introduce changes that always lead to incremental improvements in the heuristic estimation. As the number of expansions reduces, the search time is also expected to go down, but that is not guaranteed, because the heuristic calculation complexity increases from PA* to PA*-2SE. More specifically, the complexity of PA*-2S and PA*-2SE increases linearly with the number of critical points from where the target is visible. Therefore, for layouts with many critical points per target, the time spent on search may increase even though the node expansion always goes down. However, we expect the most common scenarios to be the ones with a lower number of critical points per target.

Table 4.2: Number of expanded nodes (Exp.), number of stopping condition tests (SC) and ray casting operations, and computation time for the twelve test scenarios and the five variants of PA*.

Test		Exp. (#)	SC (#)	Time (ms)	Test	
S1	PA*	33312	11795	63	PA*	S1
	PA*-1	1963	11	1	PA*-1	
	PA*-1S	1954	2	2	PA*-1S	
	PA*-2S	1947	2	3	PA*-2S	
	PA*-2SE	1946	1	3	PA*-2SE	
S2	PA*	33789	12272	72	PA*	S2
	PA*-1	11076	3694	24	PA*-1	
	PA*-1S	7765	383	8	PA*-1S	
	PA*-2S	5348	329	9	PA*-2S	
	PA*-2SE	5020	1	5	PA*-2SE	
S3	PA*	33813	12296	66	PA*	S3
	PA*-1	15604	6684	29	PA*-1	
	PA*-1S	9324	404	8	PA*-1S	
	PA*-2S	9110	404	10	PA*-2S	
	PA*-2SE	8707	1	9	PA*-2SE	
S4	PA*	65092	15349	159	PA*	S4
	PA*-1	1122	25	1	PA*-1	

Table 4.2: Number of expanded nodes (Exp.), number of stopping condition tests (SC) and ray casting operations, and computation time for the twelve test scenarios and the five variants of PA*.

Test		Exp. (#)	SC (#)	Time (ms)	Test	
	PA*-1S	1101	4	1	PA*-1S	
	PA*-2S	1096	4	1	PA*-2S	
	PA*-2SE	1093	1	2	PA*-2SE	
S5	PA*	70138	20395	150	PA*	S5
S5	PA*-1	51628	17425	105	PA*-1	S5
S5	PA*-1S	36571	2368	68	PA*-1S	S5
S5	PA*-2S	1652	39	2	PA*-2S	S5
S5	PA*-2SE	1614	1	2	PA*-2SE	S5
S6	PA*	65108	15365	114	PA*	S6
S6	PA*-1	58165	15365	106	PA*-1	S6
S6	PA*-1S	46809	4009	65	PA*-1S	S6
S6	PA*-2S	24147	2614	43	PA*-2S	S6
S6	PA*-2SE	21534	1	24	PA*-2SE	S6
S7	PA*	11822	4069	18	PA*	S7
S7	PA*-1	9741	4069	14	PA*-1	S7
S7	PA*-1S	6126	454	6	PA*-1S	S7
S7	PA*-2S	3620	454	6	PA*-2S	S7
S7	PA*-2SE	3185	19	4	PA*-2SE	S7
S8	PA*	1792	1	1	PA*	S8
S8	PA*-1	532	1	0.5	PA*-1	S8
S8	PA*-1S	532	1	0.5	PA*-1S	S8
S8	PA*-2S	532	1	0.5	PA*-2S	S8
S8	PA*-2SE	532	1	0.5	PA*-2SE	S8
S9	PA*	5143	168	6	PA*	S9
S9	PA*-1	3961	165	8	PA*-1	S9
S9	PA*-1S	3941	145	5	PA*-1S	S9
S9	PA*-2S	3778	123	6	PA*-2S	S9
S9	PA*-2SE	3656	1	5	PA*-2SE	S9

Table 4.2: Number of expanded nodes (Exp.), number of stopping condition tests (SC) and ray casting operations, and computation time for the twelve test scenarios and the five variants of PA*.

Test		Exp. (#)	SC (#)	Time (ms)	Test	
S10	PA*	9348	4503	21	PA*	S10
	PA*-1	8288	3973	18	PA*-1	
	PA*-1S	7366	3051	16	PA*-1S	
	PA*-2S	5310	2314	14	PA*-2S	
	PA*-2SE	3003	7	3	PA*-2SE	
S11	PA*	35430	12681	87	PA*	S11
	PA*-1	33638	11806	90	PA*-1	
	PA*-1S	30128	8296	80	PA*-1S	
	PA*-2S	1157	1	12	PA*-2S	
	PA*-2SE	1157	1	13	PA*-2SE	
S12	PA*	10271	4729	16	PA*	S12
	PA*-1	10159	4729	17	PA*-1	
	PA*-1S	6560	1130	8	PA*-1S	
	PA*-2S	3493	854	4	PA*-2S	
	PA*-2SE	2686	47	2	PA*-2SE	

Table 4.3 shows that the time spent building the visibility maps can be gained back with 10 to 15 searches. The visibility maps calculation can also be done once before-hand, as an offline pre-processing of the map, while PA* searches can use its information for faster real-time operation. Moreover, the algorithm to build the visibility map is highly parallelizable, so it is possible to reduce its computation times significantly.

4.4 Summary

Adding information about the structure of the environment can be used to improve the heuristics in PA*, resulting in a reduced search with less expanded nodes and ray casting operations. The critical points from the robot-dependent visibility map enabled the creation of better estimates of the motion and perception costs. We also proved they can be used in an admissible and dominant heuristic compared to the one proposed for PA*. We introduced four variants of the perception planning algorithm which incrementally add new features and improve the search performance: (1) minimum perception distance

Table 4.3: Computation time to construct the Visibility Maps of each twelve test scenarios from Figure 4.6 to Figure 4.17.

Test (#)	Time (s)
S1	0.90
S2	0.99
S3	0.95
S4	1.32
S5	1.32
S6	1.32
S7	0.76
S8	0.92
S9	0.94
S10	0.93
S11	1.33
S12	0.89

from critical points used in heuristic, (2) information from minimum perception distance used in goal stopping condition, (3) minimum motion cost from critical points used in heuristic, and (4) information from critical point position used in goal stopping condition.

Chapter 5

Using Perception Planning as Meeting Point Calculation for Delivery Services

We have shown how to calculate an optimal path for a perception task and how to optimize its heuristics. In this chapter, we show other applications that can use the PA* contribution. Here we will focus on a rendezvous algorithm to determine a meeting location for a user requesting a delivery from a vehicle in a city. As an example, we can think of the contemporary use case of food delivery.

While this looks like a simple motion planning problem with the actual user location as the target, we have all already been in situations where it is more efficient to walk towards the vehicle that comes for the delivery. Some of the reasons include: one-sided streets that force cars to drive long paths around the user; inexperienced drivers that have difficulty getting into the appropriate lanes for direct access to the user; roads that are only accessible to public transportation but not to private cars; vehicles that are too big to drive through the narrow streets of city centers; or even impatience from the user and their willingness to walk towards the car for a faster delivery.

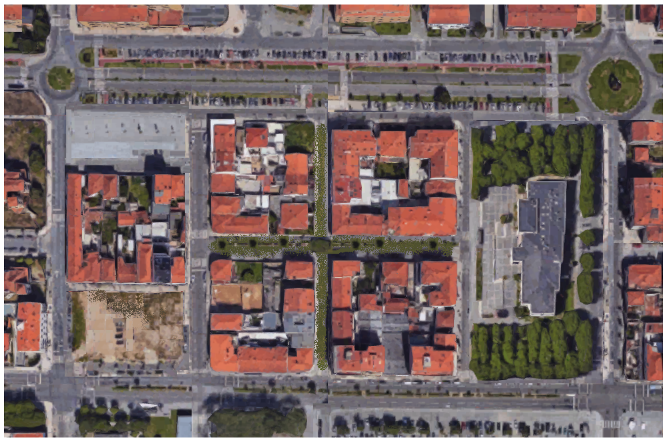
I wish there could be a way to take all this into account and be able to offer the user smart delivery locations to choose from. In the next sections of this chapter, we will show how to tackle some of these questions using our perception planning contribution.

5.1 Map Representation

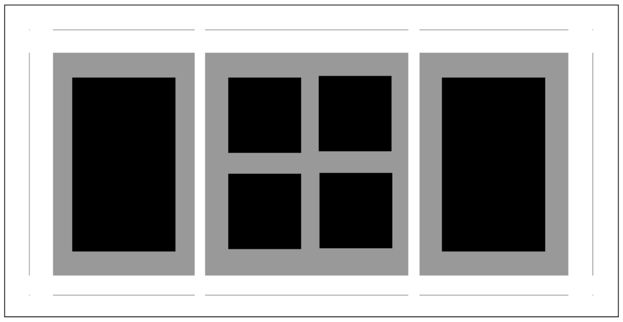
As before, we consider using a grid map to represent the planning environment. However, here we introduce a new feature to these maps. While before the maps were always binary - either free or occupied cells -, now we consider a third cell state which behaves differently for perception and navigation. This new cell type has the following properties:

- Blocks vehicle navigation
- Allows perception, being transparent to ray casting

In other words, this new cell type in a trinary map can differentiate the perception and navigation in terms of map representation, as we show in Figure 5.1. With our representation assumption, the navigation obstacle set is always a superset of the perception obstacle set.



(a) City example



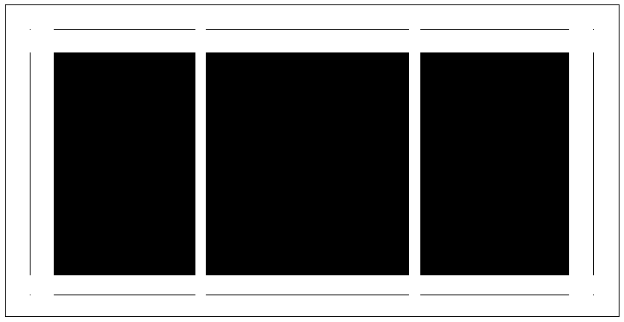
(b) Trinary Map

Figure 5.1: a) An example of a city road map, used as the basis for the experiments in a simulation environment in this chapter; b) a trinary map, as a simplistic conversion from the above city map example, with free cells in white, occupied cells in black (both navigation and perception obstacles), and special cells in gray for cells that behave as obstacles for navigation but are transparent for perception (e.g., sidewalks).

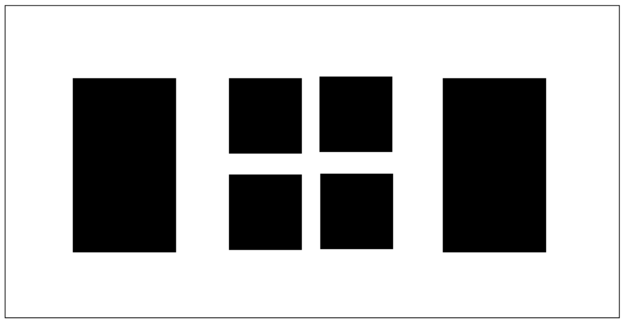
As an example, we can think of sidewalks on a road map. While they block navigation, they do no affect in terms of perception, as they still allow cars and people on the sidewalk to see each other. Therefore, in this chapter we will consider three states for grid cells:

- **Free**: allows both motion and perception (e.g., roads)
- **Special**: feasible for perception, but not for vehicle navigability (e.g., sidewalks)
- **Blocked**: occupied cells for both navigation and perception (e.g., buildings)

For all purposes, it is equivalent to having two different maps representing the world, one that codes obstacle navigability, and the other codes transparency for perception. We show an example of converting the trinary map to two binary maps in Figure 5.2.



(a) Navigation Map



(b) Perception Map

Figure 5.2: The trinary map coding both navigation and perception feasibility can be converted in two binary grid maps: a) free-space in white and navigation obstacles in black, b) free-space in white and perception obstacles in black.

5.2 Problem Definition as Perception Planning

The determination of a rendezvous point for the vehicle to deliver something to a user seems a good fit for our proposed perception planning, where we determined an optimal path for a robot to perceive a target location T. Here, we have U, the initial user position, instead of T. Moreover, there is no perception action. Instead, the perception cost will represent the cost of a user moving to the rendezvous point. The overall cost of path ρ is

$$\text{cost}(\rho) = \text{cost}_v(\rho) + \lambda \text{cost}_u(\rho, U), \tag{5.1}$$

where cost_v is a cost dependent on the vehicle path, cost_u is the cost dependent on the user walking distance, λ is a trade-off parameter, and U is the initial user position.

One possible approach would be for the cost functions to convert distance into time. In that case, assuming the goal would be to minimize the time to the rendezvous, the *max* operation should be used instead of a sum between $\text{cost}_v(\rho)$ and $\lambda\text{cost}_u(\rho, U)$, thus optimizing paths for the faster rendezvous and delivery. However, different users might have a different willingness to walk towards the vehicle and might prefer to wait longer instead of meeting the vehicle in a middle point. So, we consider the *max* operation not appropriate for this scenario, and we explain now what each cost element represents in our proposed sum for the overall cost determination.

5.2.1 Vehicle Motion Cost

As for the path of the vehicle, we will still consider its cost to represent a time calculation. While planning in a grid, the overall cost of a motion path is going to be the sum of all the individual cell connection costs for the cells that belong to the final path. The cell connection cost can then be dependent on the maximum speed for a specific vehicle traversing it, corresponding both to the vehicle and road intrinsic characteristics. There could also be considerations on traffic based upon past and current data from vehicles, for a better estimate of speed, but we leave those considerations for future work and assume the speed to be constant with time.

There could also be other cost considerations, such as tolls, but here we also assume those are not relevant in terms of rendezvous determination in a city scenario. The perception planning for user deliveries deals mainly with the motion planning for the last part of the trip, where vehicles navigate in a city to get to locations where they can deliver something to the users, such as a food delivery service, a scenario where it is quite uncommon to have tolls.

A key difference to our previous approach is the necessity to consider directionality in terms of navigation, something that we did consider before on the robot use cases. For a robot navigating indoor, apart from obstacle avoidance, robots can generally move in any direction from any position. However, in the vehicle use case that navigates in a city, it is essential to consider, for example, one-sided streets. Moreover, even normal roads have specific lanes in each direction. Therefore, to consider direction, we introduce a companion map to the original city map that represents for each cell the feasible navigation directions. For simplicity, we consider only four basic navigation possibilities:

- **Vertical Up**
- **Vertical Down**
- **Horizontal Left**

- **Horizontal Right**

On the other hand, in grid maps, cells have 8-connectivity, connecting to all its eight neighbors with eight different directions, as shown in Figure 5.3. We represent each of these connection directions with an index that goes from 0 to 7.

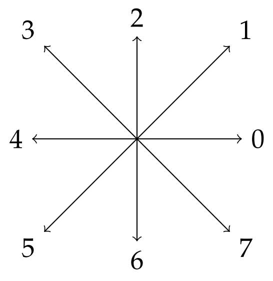


Figure 5.3: Neighbor connections and direction indexes for a cell in a grid map.

Going back to the city map, if we consider for now only a road or lane with the *Vertical Up* navigation possibility, we know that a vehicle navigating in that road can move up, but also go left and right to shift positions inside that lane. Therefore, in cells that lie in roads with *Vertical Up* navigability, the only feasible connections between grid neighbors would be the ones that represent motion to the left (connection index 4), up (index 2) and right (index 0), with all the possibilities in between (indexes 1 and 3). Extrapolating to the other basic navigation directions in the city map, we have the following feasible connections for each basic navigation direction:

- **Vertical Up**: neighbor indexes 0 to 4
- **Vertical Down**: neighbor indexes 4 to 0
- **Horizontal Left**: neighbor indexes 2 to 6
- **Horizontal Right**: neighbor indexes 6 to 2

Moreover, if we want to deal with road intersections and roundabouts, we also need to consider other compound navigation directions, which are intersections of the other four basic directions:

- **Up Left**: neighbor indexes 2 to 4
- **Up Right**: neighbor indexes 0 to 2
- **Down Left**: neighbor indexes 4 to 6
- **Down Right**: neighbor indexes 6 to 0

As an example, we represent a roundabout with four quadrants, sequentially going from *Up Left* to *Down Left*, then to *Down Right* and finally to *Up Right*. For the simulation environment used in this chapter, we present in Figure 5.4 a representation of the navigation directions for each region.

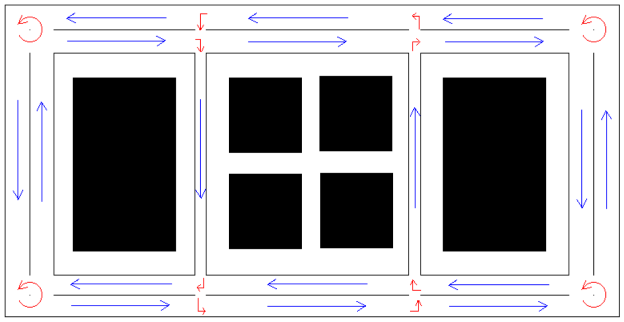


Figure 5.4: A demonstration of the direction map with basic directions in blue and compound directions in red in the roundabouts and intersections.

The delivery problem also fits nicely in the robot-dependent framework, with the possibility of pre-processing each map according to the footprint and size of each vehicle. The robot-dependent transformation allows us to render an adapted map to the footprint of each type of vehicle, thus enabling us to consider both the size of streets and cars in the planning algorithm, which usually does not take place in state-of-the-art planning algorithms for deliveries in cities. While it could be slow for a car to go through narrow streets in some city centers, it would not be obvious to eliminate that option if the vehicle in question is a motorcycle.

For simplicity, for the remaining of this chapter, we run our experiments in examples only with differently sized circular footprints.

5.2.2 User "Perception" Cost

As for the user cost, cost_u, it is a function of the minimum walking distance to the vehicle path, and, as we saw in chapter 3, it is also the walking distance from U, the initial user position, to the final rendezvous delivery location.

The drawback of solving this problem with perception planning is that there always has to be a non-occluded line-of-sight between the user and the rendezvous positions. An underlying assumption is that users can only walk on sidewalks without ever going around buildings that occlude the necessary line-of-sight. That assumption can be reasonable if, for example, we consider users would only be willing to walk towards a delivery location when they already see the vehicle.

In terms of the perception function c_p from chapter 3, here we have a similar function c_u, used to consider different models for the cost of walking to a rendezvous location, with the only constraint that, again, it is a monotonically increasing function with distance in order to guarantee optimal solutions with our proposed algorithm. A linear model is a reasonable assumption, but we can also imagine that for some users the increase in cost is stronger for shorter distances (showing some inertia and reluctance to start walking). For other users, it might be easily accepted to walk a small distance, but the increase in cost may be higher for large distances.

5.2.3 Trade-off Parameter λ

We are left with the trade-off parameter λ. As it is clear from its name, this parameter trades off the cost of the vehicle motion with the cost of a user walking to a delivery location. By its nature, it is a difficult parameter to assess, as it highly depends on user preferences. Regarding the dependence to the walking distance for the user, that is modeled with the perception function c_u, as mentioned in the previous section. Furthermore, it is quite expected to change with time even for the same user, according to the constraints of each day and how fast the user needs to get inside the vehicle. Weather is also a strong example of a variable that causes variability in the trade-off parameter, as someone might be more than willing to walk on a sunny day, but less happy to do so when it is heavily raining.

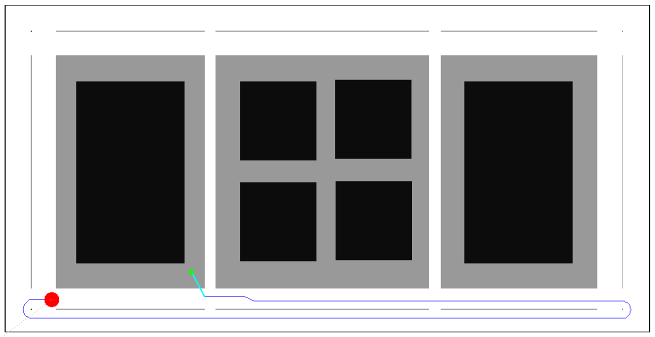
We propose, as a solution to that uncertainty, to plan paths with different values of the trade-off parameter and offer various possibilities to the users. By presenting alternatives to the user, it would be possible to create a better service by providing the user with the option of choosing according to their preference, while collecting data at the same time that can be used to learn those user preferences. However, we don't implement any learning in this chapter.

It is important to note that while the λ parameter can take any value, its variation generates only some smaller variations in the rendezvous location. By having the user choose one option does not translate directly to values of λ, but instead it translates to some windows of λ variation.

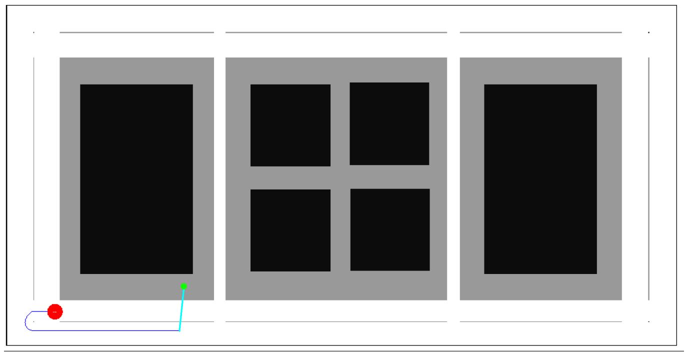
5.3 Experiments on City Motion Planning with PA*

Here we show some examples of running our perception planning algorithm in a city map for deliveries to a user in an initial position. In Figure 5.5, the starting vehicle position is very close to the user, but due to the navigation directionality of each lane, it needs to go around a long path to invert its direction and meet the user close to his initial position.

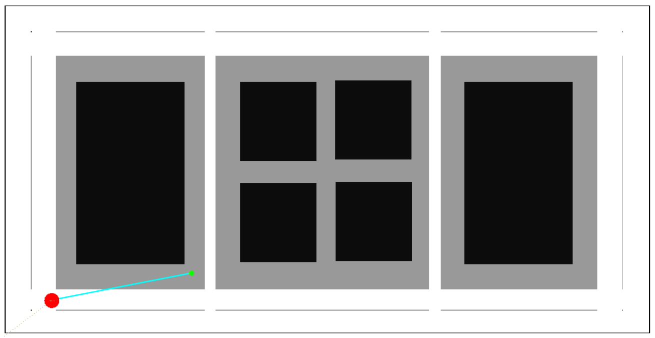
Figure 5.5a presents a solution of perception planning where the trade-off parameter has a high value, i.e., the user has a high cost of motion, so the vehicle travels a long path to get to the user in a position very close to the user's original location when requesting the delivery. On the other hand, in situations



(a) High λ



(b) Mid λ

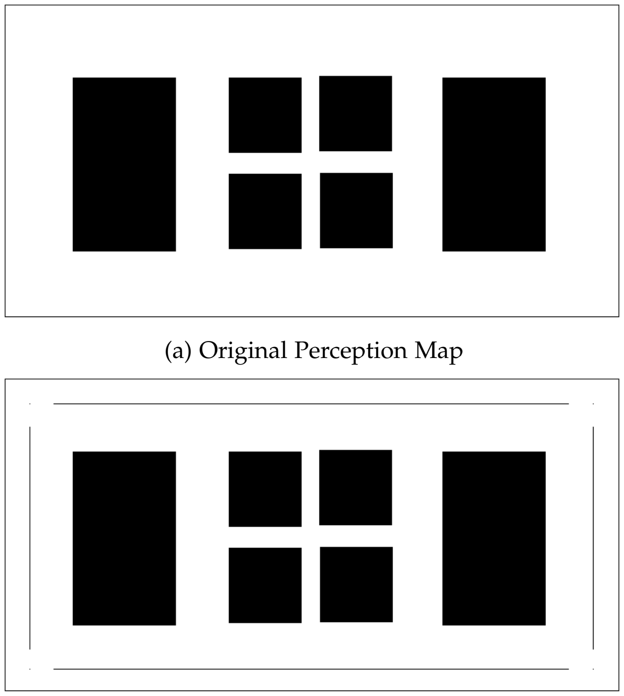


(c) Low λ

Figure 5.5: Three different delivery locations, dependent on parameter λ; initial vehicle position in red, user position represented with a green dot, vehicle path in blue, and walking path in cyan.

with heavy traffic, the vehicle might have a higher motion cost than the user, which is equivalent to having a lower λ. In Figure 5.5b, the vehicle goes around a roundabout to invert direction and approach the user, but it is still more cost-effective for the user to cross the street and get to the vehicle himself. Traveling to the other roundabout to switch lanes again and get even closer to the user would be too expensive in terms of time, compared to the willingness of the user to walk a short distance. In Figure 5.5c, the λ parameter is so low that the most cost-effective solution is for the vehicle to remain in the same position, and let the user walk to it.

As we have shown in the previous example, it is possible to come up with solutions where the user might cross a street to get to the vehicle for the rendezvous. In Figure 5.6, we present a slightly modified simulated city map, where neither the vehicle nor the user can cross the four bigger roads around the buildings. Previously, the middle of the road was already considered an obstacle in terms of navigation to separate lanes with different directions, as illustrated in Figure 5.2a, but in the perception map, those lines were not obstacles, meaning the user could cross those roads when moving towards the rendezvous position.



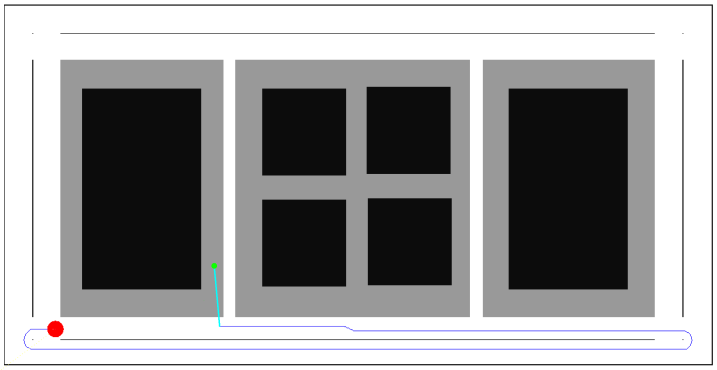
(a) Original Perception Map

(b) Modified Perception Map

Figure 5.6: Modified perception map where the middle of some road, separating lanes with different directions, is considered an obstacle in terms of perception map, as we already had for navigation in Figure 5.2a; as a result, in the new scenario it is impossible for users to cross any of the four bigger roads around the buildings.

However, with the modified perception map from Figure 5.6b, the lane separation becomes also an obstacle for "perception", and as a result, the user cannot cross those roads any more in the new scenario.

In this particular scenario, the middle of the road has a line of occupied cells separating the two road navigation directions, and such line functions as an obstacle both in the navigation and the perception map. In this new scenario, with similar initial positions for the vehicle and user, there is never a solution for which the user has to cross a road. We present again two planning solutions that are dependent on the parameter λ. In Figure 5.7a, with a high λ, the solution is the same as for the previous scenario, in Figure 5.5a. However, for a lower λ, there is a new solution where there is less driving by the vehicle and more walking by the user compared to Figure 5.7a. This solution arises from the additional constraint that the user cannot cross the streets. Otherwise, there would be other solutions (e.g., Figure 5.5b) with a lower overall cost for the same configuration and trade-off parameter.


(a) High λ

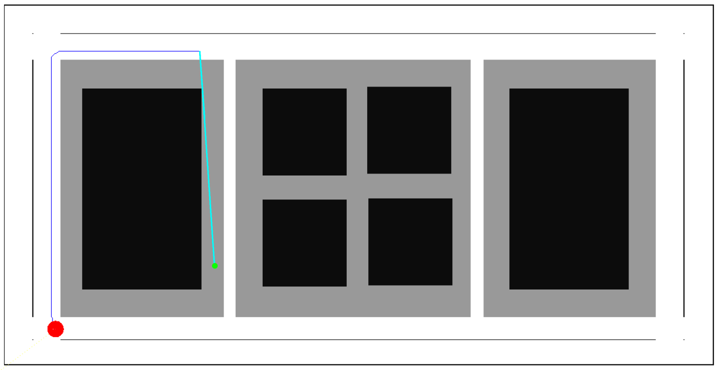

(b) Low λ

Figure 5.7: Planning solutions for two different λ on the modified map of Figure 5.6

In the experiments of Figures 5.5 and 5.7, the vehicle size did not allow it to travel through the narrower vertical streets, thus explaining the results in Figure 5.7, where the vehicle never gets genuinely close to the initial user position. That represents the inability of some vehicles to drive through some streets. In Figure 5.8 we run a planning instance with a smaller vehicle that can now drive in the narrower streets. As a result, for high λ, there is now a new solution through a different and shorter path that gets very close to the original user location for the delivery request.

5.4 Determining Vehicle-Dependent Visibility Maps

In a previous chapter, we used the information from visibility maps to generate improved heuristics for perception planning. Those improved heuristics were based on *critical points*, as the knowledge of their position and their distance to target points provided valuable information that could be used to approximate the heuristic value to the real cost, still with guarantees of optimality.

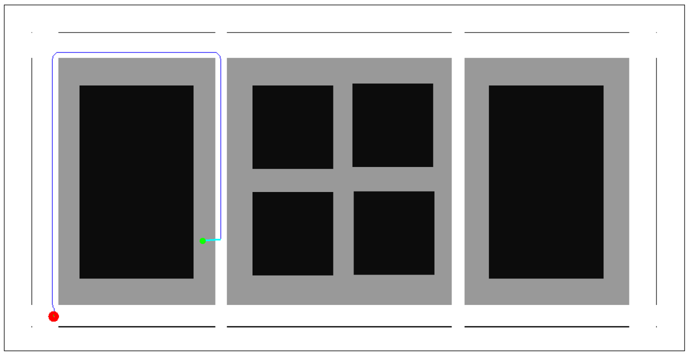


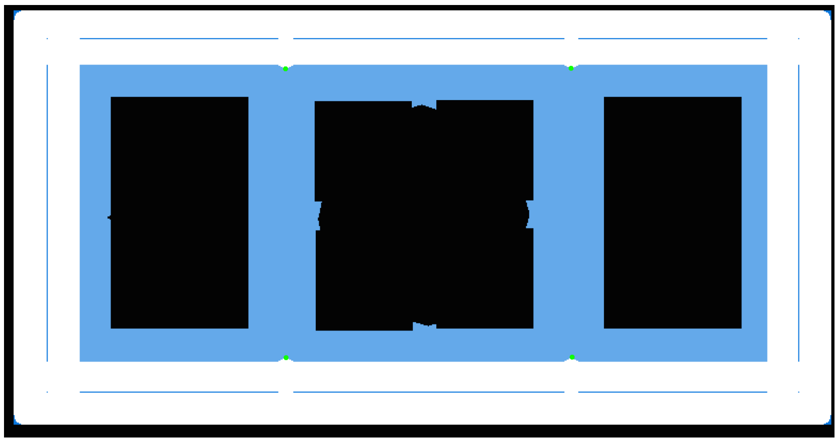
Figure 5.8: Sample of a planning instance with a smaller vehicle that can drive in all the streets of the original map.

However, the algorithm generated *critical points* on maps where the assumption was that all obstacles were always both navigation and perception obstacles. That assumption is not valid in the case of city maps, where the borders of navigation are very different and generally not coincidental with perception obstacles.

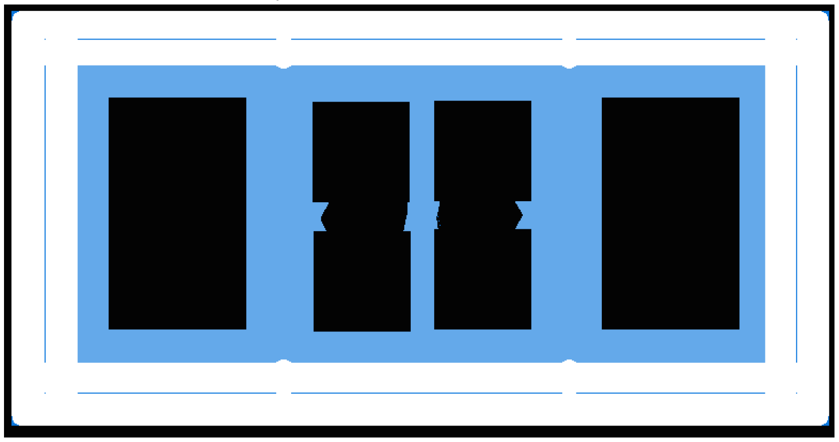
While we could still run the same algorithm based on the obstacle maps for navigation, the approximate visibility generated could be in some instances very different from the real visibility. We illustrated those differences in Figure 5.9, where first we generate visibility from the four critical points determined with the robot-dependent map. Besides showing the difference to the true visibility, we can also see it is difficult to get valuable information regarding the minimum perception distance with only four critical points. Moreover, there are no more guarantees that the distance from critical points to targets in perception-only regions is minimal. This fact makes it still possible to use critical points in the improved heuristics, but with the certainty that they will generate sub-optimal solutions, thus having little gain in using any of our contributed variants besides the original PA*.

To go around this problem, we sampled the reachable space to generate additional critical points, besides the ones determined through the methodology presented in chapter 2. We implemented a uniform generation by going through all the map cells that are part of the reachable space, and creating a critical point at that position if there is no other at less than a certain distance. We then determine visibility from each of these new critical points.

The disadvantage of this approach is that it becomes a lot more time consuming to determine the visibility map, compared to the original approach, going from less of a second to the tens of seconds for the map used in this chapter. We show in Figure 5.10 the result of sampling the reachable space to determine additional critical points for an accurate estimate of visibility.



(a) Visibility Generated from Four Critical Points



(b) True Visibility

Figure 5.9: For the trinary maps of city motion planning: a) visibility generated from four critical points; b) real visibility; regions that are visible but not navigable in blue, critical points for the robot-dependent map represented in the top image with four green dots.

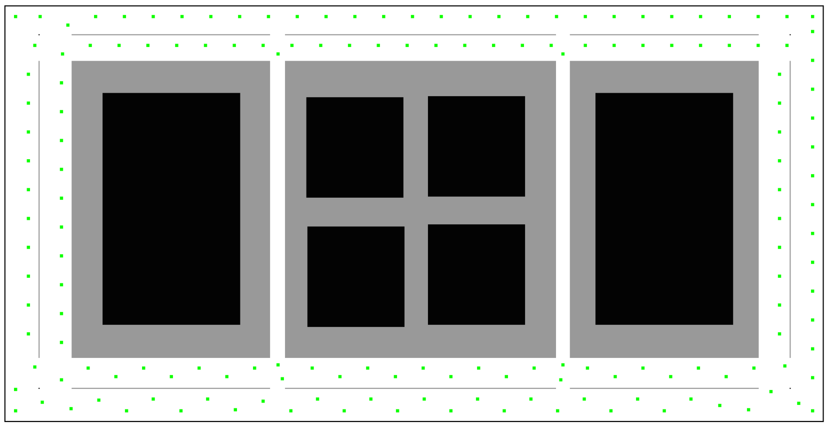


Figure 5.10: Extended critical points generated from an additional sampling of reachable space.

5.5 Results of Improved Heuristics of Perception Planning

In this section, we run a set of planning instances in the map presented previously in order to compare the different versions of improved heuristics we can use for perception planning.

We use a linear function for the perception cost, and we test four different values of the λ parameter: 0.1, 1, 10, 100. We bias the experiment towards larger values of λ because we assume it is more probable for walking to have a higher cost that a vehicle moving an equal distance. For the lower value of λ, it represents settings with heavy traffic where it is more costly for vehicles to move than it is for users to walk to a rendezvous location. With the next value, vehicles and users have the same cost for moving the same distance, then $\lambda = 10$ represents a more traditional use case, where vehicles cover some distance with a lower cost than the user walking that same distance. Finally, the last value of λ represents users with little willingness to walk, and prefer to wait for the car to reach them, even if that means long paths, be it for weather reasons or any other personal preference.

As we have seen in the previous chapter, we expect more significant differences for larger λ. Those represent planning instances where the vehicle moves closer to the user, and as such, it is expected to have higher numbers of node expansion. For those cases, it is more valuable to have information on the minimum perception distance from critical points (or minimum walking distance in this specific use case), as it has the potential to end search faster without expanding all the reachable nodes while expecting a lower perception distance that is not truly possible.

As before, we consider a vehicle starting position in one corner of the environment, and then for each λ we test 20 different target locations for the user location, spread uniformly around the environment. However, we only consider feasible user positions the ones in the “sidewalk” regions. Having the same starting position and targets located all across the city map lets us test both short and longer planning instances.

Finally, we test two different vehicles, a smaller one that can move through any road, and a bigger one that can only move in the four bigger roads around buildings.

As before, we compare the base PA* with the four variations presented previously in Chapter 4: PA*1, PA*1S, PA*2S, and PA*2SE. We briefly summarize their differences using incremental features:

- **PA*1:** Improves heuristic with minimum walking distance
- **PA*1S:** Reduces ray casting operations based on minimum walking distance
- **PA*2S:** Improves heuristic further using critical points position
- **PA*2SE:** Reduces even more ray casting operations based on critical points position.

Table 5.1: Comparison of time spent on planning for the different improved heuristics of perception planning, for two differently sized vehicles, and running on four different values of λ; results presented as a proportion of the base variant PA*

	Bigger Vehicle				Smaller Vehicle			
	$\lambda = 0.1$	$\lambda = 1$	$\lambda = 10$	$\lambda = 100$	$\lambda = 0.1$	$\lambda = 1$	$\lambda = 10$	$\lambda = 100$
PA*	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
PA*1	1.2	1.0	0.5	0.4	1.0	1.1	0.6	0.3
PA*1S	1.0	0.9	0.5	0.4	0.9	1.0	0.5	0.3
PA*2S	1.1	2.5	2.1	1.7	1.0	2.5	2.0	1.4
PA*2SE	1.4	2.9	3.5	2.5	0.7	2.5	2.2	1.4

Table 5.1 presents the different time results. Overall, we notice the trend that versions PA*1 and PA*1S have similar results to the base PA* version for lower λ as expected, and for larger λ there is at least a two times speedup.

As for the PA*2S and PA*2SE variations, they are almost always outperformed by the other versions, even the base PA*. We should note that as we saw in the previous chapter, these versions still have lower node expansion and a smaller number of ray casting operations. However, for these variants, the heuristic calculation is more expensive, to such extent that even with much lower node expansion and ray casting operations, the time spent to find a solution was frequently above two times of what needed for the base PA* with the simpler heuristic.

This considerable increase in the time spend for the more complex heuristics was something we did not notice in all other experiments before, but it can be easily explained. As we discussed earlier in previous chapters, we assumed navigation obstacles would be the same as perception obstacles. As a result, visibility can be approximated with a very low number of critical points. Moreover, for every unreachable but visible target, there were as well very few critical points from where the target was visible. As a consequence, the more complex heuristics PA*2S and PA*2SE were not very difficult to calculate, as they only needed to consider the location of one or two critical points.

However, for the city scenario in this chapter, where navigation obstacles are not necessarily perception obstacles, we used an algorithm to generate additional critical points in a uniform distribution, in order to still estimate the visibility map appropriately. As a consequence, now for each unreachable but visible target, there are plenty of critical points from where the target can be visible. As the complexity of the heuristic calculation is proportional to the number of critical points that can see the target, it becomes evident that in this scenario the more complex heuristics become too expensive to compute, losing the advantage of a lower node expansion and fewer ray casting operations.

We can conclude that for the city scenario it is better to use only the variant that considers the minimum walking distance, but not the critical point positions.

5.6 Summary

In this chapter, intending to show a practical use case of PA*, we applied perception planning to a problem of motion planning for vehicles moving in cities and making deliveries to users that can move to meet the vehicle in calculated rendezvous positions.

We presented a new map representation that differentiates navigation and perception obstacles, and which allowed the calculation of the rendezvous point between the user and the vehicle with our PA* algorithm.

The application of our techniques to this contemporary problem shows how our contributions on robot-dependent maps and perception planning can enable new advances in the problem of meeting users to make deliveries in a city. While running searches with different values for the cost trade-off parameter, we offer the user a novel algorithm where he can choose a delivery location according to his preferences. We can also adapt to vehicles with different footprints, making narrower streets only navigable by some vehicles.

We showed how to extend our visibility determination algorithm to the city scenario, and we demonstrated which changes in the heuristic function can benefit from visibility maps and improve the running time of search in this specific use case.

Finally, this chapter shows that it is possible to extend the perception planning framework for different robotic problems. Another interesting example would be to apply this framework to a multi-robot problem where the algorithm needs to compute a rendezvous location for the robots to meet, assuming it is more important to have a detailed path computation for one robot moving around obstacles. In that case, the perception part could represent a second robot with a reduced set of obstacles, as the "perception" obstacles must be a subset of the navigation obstacles of the first robot.

Chapter 6

Heterogeneous Multi-Agent Planning Using Actuation Maps

In this chapter, we consider a multi-robot coverage problem, which consists of distributing actuation tasks among the set of robots. The problem is to plan and find a route for each robot so that all the targets are actuated by the robots' actuators while minimizing the execution time. Multiple vacuum cleaning robots are an example of this type of planning problem. We modeled the problem using the standard PDDL language [14], using a 2D grid of waypoints as a discrete representation of the map. The robots can move from one waypoint to another as long as they are grid neighbors, and they can also actuate other waypoints inside the robot's actuation range.

We base our approach on a multi-agent classical planning technique that uses a task-allocation phase before planning individually for each robot.

We introduced the notion of using robot-dependent maps as a pre-processing phase to speed-up task assignment [37]. To do task allocation, a heuristic is used to estimate the cost of each robot executing a task. With the pre-processing technique, the automated planner saves time by only computing the heuristic for feasible tasks. Moreover, we also proposed using the pre-processing step to provide a faster cost estimation procedure based on robot-dependent maps, which further improves the efficiency of goal assignment in the multi-agent planning problem [26].

6.1 Multi-Agent Classical Planning

Any robotic problem can use our approach as long as it has at least the following elements:

- A map of the environment;

- A set of potential tasks to be executed by an agent over the environment;
- A way to model that scenario into a PDDL domain and problem.

The set of potential tasks can vary depending on the problem to solve. In this work, we are focusing on the coverage problem, and as a result, it is enough for the robots to move through the environment. Some other alternative tasks would be looking for objects, opening doors or achieve some clients' orders through the environment.

The potential of our approach relies on the ability to extract information from the map related to the tasks. The aim is to transform that information into a set of estimation costs that can speed up the planning process, i.e., we have computed the cost as the distance to each of the waypoints on the coverage problem.

In order to transform this kind of problem into PDDL we have to model (1) a domain; (2) a problem; and compute (3) a set of estimated costs. The domain and problem are a lifted representation in predicate logic of the planning task.

Robots execute actions, and each action has a set of preconditions that represent literals that must be true in a state to execute the action and a set of effects which are literals that are expected to be added or removed from the state after the execution of the action. As we are working with multiple agents, we consider a set of m agents given to solve the given coverage problem.

As a baseline we use the classical multi-agent planning that allocates tasks for each robot, plans paths individually, and then removes conflicts from the combined path solution. For task allocation, the baseline method uses a relaxed plan to compute per task and robot an estimated cost of executing each task. Thus the time spent on planning depends highly on the task assignment efficiency.

Usually, estimated costs are computed to divide the goals among the agents before the planning process starts. In classical planning, the algorithm runs a relaxed plan to obtain a heuristic function.

Even though the classical planner used is domain-independent, our function to compute the estimated costs is domain-dependent and should be set up differently on each domain, though it would be very similar to this one in most related robotic domains.

Therefore, we contribute a pre-processing step with Actuation Maps to speed up the task assignment phase, determining the feasibility of each pair robot-task before-hand. The planner receives the estimated cost as input and saves time by avoiding the relaxed planning computation and directly assigning tasks to robots. The Actuation Maps are generated once before planning, determining which regions are feasible for actuation for each one of the robots, and also providing a cheap estimate of the cost of using each agent to execute that task.

Figure 6.1 illustrates the contributed architecture. It has been divided into four modules and receives as input the map of the environment, the general knowledge related to the task to solve and the features of the set of robots. The aim of each module is described as follows:

1. Actuation Maps module: it is in charge of generating the AMs for each given robot. It also extracts the map features that can potentially alleviate the planning process, e.g., path-planning features, and transforms them into a set of estimation costs and generates the planning problem in PDDL.

2. Multi-Agent Planning Task Generation module: once received as inputs the domain and the outputs from the prior module, the goal assignment process is launched. This module is in charge of dividing the goals among the agents following some goal-strategy. Then, a specific domain and problem is generated for each agent, which is known as factorization.

3. Multi-Agent Planning Algorithm module: this module runs the individual planning process and the merging phase.

4. Conflicts Solver module: if any interactions need to be solved, this module employs a plan-reuse-planner to fix them.

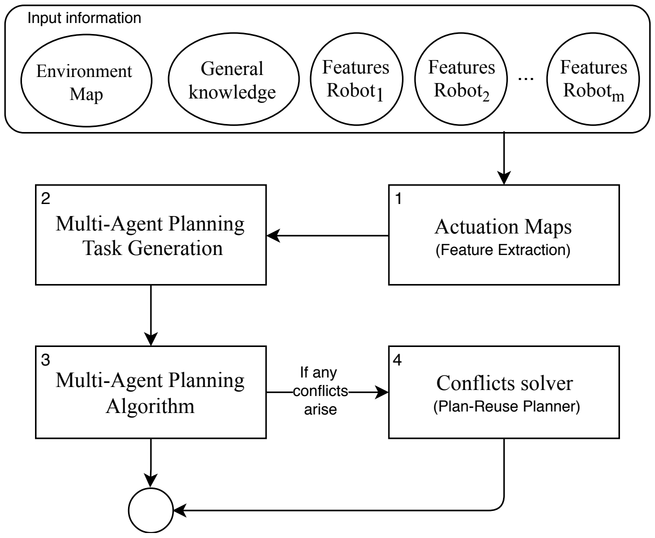


Figure 6.1: Complete architecture that combines Actuation Maps and Multi-Agent Planning [26].

6.2 Coverage Task Planning Formulation

In this work, we only consider heterogeneous teams of circular robots that actuate in a 2D environment, where the world is represented by a 2D image that can be downsampled to a 2D grid of waypoints. The actuation map gives information about the actuation capabilities of each robot, as a function of robot size and initial position. In the example with vacuum cleaning robots, the actuation map represents the regions of the world each robot can clean.

At first, we assume that robots are circular and the only feature is its size, with 2D grid positions being rotation-invariant. Other shapes can also be trivially considered in our approach by extending the PDDL domain file to take into consideration robot orientation as well.

As it was previously said, we modeled the domain and problem using PDDL[14]. The domain has two types of objects: *robots*, which act as agents; and *waypoints*, which represent positions in the discretized world. We consider a coverage problem, where the goal is to have the robots actuating on waypoints. In this version of the coverage problem, robots actuate a waypoint if it is inside its actuation radius. Thus, they do not need to be precisely placed on the waypoint to actuate it.

Therefore, the set G is a list of waypoints to actuate on (positions that need to be covered). The PDDL domain we created has four predicates:

- `At (robot, waypoint)`: defines the robot position;
- `Connected (robot, waypoint, waypoint)`: establishes the connectivity between waypoints, specified for each robot, and given the robot heterogeneity, some connections might be traversable by some robots and not by others;
- `Actuated (waypoint)`: indicates which waypoints were already actuated; this predicate is used to specify goals;
- `Actuable (robot, waypoint, waypoint)`: shows which waypoints can be actuated by a robot when located on a different waypoint location.

Every position in the world is a waypoint, and all of them need to be covered by at least one robot.

The waypoints, when connected, generate a navigation graph for a particular robot.

The domain defines two actions, and they are called `navigate` and `actuate`. The first one moves a robot from its current waypoint location to a neighbor waypoint as long as both are connected. The second action is used to mark a waypoint as actuated if it is identified as `actuable` from the robot's current waypoint location, i.e., the waypoint was located inside the robot's actuation radius on the real environment. `Navigate` and `actuate` are the two actions that can be executed by an agent when it is on specific a waypoint. Both `navigate` and `actuate` have as effect the predicate `actuated`.

6.3 Downsampling of Grid of Waypoints

For the planning problem, it is possible to consider each pixel as a waypoint. However, that approach results in a high density of points that would make the planning problem excessively complex. Moreover, there is some redundancy in having points that are too close to each other, as their difference is not significant in terms of the environment size and localization accuracy.

Therefore, we reduced the set of locations from all pixels to a smaller set of locations. We considered again waypoints distributed into a grid, but now the grid-size is greater than one pixel. Then, we can find the connectivity between points to construct the navigation graph of each robot, shown in Figure 6.2a. It is also possible to find which waypoints can be actuated from other waypoints using the distance between them, as shown in Figure 6.2b, by considering the maximum actuation radius.

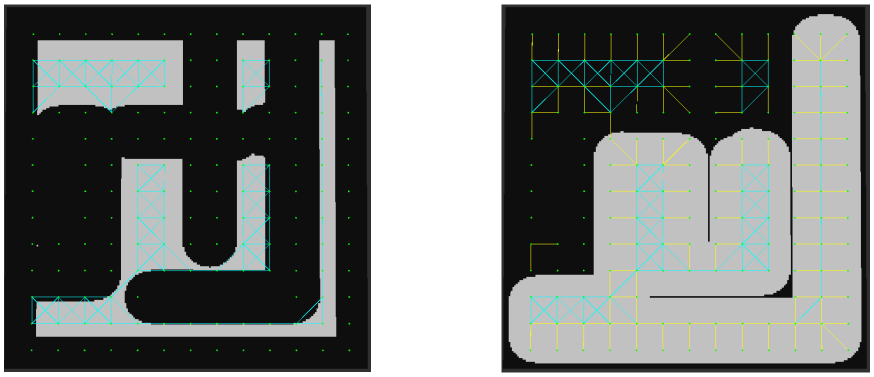

(a) Navigability (b) Actuation

Figure 6.2: a) Example of the free configuration space, with the discretization waypoints shown as green dots; blue lines represent the connectivity between waypoints in the navigation graph of the robot; using parameters δ and α it is possible to maintain the topology of the free configuration space by allowing points in the navigation graph that were initially unfeasible for the robot; b) actuation map of the same robot, and the respective actuation graph represented with yellow lines.

The problem of such discretization is the change in the actuation space topology. Adjusting the position of waypoints could allow a better representation of the topology of the environment, but the multi-robot nature of the problem compromises that solution. To deal with multiple robots with different reachable sets, for each agent, we independently adjust the waypoint position -temporarily- in a hidden manner invisible to the other agents. When discretizing each robot's configuration space, we might consider a waypoint as belonging to the free configuration space even if it is strictly outside it, as we assume an error margin to compensate for the discretization error. Nevertheless, we still maintain the original waypoint position in further steps, such as determining the actuation feasibility of that waypoint, and for visualization purposes as well. When determining the navigation graph of each robot, an unreachable waypoint position might be moved to the closest point in the configuration space, if the adjustment is under a given margin δ. As stated previously, the adjustment is always temporary to the construction of the connectivity graph of each robot. After testing the navigation connectivity, the waypoint position resets to its default grid position for the next steps, such as determining the actuation feasibility, and the navigation and actuation graphs of other robots.

Moreover, when determining the connectivity of waypoints for the navigation graph, only the eight grid neighbors are considered. A^* is then used to determine the real distance between waypoints (e.g., around obstacles), only considering connectivity if the actual distance is at most a factor of $\alpha = 1.2$ the straight line distance between them.

All waypoints that belong to the robot actuation map should be connected to some waypoint of its navigable graph. If that is not the case after the previous steps, we connect the isolated waypoints to the closest navigable vertex in line-of-sight, even if their distance is greater than the maximum actuation distance, again to compensate for the discretization error. Therefore, while the planner may return an actuate action to cover waypoint A from the navigable waypoint B in the discretized world, a real robot would have to move closer from waypoint B to waypoint A to actuate the latter.

The grid density is chosen manually to adjust the level of discretization. As for the α and δ parameters, they were tuned empirically such as the free space topology is still maintained even while using lower density discretization of the environment. By trial and error, we found empirically that $\alpha = 1.2$ works for all the tested scenarios. As for the δ parameter, we set it to always start with a value of three. The algorithm then builds the discretized model and verifies if it is valid, i.e., if all the waypoints belonging to the actuation map become feasible for the respective robot in terms of the discretized representation. If not, we increment the parameter until a topologically consistent representation is found (number of feasible goals equals the number of waypoints inside actuation map). Even though this fine-tuning methodology seems sensitive to the robot heterogeneity, the truth is that the final δ value depends on the size of the bigger robot, because the correct discretization of the configuration space is more sensitive to the δ parameter for bigger robots. Through experimentation, we found out that if a particular value of the δ parameter works well for the biggest robot, it always produces the correct discretization for smaller robots. Moreover, we also observed that δ = 4 pixels worked well for all the different and very diverse maps we tested in our experiments with circular robots, only failing for the any-shape experiments where the configuration space discretization is more sensitive to the possible robot orientation. For the any-shape robot experiments, we found that δ = 6 pixels were enough to obtain a proper discretization for all the environments tested. The consistency of the δ parameter over different environment maps shows that these parameters can be map-independent to a certain extent, with most of the work being easily automated.

6.4 Extracting Cost Information from Actuation Maps

When converting the original map and the Actuation Space to the PDDL description, it is possible to consider each pixel as a waypoint in a grid with the size of the whole image. But as discussed in the previous section, we reduce the set of possible locations by downsampling the grid of waypoints. The downsampling rate s_r is set manually. If the original pixel resolution is used, the resulting grid of waypoints G' contains all pixels and is equivalent to G. Otherwise, the set G' represents the grid waypoint positions after downsampling.

Using the Actuation Space it is possible to very easily find UG, the list of *unfeasible goals per agent r*:

$$UG = \{g \in G' \mid g \notin A_r(S_r) \quad \forall r\}. \tag{6.1}$$

The positions in the actuation space $A_r(S_r)$ are feasible goal positions for actuation tasks. Even though this was used to find the unfeasible list UG, the original Actuation Transformation does not provide any information about the cost for each robot to execute a feasible actuation task.

For that purpose, we contribute the following extension. We build the navigable space $Nav_r(S_r)$ in an iterative procedure, from the starting position S_r. In the first iteration we have $Nav_r^0(S_r) \leftarrow \{S_r\}$, and then the following rule applies:

$$\begin{aligned} Nav_r^j(S_r) = \{\mathbf{p} \in G \mid \exists \mathbf{q} \in Nav_r^{j-1}(S_r) : \mathbf{p} \text{ neighbor of } \mathbf{q} \\ \wedge \mathbf{p} \in C_m^{free} \wedge \mathbf{p} \notin Nav_r^a(S_r) \quad \forall a < j\}. \end{aligned} \tag{6.2}$$

When using this recursive rule to build the navigable space, we guarantee that any point in the set $Nav_r^j(S_r)$ is exactly at distance j from the initial position S_r.

Furthermore, if we build the actuation space sets with the intermediate navigable sets $Nav_r^j(S_r)$,

$$A_r^j(S_r) = Nav_r^j(S_r) \oplus R_m, \tag{6.3}$$

then the intermediate actuation set $A_r^j(S_r)$ represents the points that can be actuated by the robot from positions whose distance to S_r is j. The actuation space defined in the previous section can also be alternatively defined as

$$A_r(S_r) = \{\mathbf{p} \in G \mid \exists a : \mathbf{p} \in A_r^a(S_r)\}. \tag{6.4}$$

The estimated cost is defined for $g \in A_r(S_r)$:

$$EC_r(S_r, g) = \min\{j \mid g \in A_r^j(S_r)\} + 1. \tag{6.5}$$

The cost $EC_r(S_r, g)$ represents, for each $g \in A_r(S_r)$, the minimum number of actions needed for the robot to actuate the grid waypoint g if starting from the initial position S_r, measured in the pixel-based grid G. In Equation 6.5, the minimum j^* represents the minimum distance (i.e., the minimum number of *navigate* actions) needed to travel from S_r to some point from where g can be actuated. The added one in Equation 6.5 accounts for the one *actuate* action needed to actuate g, after the j^* *navigate* actions required to reach a place from where the robot can actuate g.

Considering the downsampling rate s_r, the cost has to be divided by s_r to transform the estimated cost of actions measured in the pixel-based grid G, $EC_r(S_r, g)$, to the respective cost value in the downsampled grid of waypoints G'. The *ceil* function rounds up the result of the division to the smallest integral value that is not less than $EC_r(S_r, g)/s_r$. The cost function is domain-dependent and works for the coverage problem. If a different problem is given as input, the cost function should be redefined.

$$\text{cost}(m, g) = \textit{ceil}\left(EC_r(S_r, g)/s_r\right) \tag{6.6}$$

6.5 Extending Approach to Any-Shape Robots

For the any-shape robots, a multi-layer representation is used to determine the actuation map, representing different orientations. However, in terms of accomplishing goals, we assume it is irrelevant the orientation from which a robot actuates on a waypoint position.

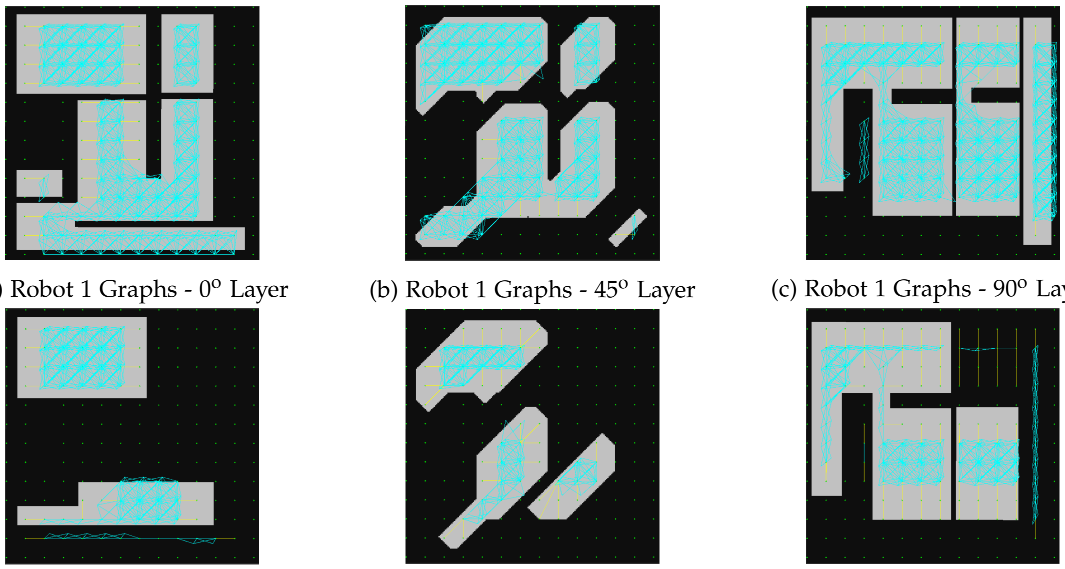
Therefore, while on the rotation-invariant scenario the domain was discretized in a series of 2D waypoints, for the any-shape case there are two types of waypoints: the 3D waypoints representing (x, y, θ) position, and the 2D waypoints representing (x, y) positions invariant to orientation.

The navigability graph now becomes a graph of connected 3D waypoints, modeling the motion capabilities of robots in the world in terms of both rotation and translation, individually or combined, as exemplified for different orientation layers on Figure 6.3.

On the other hand, the actuation graph is now a graph of 3D waypoints connected to 2D waypoints, representing the actuation of a rotation-independent position in the projected 2D actuation map, from a 3D robot waypoint location, also shown in Figure 6.3. The predicates on the PDDL problem are represented as follows:

- `Connected (robot, 3Dwaypoint, 3Dwaypoint)`
- `Actuable (robot, 3Dwaypoint, 2Dwaypoint)`

For each 2D waypoint in the circular robot scenario, there are now n_θ 3D waypoints in the same (x, y) position, representing the different orientations a robot can have on the same 2D waypoint. As we show in Figure 6.4, the two graphs are constructed independently of the initial position, allowing very easily to



(a) Robot 1 Graphs - 0º Layer (b) Robot 1 Graphs - 45º Layer (c) Robot 1 Graphs - 90º Layer

(d) Robot 2 Graphs - 0º Layer (e) Robot 2 Graphs - 45º Layer (f) Robot 2 Graphs - 90º Layer

Figure 6.3: The `connected` and `actuable` graphs shown in blue and yellow, respectively; as shown for each layer, the yellow actuation graph connects 3D waypoints to the original 2D green waypoints, and the blue connectivity graph connects 3D waypoints not only to neighbors in the same layer but also in adjacent layers.

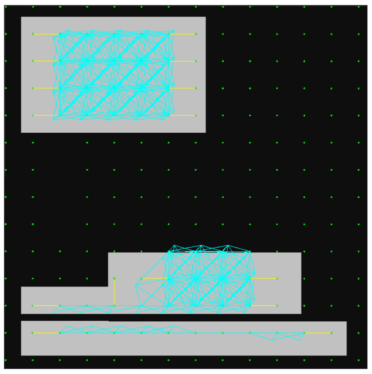
change the starting location of any robot and solve a different instance of the same problem. Thus, there were no modifications in the modeling of the PDDL problem. The 3D to 2D representation is transparent to the planning process.

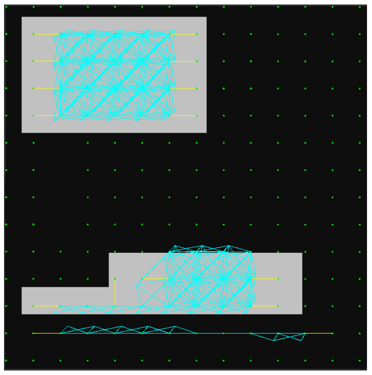
If we project the multiple layers of the graphs in a 2D image, we can analyze which waypoints are navigable in terms of the robot motion, and which ones are only feasible through an actuation action. As we show in Figure 6.5, some of the waypoints are not feasible by any of the robots, and all the feasible waypoints lie inside the Actuation Space (gray region of the images).

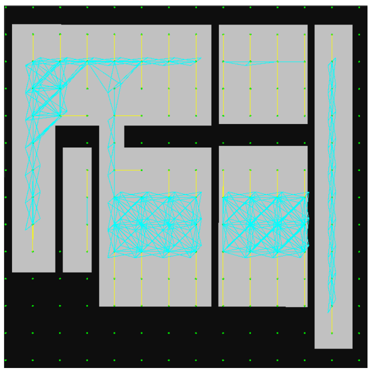
6.6 Experiments and Results

In this section, we show the results of the experiments that were designed to test the impact of the pre-processing on two different versions of our MAP algorithm. First, we describe the five scenarios designed to run the experiments. Then, we show the experiments on the Coverage problem are analyzed. Here we describe in detail the scenarios used for running the experiments. We designed three different scenarios, shown in Figure 6.6, each one with two levels of waypoint density (H, the higher, and L, the lower density) plus two more scenarios that only have one density level. The scenarios are designed for circular robots except for the last one (called *Rooms*), which is designed for any-shape robots. Furthermore, in Table 6.1 we present the size of each map image and the number of feasible and unfeasible goals for each scenario.

- Mutual Exclusive: three wide parallel horizontal halls, connected between them by two narrow vertical halls; three robots move within the horizontal sections, one in each, and their actuation


(a) Robot 2 - Graphs on Free Configuration Space - 0^o Layer


(b) Robot 2 - Graphs on Navigable Space - 0^o Layer


(c) Robot 2 - Graphs on Free Configuration Space - 90^o Layer

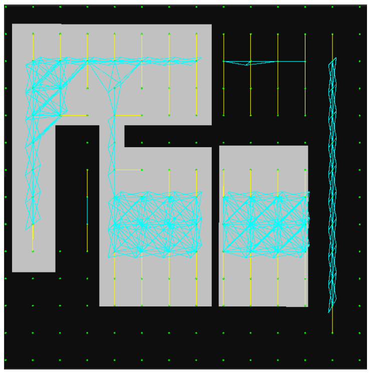
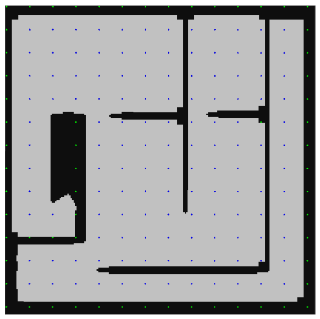

(d) Robot 2 - Graphs on Navigable Space - 90^o Layer

Figure 6.4: The discretized graphs constructed are independent of the initial robot positions, allowing to run the problem from different initial positions; the constructed graphs cover the navigable space not only on its white regions, which depend on the initial position, but also on some black regions if they correspond to white in the free configuration space (independent of initial position).


(a) Robot 1

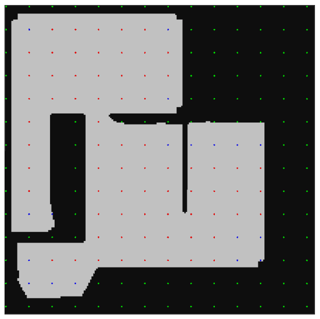
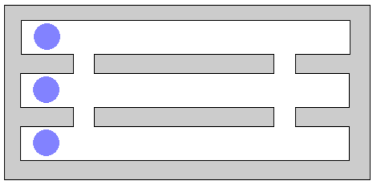
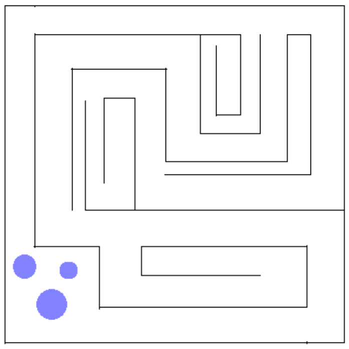
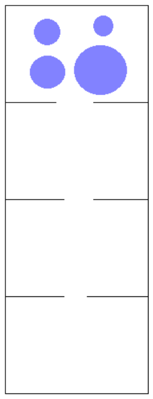
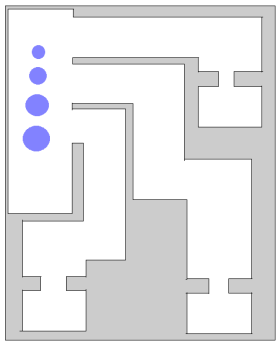

(b) Robot 2

Figure 6.5: All goal waypoints are shown as spheres on top of the actuation map: unfeasible waypoints in green, waypoints covered by the `connected` graph in red, and waypoints only covered by the `actuable` graph in blue; for the smaller robot 1, the two graphs are the same.


(a) Mutual Exclusive


(b) Maze


(c) Corridor


(d) Extremities

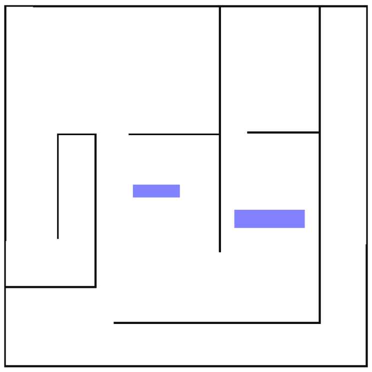

(e) Rooms

Figure 6.6: Maps of the five scenarios used in the experiments; grey regions represent out-of-reach regions which cannot contain goal waypoints, being unfeasible for all the robots; robots positioned in the region of their starting position, in blue circles.

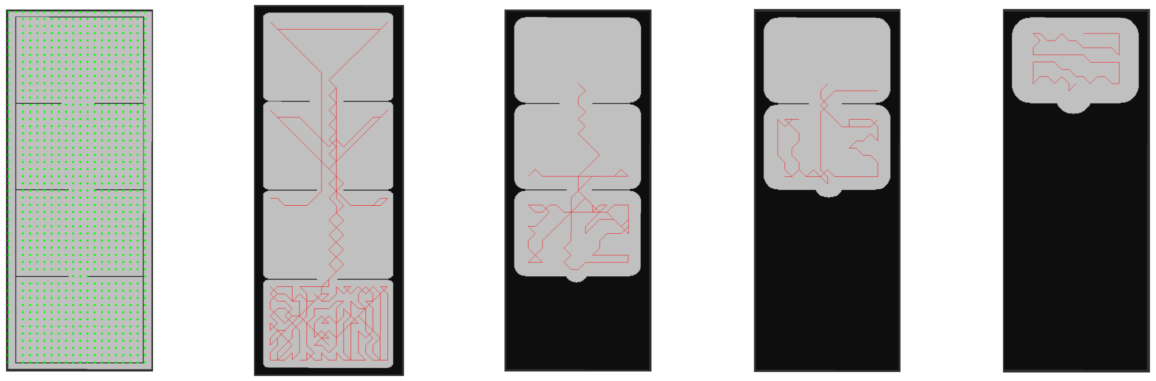
reachability is mutually exclusive.

- Corridor: four wide sections with openings of different sizes connecting them; the opening decreases from the top to the bottom, with all four robots being able to actuate in the top region, but only one being able to reach the bottom.

- Extremities: wide open section with three halls departing to different directions, where all 4 robots actuate; at the end of each hall there is a room that can be accessed through an opening, with only one robot reaching the extremity connected with the smallest opening, to three reaching the one connected with the biggest opening.

- Maze: maze-like scenario with narrow halls and passages with different sizes, resulting in bigger robots not reaching some parts of the maze, or needing to traverse bigger paths to arrive at the same locations as smaller robots.

- Rooms: simple floor plan environment with some room-like spaces connected through passages of



(a) Waypoints (b) Path 1 (c) Path 2 (d) Path 3 (e) Path 4

Figure 6.7: Corridor scenario used in the experiments: a) the waypoint discretization; b) to e), the resulting path for each robot after solving the planning problem using load balance as goal-strategy; path 1 belongs to the smallest robot, while path 4 belongs to the biggest robot.

different sizes as well, used to test the non-circular robot case where they can traverse the passages using only specific orientations.

Figure 6.7 shows a solution example obtained from the multi-agent planning algorithm. It corresponds to the scenario called Corridor-High later on the experiments.

Table 6.1: Number of feasible and unfeasible goals for all robots in each problem, and respective grid size.

	Feasible	Unfeasible	Grid Size
CorridorH	819	118	49x19
CorridorL	384	92	33x13
ExtremeH	1993	1325	51x63
ExtremeL	896	589	34x42
MutExH	499	513	45x21
MutExL	223	242	30x14
Maze	572	100	25x25
Rooms	131	61	13x13

In this section, we show some experiments that test the impact of the pre-processing on the MAP algorithm (MAPM in advance). As it was previously said, we have modeled five different scenarios that include up to four agents with different sizes, and thus different actuation capabilities. Planning results are shown using as metrics the time in seconds, the length of the resulting plan and the makespan. In non-temporal domains, we refer as makespan the length of the parallel plan (number of execution steps, where several actions can be executed at the same execution step). Given that we are dealing with MAP tasks that have no interactions, it is expected that agents can execute their actions in parallel whenever possible.

Four different configurations of our MAP algorithm have been set up:

- MAPM-LB-EC with estimated-cost information (EC). EC refers to the configuration that combines Actuation Maps and MAP.
- MAPM-BC-EC with estimated-cost information (EC), also combining Actuation Maps and MAP.
- MAPM-LB, same as before but without EC information.
- MAPM-BC same as before but without EC information.

The load balance strategy helps to minimize the *makespan* metric. The best cost strategy focuses on minimizing the *plan length* metric. We also run the problems without the pre-processing stage in order to evaluate our impact in terms of computation time and plan quality.

Furthermore, the following state-of-the-art planners have been chosen as a comparison baseline:

- LAMA [44], centralized planer and winner of IPC 2011.
- YAHSP [51], a greedy centralized planner.
- ADP [8], a multi-agent planner that automatically detects agents.
- SIW [29], a multi-agent planner that factorizes the problem into subproblems solving one atomic goal at a time until all atomic goals are achieved jointly.
- CMAP [5], a multi-agent planner that employs a centralized approach to solve the problem.

The three multi-agent planners that have been chosen participated in the 1st Competition of Distributed and Multi-agent Planners (CoDMAP[1]) and obtained good results on the final classification.

Neither of these five planners perform a goal allocation phase separated from the planning process. Thus, we had to test them using the equivalent PDDL problems that do not contain unfeasible goals. Also, in order to fairly compare the results of the makespan metric, we had to apply our parallelization algorithm to the resulting plans of ADP and SIW, as they only return the sequential plan.

We have generated two problems per scenario, one of them with less number of waypoints (which we identify as L in tables) and the other one with a high density of waypoints (H), except for the last two scenarios that only have one density level (*Maze* and *Rooms*), making it a total of eight problems. The *Rooms* scenario works for any-shape robots while the rest work for circular robots. Before discussing the results on the tables, we need to clarify that a maximum of two hours was given to each planner to solve each scenario. YAHSP results do not appear in the tables because it could not solve any of the scenarios.

[1]`http://agents.fel.cvut.cz/codmap/`

The maximum time spent on the pre-processing for any scenario was 170 milliseconds, for the Extremities problem with four robots. We included the pre-processing times (to generate the Actuation Maps) in the GA column of Table 6.2, and in the total time in Tables 6.3. Hardware used for running the planner was IntelXeon 3,4GHz QuadCore 32GB RAM. Actuation maps were computed using a 2.5GHz DualCore 6GB RAM. Table 6.2 is shown to prove the remarkable impact that information from Actuation Maps (AMs) has in combination with the MAP algorithm. Goal assignment (GA) times in Table 6.2 are minimal (MAPM-LB-EC) in comparison with the ones when MAPM-LB needs to compute the relaxed plans for every goal-agent pair. Even though the individual planning time and parallelization time for MAPM-LB-EC is slightly higher than MAPM-LB, the time gains in GA completely dominate the overall planning time.

Table 6.2: Detailed time results in seconds for the MAP algorithm using the Load Balance strategy with and without estimated cost information; from left to right: total time, goal assignment time, individual planning time and parallelization time.

	MAPM-LB-EC				MAPM-LB			
Name	TOTAL(s)	GA	Planning	Parallel	TOTAL(s)	GA	Planning	Parallel
CorridorH	33.58	0.64	24.37	8.57	1232.97	1204.20	20.88	7.89
CorridorL	6.18	0.26	4.62	1.30	128.78	123.59	4.10	1.09
ExtremH	602.68	3.06	428.28	171.34	*timeout*			
ExtremL	58.32	0.92	40.93	16.47	3870.00	3823.75	32.89	13.36
MutExH	7.39	0.34	5.03	2.02	903.65	896.82	4.81	2.02
MutExL	1.39	0.12	1.04	0.23	69.41	68.19	0.98	0.24
Maze	254.40	0.32	210.32	43.76	*timeout*			
Rooms	146.60	0.15	94.30	52.15	1620.02	1554.36	95.39	50.45

Table 6.3: Total time results in seconds; from left to right: MAPM with estimated-cost information in Load-balance (LB-EC); MAPM without estimated cost information in LB; MAPM with estimated cost information in Best-cost (BC-EC); MAPM without estimated cost information in BC; ADP, SIW and CMAP are other multi-agent planners and LAMA is a centralized planner.

	Total Time (s)							
	MAPM-LB-EC	MAPM-LB	MAPM-BC-EC	MAPM-BC	ADP	LAMA	SIW	CMAP
CorridorH	**33**	1232	41	2839	*MEM*	*TO*	96	*TO*
CorridorL	6	128	9	135	104	**5**	9	180
ExtremH	**602**	*TO*	1788	*TO*	*MEM*	*TO*	*TO*	*TO*
ExtremL	**58**	3870	112	3929	*MEM*	*TO*	196	*TO*
MutExH	7	903	7	910	5	6	**4**	1274
MutExL	1	69	1	72	**0.84**	1	1	95
Lab	**254**	*TO*	308	*TO*	427	*TO*	320	*TO*
Rooms	**146**	1620	268	1628	*MEM*	387	288	1328

The easiest scenario to be solved using planning is the Mutual Exclusive (MutExH, MutExL) because it is designed for each robot to traverse a mutually exclusive subset of waypoints, and that is why time results are very similar among all planners except for MAPM-LB and MAPM-BC where the planner needs to compute the relaxed plans for each pair robot-goal. Regarding time results, the fastest configuration is

MAPM-LB-EC if all total times are summed up. Also, the impact of combining information from actuation maps with MAP can be easily appreciated if columns from MAPM-LB-EC and MAPM-LB are compared in Table 6.3. The same happens with best cost configurations. ADP and LAMA were only capable of solving four problems. ADP reached the memory limit (MEM) when planning the solutions before the two hours limit. Even though ADP is a multi-agent planner, the effort of computing plans when all goals are assigned to all agents is considerable. LAMA reached the two hours limit (TO) without returning a solution on the other four problems.

Table 6.4: Plan length; from left to right: MAPM with estimated-cost information in Load-balance (MAPM-LB-EC); MAPM without estimated cost information in LB; MAPM with estimated-cost in Best-Cost (MAPM-BC-EC); MAPM without estimated cost information in BC; ADP, LAMA, SIW and CMAP.

	Plan Length							
	MAPM-LB-EC	MAPM-LB	MAPM-BC-EC	MAPM-BC	ADP	LAMA	SIW	CMAP
CorridorH	1289	1268	1226	**1136**			1154	
CorridorL	605	598	588	475	1403	**470**	492	**470**
ExtremH	3428		**3116**					
ExtremL	1490	1587	1365	**1233**			1398	
MutExH	**642**	**642**	**642**	**642**	748	**642**	723	**642**
MutExL	**277**	**277**	**277**	**277**	278	**277**	339	**277**
Maze	1463	1437	**1355**	1346	1553		1376	
Rooms	473	**469**	475	475	481	478	476	478

Table 6.5: Makespan; from left to right: MAPM with estimated-cost information in Load-balance (MAPM-LB-EC); MAPM without estimated cost information in LB; MAPM with estimated-cost in Best-Cost (MAPM-BC-EC); MAPM without estimated cost information in BC; ADP, LAMA, SIW and CMAP.

	Makespan							
	MAPM-LB-EC	MAPM-LB	MAPM-BC-EC	MAPM-BC	ADP	LAMA	SIW	CMAP
CorridorH	403	408	461	734			**397**	
CorridorL	219	**189**	303	458	1313	286	263	286
ExtremH	**1453**		1929					
ExtremL	556	511	928	1140			**463**	
MutExH	**116**	**116**	**116**	**116**	162	**116**	187	**116**
MutExL	**59**	**59**	**59**	**59**	60	**59**	101	**59**
Maze	466	479	597	793	603		**461**	
Rooms	**240**	**240**	242	242	257	253	245	253

Table 6.4 shows the results regarding the plan length and Table 6.5 the results regarding makespan. The best configuration overall regarding plan length is MAPM-BC-EC. Regarding makespan, MAPM-LB-EC is better. Moreover, MAPM-LB-EC configuration is the best one for problems with a higher density of waypoints, while MAPM-LB proves to be better for reducing makespan in low density problems, explained by the discretization errors from Equation 6.5, which are greater when the downsampling rate is bigger. When allocating goals, the estimation costs are the only guide for the MAP algorithm. The consequence

of having slightly inaccurate cost estimates results in the allocation of some goals to different agents than the ones that the estimated costs from the relaxation of plans would suggest. However, this issue does not have a significant impact on makespan and plan length results.

From the set of planners chosen to compare our approach, SIW obtains the best performance on time, plan length and makespan. SIW can solve most of the scenarios due to its factorization process. The importance of factorizing a MAP problem is a conclusion that can be extracted after observing Tables 6.4 and 6.5, as the planners that do not perform factorization (LAMA, ADP, CMAP, YAHSP) have to solve bigger and more complex tasks.

Regarding our configuration, MAPM-BC-EC's goal-allocation works better than the one performed by MAPM-BC. On the other hand, the lower the number of agents used to plan, the harder the planning task needed to solve for the ones in use. Thus, plan length is better with best cost, but on the contrary, the total time is usually worse than load balance configurations.

6.7 Summary

In this chapter, we showed how to combine information from actuation maps with multi-agent planning to solve a multi-robot path planning problem more efficiently skipping the computation of estimated cost during planning. We used actuation maps in a pre-processing step to determine the feasibility of pairs robot-goal and to extract an estimated cost. That cost is used later to avoid the computation of relaxed plans during goal-assignment. The environment map was discretized into a grid of waypoints. The goals were distributed thanks to a goal-allocation algorithm and unfeasible goals identified and discarded from the planning task. Then, the planning task was factorized for each robot. They generate their individual paths that result in maximal space coverage in terms of actuation.

On the experiments, we have designed a total of eight scenarios, seven for circular robots and one for any-shape robots, which show we were able to reduce the overall planning time when preprocessed information was provided to the multi-agent planner. For the smaller and less complex layouts, almost all planners obtained solutions with the same cost, and some of them were faster than the ones we contributed. However, for problems with bigger domains, our contribution can find solutions while other planners have timeouts or memory problems and are not able to generate a solution. When that happens, our contributed planner has better solutions, or finds solutions with a similar cost but in much less time. Finally, and exactly as expected, the load balance strategy was better at generating good solutions regarding makespan, while best cost is better at optimizing for plan length.

Chapter 7

Multi Robot Planning for Perception of Multiple Regions of Interest

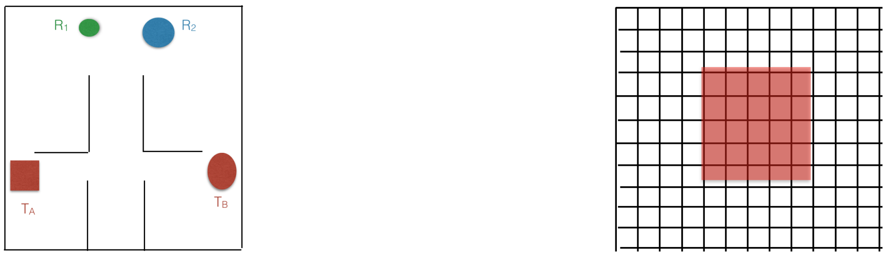
In this work, we consider multiple heterogeneous robots that have to plan together to perceive a set of target regions of interest. We consider the robot's physical characteristics when planning their paths in a structured environment that has been mapped before. For a given environment, not all target positions can be perceived by all robots. Instead of having a single target location that needs to be perceived, this approach uses regions composed of multiple perception targets and calculates waypoints from where targets can be perceived.

To generate waypoints, we use the contributed technique for perception planning of single targets. Running simple search instances for every target that belongs to the regions of interest for perception, we generate many perception locations to optimally perceive specific targets, and those locations are mostly grouped together in space. Thus the perception locations are clustered to create waypoints from where we know robots can perceive multiple targets.

From our previous work, we contributed a heuristic-based waypoint allocation method that distributes waypoints among the heterogeneous robots, minimizing both motion and perception costs for a team of robots with different capabilities [36].

7.1 Problem Formulation

We consider a 2D grid map of obstacles to represent the environment and mobile robots that are heterogeneous regarding geometric properties, such as size and sensing range. As shown in Figure 7.1, we assume there is a set of heterogeneous robots (Rs) and target regions of interest (Ts) that need to be perceived. The target regions can represent areas that need to be covered by the robot's sensors for inspection or search. The regions of interest could also represent location uncertainty around a point that needs to be perceived, and the target regions can have any shape and size.



(a) Problem (b) Target Region in Grid Map

Figure 7.1: a) Environment with obstacles represented in black, circular robots R and target regions of interest T that need to be perceived; b) a target region with a given shape, size and position in a grid map which results in the discretization of T.

In traditional multi-robot path planning for perception tasks, an infinite perception range is a common assumption, or even a finite maximum range. However, determining paths for robots executing perception tasks should also include the cost of perception. Therefore, we introduce the following problem, where the goal is to find paths for each robot that minimize the total cost of motion and perception, given by

$$\text{cost} = \sum_{R} C_R + \lambda \sum_{T} C_T, \tag{7.1}$$

where C_R is the path size for robot R, C_T is the cost of perception of target region T, and λ is the trade-off parameter between perception cost and motion cost. We assume all target regions have to be observed.

The cost of perception of a target region T perceived from a robot depends on its path ρ, and we assume it is the average of perception cost for the grid points inside the region of interest.

$$C_T(\rho) = \frac{1}{\#T} \sum_{\mathbf{t} \in T} \min_{\mathbf{p} \in \rho} c_p(||\mathbf{p} - \mathbf{t}||) \tag{7.2}$$

The number of points of the grid map inside the target region is represented by $\#T$. For multiple robots, C_T uses the minimum of the perception cost not only for ρ, but the paths of all robots.

The perception cost function, c_p, models sensor accuracy and it is a function of perception distance d_p. As an example, if the sensing error increases quadratically with distance, then the perception cost is a quadratic function.

$$c_p(d_p) = d_p^2 \tag{7.3}$$

Given the problem with robots Rs and targets Ts, a planner finds the paths for each robot such as all the target regions are perceived by at least one robot, and the overall cost function is minimized. We assume the total motion cost to be a weighted sum of all paths' sizes, thus minimizing the energy spend to move the robots by using appropriate weights for each robot.

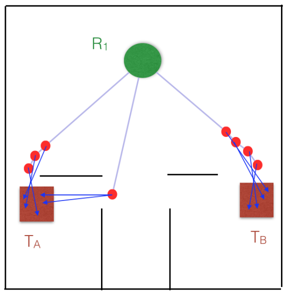
The approach we contribute starts with a first step to determine perception points for each target grid point. For that we use PA* [35], a technique to determine from a given initial position the optimal perception position to perceive a target, assuming some perception cost function and the λ parameter. We then cluster the perception points and use the clusters as new initial positions from where to rerun PA*. Our algorithm is then able to obtain a set of clusters used as waypoints for path planning.

In the second step, the planner uses the set of waypoints to construct paths for each robot. Given the combinatorial nature of our problem, we use a constructive heuristic to iteratively add new waypoints to the robots' paths, and construct a solution that covers all the targets that need to be perceived, while minimizing the overall cost. We contribute an algorithm that can be used to find paths to perceive target regions of interest both for single and multi-robot teams.

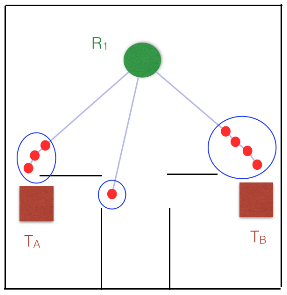
In the next sections, we describe our proposed method in more detail.

7.2 Perception Clusters from PA*

We start by considering first a single robot scenario. For each target grid point **t** inside target regions of interest, we run PA* to find a path to perceive **t** from initial robot position **r**, optimizing for both motion and perception costs using λ as the trade-off parameter, as shown in Figure 7.2. PA* returns the optimal path with minimal cost, where the final position is the optimal perception point. PA* search results in a perception point $\mathbf{p_t^r}$ for each **t**.



(a) Perception Points



(b) Perception Clusters

Figure 7.2: a) When running PA* from the robot initial position to each point inside target regions, the search returns an optimal perception point, shown as a red dot; b) in order to reduce the number of possible combinations, the perception points are clustered in groups as single waypoints for path planning.

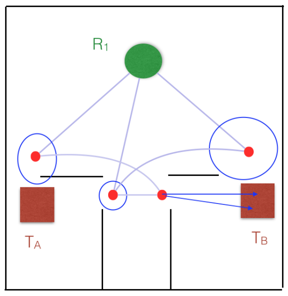
We should note that this perception position is optimal only for the local scenario of a robot starting at **r** to perceive **t**, but it is not necessarily optimal in the multiple target regions scenario. However, we use these points as an initial step for constructing paths for the robots to perceive those regions.

The algorithm then computes the robots' paths as a combinatorial solution of the determined perception points. Unlike the traveling salesman problem (TSP), not all perception points need to be visited, and the robot does not need to return to the initial position. To avoid a combinatorial explosion for path planning, we cluster perception points based on distance. The point closer to each cluster's center of gravity is the one used as a waypoint in the path planning, and the perception cost for each $\mathbf{p}_\mathbf{t}^\mathbf{r}$ associated with the respective cluster.

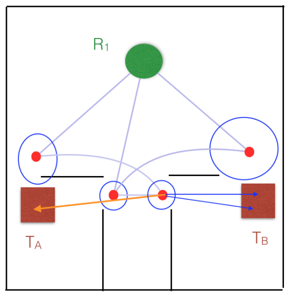
The proposed approach does not find all needed perception points, as the optimal paths from PA* depend on the initial position. So, the PA* search to targets $\mathbf{t}$ needs to be rerun again from each cluster centroid, resulting in new perception points $\mathbf{p}_\mathbf{t}^\mathbf{q}$. New clusters might appear from each iteration when running PA* from new initial positions, as shown in Figure 7.3a. If a new cluster's centroid is close to an existing one, they can be merged, with the robot radius being the merging threshold. The cost of perception of target point $\mathbf{t}$ in cluster P_i is

$$c_\mathbf{t}^i = \min_{\mathbf{p}_\mathbf{t}^\mathbf{q} \in P_i, \mathbf{q} \in \{\mathcal{Q} \cup \mathbf{r}\}} c_p(||\mathbf{p}_\mathbf{t}^\mathbf{q} - \mathbf{t}||), \tag{7.4}$$

where $\mathcal{Q}$ is the set of cluster centroids.



(a) Additional Clusters



(b) Ray Casting Extension

Figure 7.3: a) When running PA* from cluster centroids, new perception points might result in new clusters; b) from that clustering strategy some clusters might only be associated with specific targets, and additional perception feasibility to other target points can be obtained using ray casting to test for line-of-sight.

Running PA* to target points $\mathbf{t}$ from different initial positions generates clusters, but $c_\mathbf{t}^i$ is only updated if PA* searches to $\mathbf{t}$ result in perception points that are clustered to P_i. Nevertheless, other target points might still be observable from cluster P_i, even if PA* finds the cluster position non-optimal to perceive those points. In Figure 7.3b, for every cluster centroid, ray tracing is used to determine line-of-sight and perception cost to other target points $\mathbf{t}$ whose cost was not previously determined as $c_\mathbf{t}^i$. Ray tracing determines perception feasibility from a cluster centroid to any other target point, and the respective distance is used to associate a perception cost to the tuple centroid-target point.

7.3 Path Construction

Even though there might not be any connections between some pairs of clusters initially, we still consider them in the heuristic path construction, as shown in Figure 7.4, because PA* is optimal locally for each target point but is globally sub-optimal in the general multi-target path planning setting.

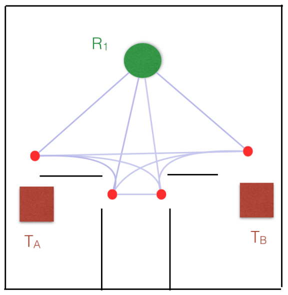


Figure 7.4: Map with robot and target regions of interest, with red dots as cluster centroids and lines connecting all of them showing all the path's combinatorial possibilities.

The clusters centroids can be used as waypoints when determining the path for a robot to perceive all the target points. Pairwise distances between all cluster centroids and initial robot position can easily be determined with A*. The waypoints are $\mathbf{q}_j$, with $0 \leq j \leq m$ where m is the number of clusters and $\mathbf{q}_0 = \mathbf{r}$ is the initial position. The path ρ is a sequence $\{s_i\}$, with $0 \leq i \leq L$ (L is path length in terms of number of clusters covered) and $1 \leq s_i \leq m$ for $i \geq 1$ and $s_0 = 0$. The path cost is given by:

$$\text{cost}(\rho) = \sum_{i=1}^{i \leq L} \text{dist}(\mathbf{q}_{s_{i-1}}, \mathbf{q}_{s_i}) + \lambda \sum_{T} \left(\frac{1}{\#T} \sum_{\mathbf{t} \in T} \min_{1 \leq i \leq L} c_{\mathbf{t}}^{s_i} \right). \tag{7.5}$$

Any point can be visited more than once, but that would be redundant. Moreover, not all points need to be visited. Given the combinatorial characteristics of this problem, solving it optimally for any $m > 10$ is already very time consuming. Therefore, we use a construction heuristic to iteratively construct a path from the initial position that covers all the target points with the robot's sensor. Examples of constructive heuristics used in the TSP are the nearest neighbor, nearest insertion, cheapest insertion, and farthest insertion.

Improvement heuristics can be used to improve the solution after finding a feasible path. Examples are point removal, k-opt moves, and meta-heuristics.

At each iteration, and for each point i that can still be inserted in the robot's path, the added motion cost is the cheapest insertion, which finds the best position in the current path to insert the new point.

$$\text{cost}_m(i) = \min \left(\min_{1 \leq j \leq L} \text{dist}(\mathbf{q}_{s_{j-1}}, \mathbf{q}_i) + \text{dist}(\mathbf{q}_i, \mathbf{q}_{s_j}) - \text{dist}(\mathbf{q}_{s_{j-1}}, \mathbf{q}_{s_j}), \text{dist}(\mathbf{q}_{s_L}, \mathbf{q}_i) \right) \tag{7.6}$$

For each point to be inserted, there is also a possible gain associated with the improvement in perception cost from sensing from a closer distance.

$$\text{gain}_p(i) = \lambda \sum_T \frac{1}{\#T} \sum_{\mathbf{t} \in T} \max\left(\min_{0 \le j \le L} \left(c_{\mathbf{t}}^{s_j} \right) - c_{\mathbf{t}}^{i}, 0 \right) \tag{7.7}$$

We use for $c_{\mathbf{t}}^0$ the maximum perception cost, $\lambda c_p(r_p)$, where r_p is the maximum perception range. The bigger $c_{\mathbf{t}}^0$, the highest priority is given to points that perceive previously unseen target points, which is a behavior similar to the farthest heuristic. Points are valid if the gain is positive, or if it adds visibility to any previously unseen target. Otherwise, the planner might not add to the path the only positions that can observe some far away target, even though we want complete coverage in terms of perception. Algorithm 1 shows the overall base method.

Algorithm 1 Base Path Construction from Cluster Centroids

```
Require: List of points to insert: $\{1..m\}$
 1: while There is valid points to choose from do
 2:     for all points not yet inserted do
 3:         Find added motion cost, $\text{cost}_m(i)$ as cheapest insertion of point $i$
 4:         Find gain in perception cost, $\text{gain}_p(i)$
 5:         $\text{gain}(i) = \text{gain}_p(i) - \text{cost}_m(i)$
 6:         if $\text{gain}(i)$ is valid then
 7:             Add $i$ to list of points to consider for insertion in this iteration
 8:         end if
 9:     end for
10:     Choose point that maximizes gain
11:     Insert point in path according with cheapest insertion
12:     Update path perception cost to each target point
13: end while
14: Return $Path$
```

7.3.1 Avoiding Local Minima

As shown in Figure 7.5, the base algorithm presented before can very easily get stuck in local minima, as it uses a greedy heuristic. In the figure's example, in the first iteration cluster 1 has the highest gain and is added to the robot's path, but as we show later, that cluster is not even part of the optimal path.

To help avoid local minima, we contribute an n-level depth search for the greedy constructive heuristic. Instead of looking only one step ahead, it looks at the insertion of n points and chooses the one with minimal cost. For that purpose, we use Algorithm 2, where we contribute a recursive function that implements the n depth search and determines the best combination of n points to insert, then chooses the first point to insert and repeats the process. This function is called once in each iteration, returning the best point to insert in the path at each time, until there are no points to insert in the robot's path.

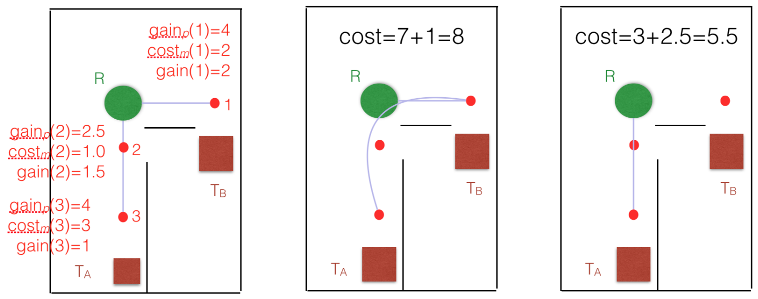


Figure 7.5: In the left image, the robot can move to three cluster centroids from its initial position, and cluster 1 has the highest gain, considering a quadratic perception cost and $\lambda = 0.5$; in the middle figure we show a path that moves through cluster 1 and then to cluster 3 in order to perceive both T_A and T_B, with motion cost 7 and perception cost of 1 ($2 \times 0.5 \times 1^2$); in the last image, we show the optimal path that moves through cluster 2 and then cluster 3, perceiving both targets with a lower overall cost, motion cost equal to 3 and perception cost of 2.5 ($0.5 \times 2^2 + 0.5 \times 1^2$).

Algorithm 2 Recursive function used in n-depth heuristic for path construction

```
Require: List of points to insert: ${1..m}$
 1: function SEARCH(dists, $c_t$'s, path, $n$)
 2:     if n==0 then return $< -1, 0 >$
 3:     end if
 4:     for all points not yet inserted do
 5:         Find added motion cost, $\text{cost}_m(i)$ as cheapest insertion of point $i$
 6:         Find gain in perception cost, $\text{gain}_p(i)$
 7:         $\text{gain}(i) = \text{gain}_p(i) - \text{cost}_m(i)$
 8:         if $\text{gain}(i)$ is valid then
 9:             Create new temporary path, $\text{path}_i$, updated with insertion of point $i$
10:             Find the gain from next (n-1)-depth search:
11:             $< j, \text{nextgain}(i) >= \text{SEARCH}(\text{dists}, c_t\text{'s}, \text{path}_i, n-1)$
12:             $\text{overall\_gain}(i) = \text{gain}(i) + \text{next\_gain}(i)$
13:         end if
14:     end for
15:     if no valid point then
16:         return $< -1, 0 >$
17:     end if
18:     Choose point that maximizes overall gain
19:     return $< i, \text{overall\_gain}(i) >$
20: end function
```

Because we consider combinations of n points and we use the cheapest insertion heuristic, a 2-level search that inserts first the cluster i and then the cluster centroid j has the same gain as the reverse, inserting first cluster j and then i. As a tiebreaker rule, we insert first the point with the highest gain in the top level of the recursive search (variable determined on line 7 of Algorithm 2).

For the path construction algorithm, we also created a brute-force method that can solve smaller problems with lists with few points to insert in the robots' paths, resolving the problems optimally. For those simpler scenarios, we can compare the results from the brute-force algorithm with the output of Algorithm 2 to validate the algorithm correctness. For larger domain problems, the algorithm above was the only solution implemented, thus in those larger problems we don't have a baseline to compare to, trusting the qualification obtained with smaller problems only. The most common depth chosen in our tests was $n = 2$ or $n = 3$.

7.4 Extending Heuristic Path-Constuction to Multiple Robots

The extension of the previous n-depth heuristic from the single robot approach to the multiple robot setting is now straightforward. We build clusters of perception points from PA* for all the robots. Then the construction heuristic considers multiple lists of cluster centroids, and at each search level, it can choose to add any of those points to the respective robot's path. Insertion on paths at different depth levels of the recursive search might be for different robots.

The complexity of the n-level heuristic search in the multi-robot scenario is $M!/(M-n)!$ in each iteration, where M is the total number of cluster centroids over whole the robots. In each iteration, one cluster is added to a robot's path.

However, new inefficiencies of the heuristic arise in the multi-robot scenario, as shown in Figure 7.6. In that example, either cluster centroids 1 or 2 can be added to the respective robot's paths. From point 2, all target points can be observed, but from point 1 only part of T_A can be observed. Using constructive heuristic with a 1-level search, adding point 1 to R_1 path has a higher gain, even though in the next iteration R2 will still have to move to point 2 to perceive the yet unseen parts of T_A, resulting in sub-optimal path construction. In some cases, this inefficiency can be solved with higher n, as here a 2-level search would already avoid this problem. Nevertheless, for big problems with multiple targets and robots, n has to be small to reduce the search complexity, and might not be enough to solve this inefficiency.

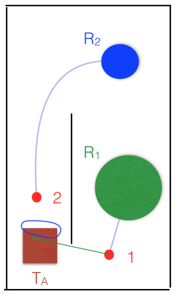


Figure 7.6: Inefficiency arising from different robots being able to perceive targets from different locations; here R_2 can move to cluster 2 and perceive all points in T_A, but R_1 can only move to the first cluster where it only perceives a part of the target.

7.4.1 Unfeasibility Subsets

There are target points that can be perceived by all robots and others that can only be observed by a subset of robots. Therefore, the idea is, at each iteration of the path construction phase, to consider first the cluster centroids that are the only ones that can observe some target points. We start by centroids that are associated with targets that are perceived by one robot only, then by two, and so on, until the only remaining are the ones that can be observed by any robot. Using this approach solves the problem in Figure 7.6 without increasing n. The separation of cluster centroids by subsets of unfeasibility is accomplished by adding a component to the gain that is proportional to the number of robots that cannot perceive a target, and the maximum gain, $K\lambda c_p(r_p)$, where K is the number of regions.

Algorithm 3 shows our complete contribution using unfeasibility sets. Compared to the previous version, Algorithm 2, this algorithm improves by considering not only one list of waypoints to insert but one list per robot, extending our solution to multiple robot scenarios. Moreover, calculating in line 8 for each target point the unfeasibility gain, we compute a bias in the overall gain function that takes into account the unfeasibility subsets.

Algorithm 3 Recursive n-level constructive heuristic with unfeasibility subsets

```
Require: List of points to insert: $\{1..m\}$
 1: function SEARCH(dists, $c_{\mathbf{t}}$'s, paths, $n$)
 2:     if n==0 then return $< -1, -1, 0 >$
 3:     end if
 4:     for all $< r, i >$ all robot and cluster points not yet inserted do
 5:         Find cost$_m(r,i)$, as cheapest insertion of point $i$ in path of robot $r$
 6:         Find gain in perception cost, gain$_p(r,i)$
 7:         for all t do
 8:             if t is not yet observed by any robot path then
 9:                 unfeas_gain$(r,i,\mathbf{t})$ = #(Robots that cannot perceive $\mathbf{t}$) $\times$ $(K\lambda c_p(r_p))$
10:             end if
11:         end for
12:         gain$(r,i)$ = gain$_p(r,i)$ $-$ cost$_m(r,i)$ + max$_{\mathbf{t}}$(unfeas_gain$(r,i,\mathbf{t})$)
13:         if gain$(r,i)$ is valid then
14:             Create new temporary paths, paths$_i$ updated with insertion of point $i$
15:             Find the gain from next (n-1)-depth search:
16:             $< s, j,$ next_gain$(r,i) >=$ SEARCH(dists, $c_{\mathbf{t}}$'s, paths$_i$, $n-1$)
17:             overall_gain$(r,i)$ = gain$(r,i)$ + next_gain$(r,i)$
18:         end if
19:     end for
20:     if no valid point then
21:         return $< -1, -1, 0 >$
22:     end if
23:     Choose point that maximizes overall gain
24:     return $< r, i,$ overall_gain$(r,i) >$
25: end function
```

Taking into account the subsets allows the algorithm to reduce the inefficiencies of incremental path construction by assigning first to robots the waypoints that perceive targets with limited coverage by the robots.

7.4.2 Simulation Example

We show in Figure 7.7 the resulting paths for the planning problem of two heterogeneous robots perceiving three regions of interest, for a large λ that makes robots move close to the target regions. We consider two test scenarios with a changing position for one of the target regions, and we show how it impacts the resulting plan. The smaller robot 1 can get into the region where the changing target is, and observe it from a close distance. However, the bigger robot 2 can only perceive this region from a distance. Therefore, when the target moves closer to the opening from where it is perceived, the perception cost for the bigger robot reduces, and the planner moves this robot such as it perceives two target regions, while the first robot moves to perceive the target that can only be observed by the first robot. Nevertheless, when the changing target moves away from the opening, the quadratic perception cost for robot 2 increases significantly, and as a result, there is a point from where it is worth for robot 1 to move forth and back to observe all the target regions from a closer distance.

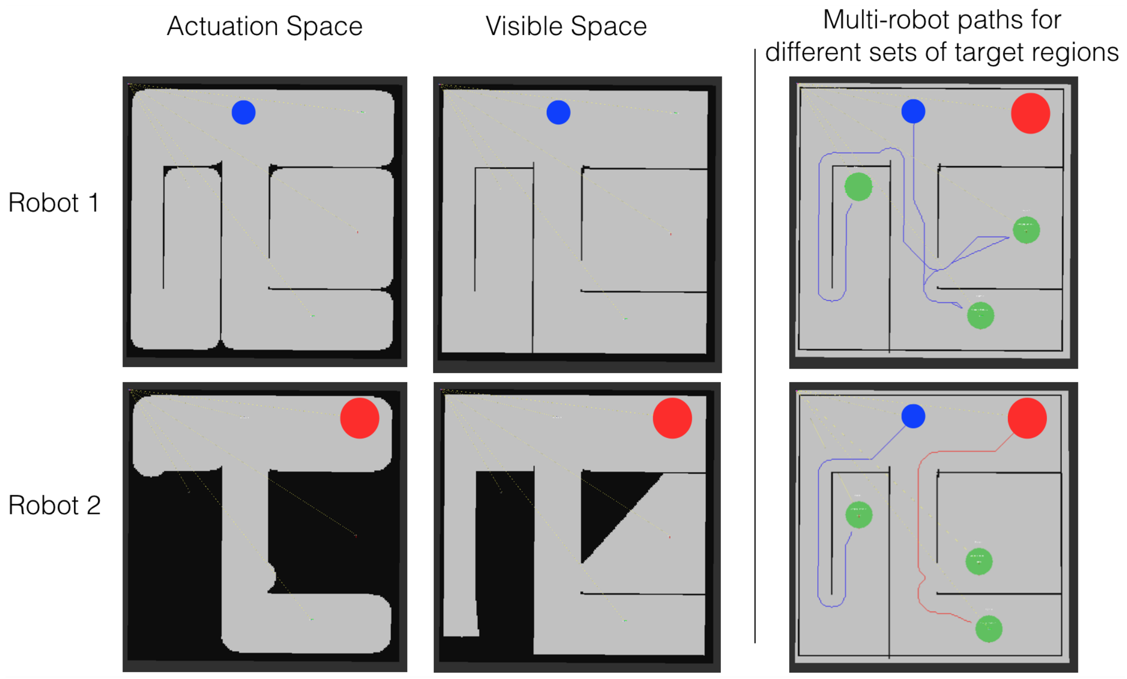


Figure 7.7: Planning scenario with two heterogeneous robots and 3 regions of interest; in the first column we show the space where robots can move, and in the middle column the associated visible space of each robot; in the last column, the resulting paths for the robots when one of the regions of interest changes its position.

For scenarios with cluster lists up to ten centroids per robot, we also run a brute-force algorithm to test all possible combinations and compare them with our heuristic. In the simulated environment we used, in Figure 7.7, with varying targets' sizes and positions, the heuristic always returned the same paths as the brute-force algorithm, but with lower computation time, in the order of seconds, proving its efficiency. For bigger cluster lists, we could only use the heuristic approach for path planning. For the problems in Figure 7.7, in a map with 200 by 200 pixels, and a total of five clusters for the two robots, the cluster determination took around 30 seconds, and the path construction 5 milliseconds.

7.5 Visibility Maps for Efficient Perception Cluster Determination

There is a high computation burden just in the determination of the perception points clusters, which results from many PA* searches from the clusters centroids and initial robot position to all the target points in the target regions.

Robot-Dependent Visibility Maps (RDVMs) provide structure information about the environment that can help alleviate the heavy computation effort by bypassing the PA* searches. For each unreachable target, RDVMs provide a list of critical points from where targets can be observed.

We use critical points to avoid repeated PA* searches, determining perception positions for each target from the critical point location, maintaining the clustering method and all the following methodology, including the construction heuristic to do path planning with cluster centroids.

The path construction also remains the same algorithm, with only the determination of perception points for each target being done using the critical points instead of PA*.

7.5.1 Perception Cluster Determination from Critical Points

In this section, we show that with an initial fixed cost of building the RDVM, it is possible to use the critical points to improve the cluster determination with them instead of using PA*.

Points in the unreachable regions have some information about a possible position from where they can be sensed because they have associated a critical point. The distance between a target in region $U^l(\mathbf{p}_0)$ and a critical point $\mathbf{c}^*_{li}(\mathbf{p}_0)$ can be used as an estimate of the perception distance.

Figure 7.8 illustrates how critical points have the potential of being used as the equivalent to perception cluster centroids. Moreover, we can see that while before the PA* resulted in 3 clusters when searching from the initial position, and the forth appeared only when searching from the cluster centroids, with the critical points the 4 points are known before-hand, without using any path search.

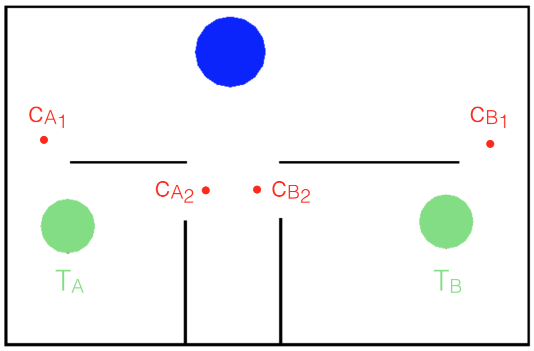


Figure 7.8: Planning scenario with one robot and two regions of interest, and the multiple critical points associated with the target regions of interest, dependent on the environment structure and robot footprint shape.

The figure shows target regions A and B, each with two associated critical points. We asume perception points for targets can be found by running a simple A* search from the initial position to the multiple critical points of each target.

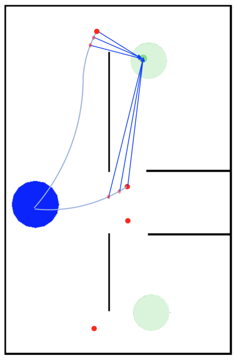
The increased efficiency of this approach comes from A* search to critical points being reused for all the targets that share the same critical points. The number of A* searches for each iteration of perception point determination (different initial position) is only a function of the number of critical points, instead of the number of target points, as we had before with the PA* approach.

For each perception cost function c_p, there is an optimal sensing distance in a straight line with no obstacles. For the quadratic cost function, the optimal sensing distance of PA* is $1/(2\lambda)$, where λ is the trade-off parameter between motion and perception cost. If optimal sensing distance is less than the distance between critical point and target, the path to the critical point is the optimal solution of PA* (ignoring other critical points), and the perception point determination becomes immediate.

However, if the optimal sensing distance is greater than the distance from the target point to the critical point, the critical point position is too close to the target, and the optimal perception point should be further away. So, the perception point position can be estimated by taking testing positions along the path to the critical point that have line-of-sight to the target and have a distance less or equal than the optimal sensing distance, shown in Figure 7.9a. The search does not need to start iterating from the beginning of the path but from a given distance related to the optimal sensing distance. This fact holds true because other positions further away along the path that could optimally perceive the target can be other critical points, and all critical points are eventually used for perception point determination.

The full algorithm to determine perception locations from critical points is in Algorithm 4.

For the problem we have in Figure 7.9b, with around 3000 target points in unreachable space, our solu-


(a) Perception Point from Critical Point

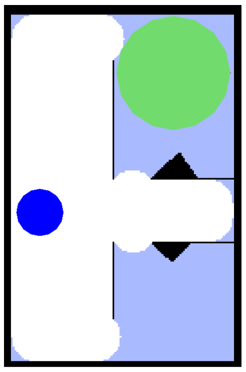

(b) Large Target Unreachable Region Testing

Figure 7.9: a) Determining perception point from path planning to critical points, testing positions along that path with line-of-sight to a specific target point; b) scenario with one large region of interest in unreachable space to test the efficiency of our approach with critical points.

Algorithm 4 Perception Point from Path to Critical Point

```
Require: Critical Point $\mathbf{c}$, current position $\mathbf{q}$, optimal distance $d_s^*$, target $\mathbf{t}$
 1: if path($\mathbf{q}, \mathbf{c}$) exists then
 2:     $\rho \leftarrow$path($\mathbf{q}, \mathbf{c}$)
 3: else
 4:     Determine path $\rho$ from $\mathbf{q}$ to $\mathbf{c}$
 5:     Save path $\rho$: path($\mathbf{q}, \mathbf{c}$)$\leftarrow \rho$
 6: end if
 7: $\mathbf{p_t^q} \leftarrow \mathbf{c}$                                   ▷ Initialization
 8: $c_m \leftarrow \rho$.cost
 9: $c_s \leftarrow \lambda c_p(||\mathbf{c} - \mathbf{t}||)$
10: offset $\leftarrow ||\mathbf{c} - \mathbf{t}||$
11: margin $\leftarrow \max(d_s^* - \text{offset}, 0)$
12: if margin $> 0$ then
13:     $b \leftarrow \max(\rho$.size -1- ceil(margin)$, 0)$
14:     valid $\leftarrow$ false
15:     for $i$ from $b$ to $\rho$.size-1 do
16:         $\mathbf{n} \leftarrow \rho[i]$
17:         if not valid then
18:             $d' = ||\mathbf{n} - \mathbf{t}||^2$
19:             if $d' < r_p^2$ and raytracing valid then
20:                 valid $\leftarrow$ true
21:                 $c_s \leftarrow \lambda c_p(\sqrt{d'})$
22:                 $\mathbf{p_t^q} \leftarrow \mathbf{n}$
23:             end if
24:         else
25:             $c_m - = \text{dist}(\mathbf{n} - \mathbf{n'})$
26:         end if
27:         $\mathbf{n'} \leftarrow \mathbf{n}$
28:     end for
29: end if
30: Return $< c_m, c_s, \mathbf{p_t^q} >$
```

tion using critical points took 560 milliseconds to compute the perception points and respective clustering, while our previously proposed approach with PA* took 22 seconds.

7.5.2 Perception Point Determination for Points in Navigable Space

There are no critical points for targets in the navigable space. Thus the previous technique will have no impact in many situations, such as the ones with target regions of interest inside the navigable space or actuation space (Figure 7.10). In those cases, the bottleneck of perception cluster determination is going to continue being a problem.

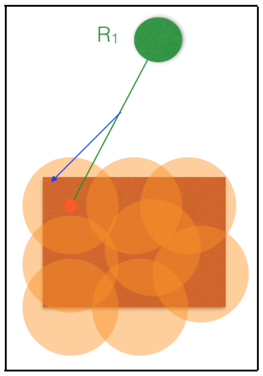
Therefore, in the case of critical point absence, we propose to create groups of targets in the navigable space, using the optimal sensing distance as the clustering threshold, and use the group center as an equivalent to the critical point. Algorithm 5 shows the process of clustering in the navigable space. As before, A* path planning to the cluster centroid determines the perception point for targets. The algorithm still uses ray casting to test for line-of-sight, and search along the A* path starts before the cluster centroid, by twice the optimal sensing distance.

Algorithm 5 Clustering of Target Point in Navigable Space

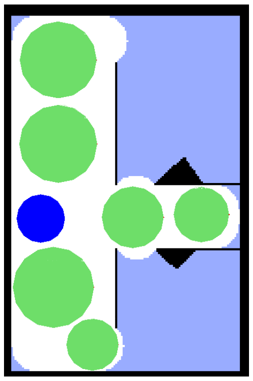
```
Require: Set of target navigable points $\{\mathbf{t}\}$
1: $\{C\} \leftarrow$ CLUSTER($\{\mathbf{t}\}$)
2: $G \leftarrow \emptyset$
3: for each $C$ do
4:     $\mathbf{p}_C \leftarrow$ CENTEROF($C$)
5:     for each $G_i \in G$ do
6:         $\mathbf{p}_i \leftarrow$ CENTROIDFROMCLUSTER($G_i$)
7:     end for
8:     $\mathbf{p} \leftarrow \text{argmin}_{\mathbf{p}_i} ||\mathbf{p}_i - \mathbf{p}_C||$
9:     if $||\mathbf{p} - \mathbf{p}_C|| < d_s^*$ & valid ray casting then
10:        $G^* \leftarrow$ CLUSTERFROMCENTROID($\mathbf{p}$)
11:        $G^* \leftarrow G^* \cup C$
12:    else
13:        $G_n \leftarrow C$
14:        $G \leftarrow G \cup \{G_n\}$
15:    end if
16: end for
17: for each $G_i \in G$ do
18:     $\mathbf{p}_i \leftarrow$ CENTROIDFROMCLUSTER($G_i$)
19: end for
20: Return $\{\mathbf{p}_i\}$
```

The algorithm used is the same as Algorithm 4, but the margin from line 11 becomes two times the optimal sensing distance.

For points in the actuation region, given by the robot-dependent map, we determine all the reachable points from where it can be actuated, using the robot radius for the search window, we then proceed to the clustering step, and for each cluster, we choose the closest reachable point.



(a) Perception Points from targets in Reachable Space



(b) Test with multiple Target Regions in Reachable Space

Figure 7.10: a) The path plan to a target group center in reachable space, and how the positions along that path are tested to access line-of-sight to a specific target point when determining the perception point position from the group center; b) scenario with multiple large regions of interest in the reachable space to test the efficiency of our approach with the extended concept of critical points to reachable space.

7.5.3 Comparison with PA*-Based Cluster Determination

We tested our method in comparison to PA* searches using the scenario shown in Figure 7.11. As stated previously, the PA* solution for this example took approximately 34s (33s for cluster determination and less than one for the heuristic path construction), while our new solution using critical points took only 100ms in total.

Figure 7.11 shows one other benefit of using critical points. In some cases, even running PA* from previously found clusters does not find all possible perception points, leading to sub-optimal path solutions because the path construction uses an incomplete set of perception positions. In this case, using visibility maps, we were able to obtain a solution with cost 329 in 106 milliseconds, while with PA* the non-optimal solution found had cost 348 and took 4.7 seconds.

Using visibility maps, every target point has associated with them all the critical points from where it can be observed, resulting in a complete set of perceptions points. Thus the solutions with critical points are less probable to be stuck in local optima, having better cost solutions with less computational cost.

7.5.4 Experiment With Variant Heuristics from Visibility Maps

We run an experiment with multi-robot path planning for perception tasks, shown in Figure 7.12. This problem has two robots, and a set of regions of interest defined by the user, that have to be observed by any of the robots, while minimizing both the motion and perception cost with the trade-off parameter

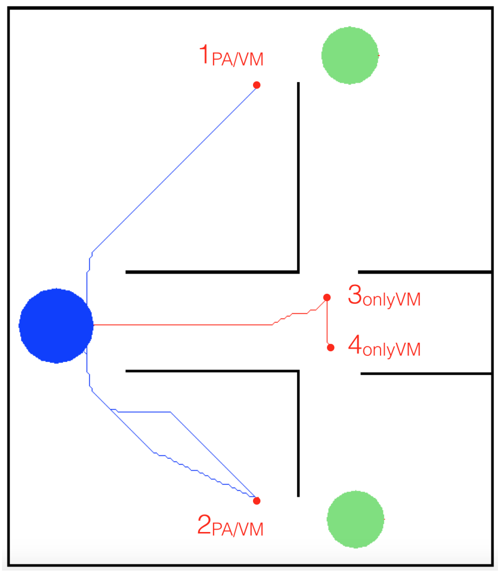


Figure 7.11: An example of a scenario where PA* searches would return an incomplete and thus suboptimal solution, while using critical points yields a complete solution that considers all possible perception positions for the target points with less computational effort; blue path is costly solution with PA* technique, and red is the more cost-efficient solution using visibility maps.

λ. This problem can be solved by running PA* from different locations and clustering the perception positions in waypoints that can be used with a constructive heuristic to find paths for the robots. This method is heavily dependent on PA*, and we evaluated the impact of our heuristic improvements. As shown in Figure 7.12, we chose thirty regions of interest with equal size, out of each only ten are inside unreachable regions and can benefit from our contributions, while the other twenty will just run with the base PA*. Moreover, we use $\lambda = 0.04$, which results in a small $d_s^* = 12.5$, but the ten regions that benefit from our heuristics are evenly distributed in space. Thus some of them have large distances to the respective critical points, while others have small perception distances from the critical points.

The first part of the algorithm runs PA* multiple times and clusters the perception points, taking 2439 seconds when using the base heuristic, and 1088 seconds when using our contributed PA*2SE variant, reducing more than half the total computation time of this phase. Moreover, the time to compute the visibility maps was only 2 seconds, which is completely negligible compared to the total time to complete all the PA* searches and clustering. The second phase, with the constructive heuristic, took 321 seconds to compute the robot paths, shown in the figure, as a combination of waypoints determined from the first phase.

In our approach, we plan paths for each robot without taking into account conflicts between the robots. We assume that a post-processing iteration could easily solve most conflicts, but for that assumption to hold, there would have to be few conflicts in the solutions we generate. So our contribution has the

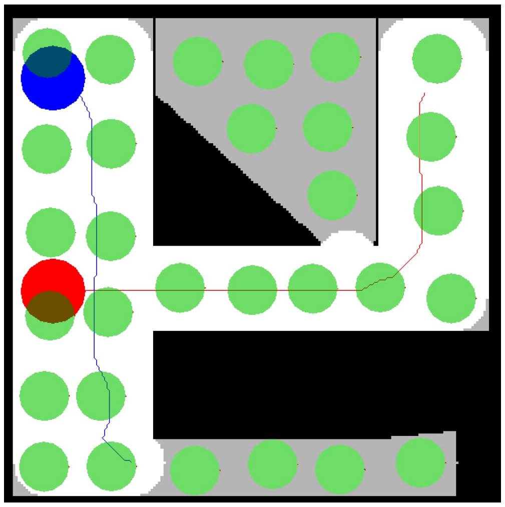


Figure 7.12: Path planning for two robots (blue and red) which have to perceive a set of regions of interest (green), minimizing both motion and perception cost, in a 200x200 grid map, with both robots having size 13, $r_p = 130$ and $\lambda = 0.04$; white represents actuation space, gray the visible space, and black the obstacles or non-visible space.

most value in scenarios with sparse interactions, not cluttered with obstacles, allowing robots space to maneuver around each other in most situations.

7.6 Summary

In this chapter we contribute a constructive heuristic for path planning, to use with heterogeneous multi-robot settings in the problem of perception of multiple regions of interest. The solution can be used in inspection, surveillance or search in robotics. We introduce mechanisms to avoid local minima of the proposed heuristic, such as considering sets of unfeasibility, and n-depth search. We were able to successfully generate paths for multiple robots in simulated environments, in a novel problem that considers both motion and perception cost. Furthermore, we used visibility maps and their structured information of critical points to reduce the time complexity of our algorithm, introducing an alternate method of determining the perception points. This alternative method enables us to reduce the time spent on perception cluster determination, the bottleneck of our algorithm using PA*.

In this way, we have a formal representation of heterogeneity and use that to coordinate multi-robot teams efficiently, with robots executing tasks according to their characteristics. We were able to have multiple robots coordinate to perceive target regions of interest with minimal cost, considering both motion and perception cost, and quickly finding paths for all robots that minimize the total cost.

Chapter 8

Related Work

The problem of planning for single or multiple robots has been studied for a long time. When considering the specificity of each robot and determining the feasibility of tasks for each robot, one approach is to consider the physical constraints while planning. However, that approach can be costly if there are many targets in similar positions, where most of the search effort is repeated each time.

Solutions focused on motion planning have been proposed, such as the probabilistic roadmap for path planning in high dimensional spaces [19], where a pre-processing technique is used to learn a graph of nodes of collision-free positions connected with edges of feasible paths. During online planning, the algorithm uses that graph to quickly search for solutions. Another example is the lattice graph [41], which is a discrete robot state representation that constrains the planning to feasible positions (avoiding obstacles) and feasible motion dynamics (differential constraints). Another solution that uses some kind of pre-processing was proposed based on experience graphs [40], where past experience is used to inform the planner, directing the heuristic search in motion planning not only towards the goal but also towards previous paths.

In this work, we contribute an efficient algorithm to determine the impact of a robot's physical characteristics in the robot's reachability in the world, both in terms of actuation and perception, and then use it to improve planning. Various robotics techniques use different map representations, e.g., occupancy grids, geometric landmarks, or topology. An example that combines multiple types is the manifold for map representation [17].

In this section, we present past work related to our approach of finding robot-dependent maps. We start by explaining morphological operations, a key component of our approach. Then we present related work used to determine visibility for perception tasks and methods used in similar perception planning problems. We then show some other works and background on heuristics for informed search that can be used for changing target positions and in dynamic environments. Finally, we show the relevant literature on multi-robot planning.

8.1 Visibility and Perception Planning

In other work it was proposed that robots maintain reachability and visibility information, both of a robot and a human partner in a shared workspace [31], used for human-robot interaction such as the robot can reason about what humans might be able to reach or see.

Visibility graphs, as graphs of intervisible locations, have nodes that represent point locations, and edges that represent visible connection between them [22]. Motion planning can use visibility graphs, where first the visibility graph is constructed, and then used in planning to find the shortest path with straight lines (except at the vertices of the obstacles, where it may turn). The art gallery problem is also related to the concept of visibility graphs and visibility decompositions. This visibility transformation can also be used for patrolling. However, most of these problems assume a vectorial map of obstacles, so visibility can easily be calculated using ray casting at the extremes of lines with analytical calculations. However, in our work we assume robots build with SLAM and update maps in 2D discretized grids, making it essential to have methods that reason about visibility in grid maps.

Another class of problems for visibility is the inspection problem. A neural network approach was used to solve the NP-hard Watchman Routing Problem, determining a path that can sense multiple targets. A fast method was proposed to answer visibility queries [13], and the approach has been extended to 3D [18]. However, in this works, queries ask for visibility from one specific point, while in our work we aim at finding the overall visibility of a robot from any position reachable from its initial location.

Many robotic applications consider perception separately from planning, with both being computed interleaved [45]. It has been used for tasks as varied as SLAM [6], object recognition [11], or even active vision [48], where for all these cases perception is just used as a means to achieve some goal.

However, recently perception got a more active role in planning, where robots plan on how to perceive. An example is object detection, where the next moves of the robot should be planned to maximize the likelihood of accurate object detection and classification [43, 50]. In [10], probabilistic active perception is planned for scene modeling in realistic environments, with arbitrary object positions. They introduce a POMDP technique which reasons about model and state transition uncertainties to improve detection results and deal with occlusion problems.

The problem of planning for both the perception and the traveling cost has also been studied [53]. In that work, the planner finds a sequence of sensing actions with the minimum overall cost to inspect objects in a known workspace.

8.2 Motion Planning

One example of motion planning in robotics is coverage path planning, which finds a path that passes a detector over all points in an environment.

One possible way to achieve full coverage is by using the morse decompositions [1]. In this work, the robot can simultaneously explore and cover the space while incrementally constructing the graph with the cell decomposition, by sensing the critical points of all cells. Another work extends this approach to consider coverage with a finite detector range that goes beyond the robot [2]. This algorithm divides space into two areas, the open spaces where the robot can use the full range of its detector, and the narrow spaces where obstacles are in the detector range. For the first, the morse cell decomposition technique is used, and then a simple motion can be used to guarantee total coverage of open spaces. For the latter, the generalized Voronoi diagram is used to find a skeleton that, if followed, guarantees coverage with minimal motion. The vast and narrow spaces could also be found using morphological operations on the configuration space.

In the rest of this section, we focus on heuristic search methods for motion planning in graphs and multi-robot planning.

8.2.1 Heuristic Search for Motion Planning

A* is a graph search algorithm that finds the lowest cost path from a given initial node to a goal node. In traditional motion planning, the total cost at each node is estimated by summing the past cost from the starting position S to the current node n, with a heuristic of the cost from n to the goal position G.

$$f(n) = g(S, n) + h(n, G) \tag{8.1}$$

If the heuristic used is admissible, i.e., always less or equal than the true value, then the path returned is guaranteed to be optimal. Therefore, the natural choice for the heuristic is just the Euclidean distance between the current node and the goal, without considering any obstacles.

$$h(n, T) = ||n - G|| \tag{8.2}$$

There are many extensions to heuristic search to deal with different problems. In order to consider cost changes in the environment graph, Lifelong Planning A* [21] provides a technique using a heuristic search that updates the optimal path based on those changes, while minimizing the amount of extra nodes expansion (compared to recomputing A* from scratch). That technique was then extended to deal with changing position from the robot (the updates in the graph can be thought of perception updates from a moving robot) [20]. For that purpose, the core Lifelong Planning A* was used backward, starting the search from the goal position (which is constant) to the robot position which is being updated. This technique, D* lite, uses the concept of local inconsistency to guarantee optimal solutions.

Anytime planning is a different approach where intermediate sub-optimal solutions are computed and improved until the planning time is used completely, which can end with the path being improved until being optimal. In ARA* [23], for each intermediate step, there are guarantees on the sub-optimality bound. In AD* [24], the previous concepts of anytime planning are merged with the D* lite algorithm to obtain a technique for both incremental and anytime planning.

8.2.2 Motion Planning with Multiple Objectives

One of the novelties of our work is the introduction of a method that optimizes paths considering two different objectives, namely minimizing both motion and perception costs.

There have also been other works with also consider and optimize for two or more mostly opposing objectives, such as exploration of unknown territory. For the exploration problem, one can have one or multiple robots, and the overall goal is usually to explore the entire environment while minimizing the motion cost. The conflicting goals are, 1) minimizing motion, and 2) maximizing the exploration of unknown regions. These two objectives are quite the opposite, as exploring unknown regions usually requires a lot of motion.

One method in particular, frontier-based exploration, introduces the concept of frontiers to prioritize exploration [47]. Obstacles and unknown territory bound the known regions, and from those boundaries, it is possible to create multiple frontiers. Exploration is then simply a problem of estimating the information gain and motion cost for each frontier and then choosing to explore the one with the best trade-off between motion cost and information gain.

There are similarities between this robot exploration problem and the algorithm we introduced for perception planning, where we find paths that minimize both motion cost the cost of perceiving a target. An alternative to our approach could be to consider multiple final destinations and then choose the one with the best trade-off between motion and perception cost. The sensor cost function we introduce could also be used here to estimate the perception cost from the set of possible final destination and the perception target. However, our perception planning contribution has several advantages. First, we do not need to find a priori a set of destination positions. Moreover, when using a set of destination positions, and estimating the perception cost from the last position, it is possible to have solutions where there are points in the middle of the path with a lower perception cost than the final position. Finally, our algorithm can determine a final position in almost a continuous space (except from running in a 2D grid, but we assume the grid has a good resolution), while the frontier-based exploration is constrained to a set of fixed final path positions. If we increase the number of possible destinations, it quickly becomes very computationally expensive and makes it reasonable to choose an informed search algorithm that scales better with the size of the environment due to search being directed towards the perception target.

8.2.3 Multi-Robot Planning

There have been many works in the field of multi-robot path planning. One approach considers regions of known interaction, where explicit coordination is needed [28]. Therefore, they propose a decentralized approach with sparse interactions where for most of the time robots act independently, while learning which regions require interactions, where planning is treated differently. Other approaches plan independently for each robot, and if the algorithm finds interactions, robots are coupled in sets where planning occurs in the joint space [52]. Therefore, the size of the problem is reduced as much as possible, and the dimensionality is only increased whenever needed if interactions occur.

Regarding the use of different capabilities, not only terrestrial mobile robots of different sizes and characteristics but also aerial robots have been used in heterogeneous robot systems for search missions [9, 27]. In these exampless they use the different sensing capabilities of each one to attribute and coordinate tasks. However, neither of these works focuses on determining the feasibility of goals for each robot, and the coordination technique does not consider an objective representation of the team heterogeneity.

The works in multi-robot coordination commonly assume homogeneous robots, and when there is heterogeneity, the strategies are usually hard-coded for the specific kind of diversity found in the team. In our work, the goal is to have an objective representation of task feasibility for each robot, depending on the robot's physical characteristics, and then plan accordingly, without any predefined strategy based on the robot's differences.

It is also possible to use communication between independent robots, such as dynamic changes in the environment, so robots better accomplish their tasks[7]. It is possible to have an efficient replanning for the tasks that become unfeasible when using the relationships between dynamic conditions and the scheduling decisions.

Another problem is multi-robot coordination for coverage, where a wide range of planning methodologies has been proposed [3, 4]. However, these solutions often assume visibility information is trivial, with simplistic models for robot shape, motion, and perception.

Finally, in the multiple robot surveillance problems, multiple points have to be allocated to each robot, with time limits to finish the surveillance task. An algorithm was proposed using graph searches to solve the problem optimally [49]. Three levels of graphs are used, where the top one does the goal assignment, the middle plans opportunistically to include new goals in each robot's path, and the lower level considers obstacles. Thus this approach combines the orienteering problem and the dynamic constraints of the robots at the same time.

Perception got a more active role in planning recently. An example is object detection, where the next moves of the robot should be planned to maximize the likelihood of correct object detection and classification [43].

Planning sequences of perception points to cover regularly all interest points in the environment is also relevant for multi-robot patrolling [42], where a probabilistic strategy was used for a team of agents to learn and adapt their moves to the state of the system at the time, using Bayesian decision rules and distributed intelligence. When patrolling a given site, each agent evaluates the context and adopts a reward-based learning technique that influences future moves.

Other relevant work focuses on the sensing horizon, and how to opportunistically plan navigation and view planning strategy to anticipate obstacles with look-ahead sensing [30]. Candidate positions are considered based on the possibility of anticipating obstacles and used as waypoints. In the same topic, it has also been shown that perception planning and path planning can be solved together [15], selecting the most relevant perception tasks depending on the current goal of the robot, thus successfully solving navigation and exploration tasks together.

On the other hand, in our proposed solution for multi-robot perception coverage, we cannot use the perception planning technique directly because we have to deal with multiple robots who should split the overall perception task. The proposed solution is computationally heavy, as we run PA* for every pair of robot and perception target in order to determine a set of waypoints. After having the waypoints, the algorithm uses a heuristic allocation method to create paths for the robots. However, the main advantage and novelty of our contribution is that we provide estimates on the perception cost, which are different for every robot and depend on their sensor model. The information on perception cost enables us to create a solution that can effectively trade-off motion and perception costs, and consider robot heterogeneity and their unique characteristics when planning their paths. Regarding the computation effort, we also provided an alternative method based on Robot-Dependent Maps that can compute much faster the waypoints used to plan paths for multiple robots in a perception planning problem.

Chapter 9

Conclusions and Future Work

In this chapter, we summarize the thesis contributions and discuss possible directions for future work.

9.1 Contributions

The thesis main contributions are the following:

Robot-Dependent Maps represent the coverage and perception reachability of a robot moving in a structured environment, which depends on the robot's physical characteristics. Our contributed technique receives as input a binary image representing the environment floor plan, the robot's footprint and initial position, and the sensor characteristics such as field of view and maximum range. Given those inputs, our algorithm transforms the environment original map, producing new maps that represent the robot feasibility in terms of coverage and perception tasks, i.e., the accessible regions of the map for coverage or perception. The computation of such maps has a low cost that amortizes over multiple search instances when planning task execution. Our technique, based on the morphological closing operation, first determines the actuation map of a robot, which represents its coverage reachability, depending on the robot's initial position. In order to find the map that represents the perception capabilities of the robot, we introduce the concept of *critical points*, as a smart sampling of the navigable space that allows our algorithm to incrementally add regions of visibility to the actuation map, maintaining overall efficiency while increasing the approximation quality of the visibility map. We also extended the described approach with morphological operations with a multi-layer representation to deal with any-shape robots and different sensor models. Our algorithm results in a speedup of the computation of those reachabilities when comparing to a brute-force approach, at the cost of returning an approximation instead of the complete solution. Our experiments show that an almost complete solution can be obtained with very low computation effort.

Informed Search for Perception Planning is a heuristic search for path planning of perception tasks, i.e., perception planning. The goal is to find the optimal motion path and perception location for a robot that has to sense a specific target position in a 2D map. The optimal path solution minimizes not only the motion cost but the perception cost as well. The perception cost models the decreasing quality of measurements with increasing distance, accounting for different inaccuracies and uncertainty of sensors. The algorithm considers the overall cost of a path solution to be the sum of motion cost with perception costs, using a trade-off parameter to change the weight of each of these components. Changes in the trade-off parameter may generate different path solutions. The search algorithm minimizes the number of ray casting operations. We prove that, under certain assumptions and accurate sensor and motion models, our technique also guarantees optimal path solutions. This thesis also provides examples of perception cost functions, presenting the respective search heuristics. We theoretically prove those heuristics obey the triangle inequality, thus being both consistent and admissible heuristics. As the heuristics do not overestimate the real cost of perceiving a 2D target position, we demonstrate the optimality of the solutions generated by our technique.

Robot-Dependent Heuristics in Perception Planning is a technique that uses information extracted from robot-dependent maps to improve the efficiency of node expansion in heuristic search for perception planning. When computing the robot-dependent maps, the algorithm can retrieve additional data for targets in unreachable regions. The extracted information comes from critical points, which are the closest reachable positions from where some targets can be perceived, thus bringing knowledge on the minimum motion and perception distance. We build dominant heuristics using the information from robot-dependent maps. Again, the consistency and admissibility of the new heuristics are proven theoretically, as well as their dominance over the original heuristic. Through experiments, we demonstrate that using those dominant heuristics in our search algorithm results in fewer node expansion and fewer ray casting operations. As a result, there is a speedup in the search process of finding a path for a robot that needs to perceive a target location.

Pre-Processing Phase in Multi-Robot Coverage Task Planning represents the combination of robot-dependent maps with automated planning, which we used here for the problem of coverage task planning with a team of multiple heterogeneous robots. For planning approaches that deal with multi-robot problems by distributing coverage tasks among the multiple robots, task allocation is mandatory and requires knowledge on the cost of each robot executing each coverage task. For an effective task allocation, heuristic information is needed to estimate the execution cost of each robot-task pair. We contribute a pre-processing phase for the coverage task planning problem that uses robot-dependent maps to scale for larger problems with many robots and a large number of coverage tasks. With the pre-processing step, the feasibility of each robot-task pair can be easily obtained from the robot-dependent maps. Thus the information from the robot-dependent maps saves on computation effort by allowing the planner to only compute the heuristic cost for feasible tasks. Moreover, an alternative and faster cost estimation procedure is also possible, based on robot-dependent maps. We show experimentally that using our contributed pre-processing phase it is possible to improve the execution time of standard task allocation-based multi-robot planner. That improvement is a result of reducing the time spent on task assignment while maintaining the capability to find suitable solutions for the coverage problem. As expected, increasing the number of robots has an impact on performance, because the number of planning sub-problems that we need to solve increases linearly with the number of robots. However, the same is true even if we did not use our pre-processing step. As we address the main bottleneck in terms of planning time, the heuristic calculation for goal assignment, our approach is faster and is more scalable in terms of number of robots, as the planning time increases at a lower rate compared to the situation without any pre-processing step.

Multi-Region and Multi-Robot Perception Planning is an algorithm that finds paths for heterogeneous robots that need to execute multiple perception tasks. Multiple target regions need to be perceived by any robot, where each region must be fully "covered" by any of the robots' sensors. The optimization goal is not only to minimize the traveled distance but also to minimize both the perception cost. The perception cost accounts for the quality of measurements and may change for different sensors. For this contribution, multiple perception tasks are created from the target regions, and individual perception planning instances are run for every robot-task pair, generating multiple final perception positions. The algorithm clusters those positions from where the robots perceive the target regions in a series of waypoints, which represent the locations robots have to move to in order to optimally perceive the target regions. The results from the individual perception planning instances are also used to calculate heuristic values for each waypoint. The waypoint heuristic value estimates both the motion and perception cost associated with a robot moving to a particular waypoint to perceive some target regions. Finally, the algorithm incrementally assigns those waypoints to the robots' paths, allocating them as intermediate goal positions, thus generating a solution that covers all target regions while minimizing the overall motion cost and maximizing the perception quality.

Meeting Point Calculation for Delivery Services demonstrated how to use our perception planning technique for rendezvous calculation, in a scenario of deliveries from vehicles to users in a city. When a vehicle has to make a delivery for a user in a city, it usually drives directly to the passenger location, without ever suggesting the user to walk to some other location. With this contribution, we used the perception planning technique to calculate different rendezvous locations, from which the user can choose according to their preferences. We still consider vehicle heterogeneity with different reachability in the world, and again the algorithm optimizes not only for vehicle motion cost but also for the cost of the user walking to the calculated delivery location.

9.2 Future Work

In this thesis, we addressed the challenges of efficiently planning for coverage and perception tasks with multiple heterogeneous robots. We considered robot differences in terms of their geometrical properties, such as footprint, sensor range and field of view. In terms of planning, we explored algorithms that could deal both with the minimization of traveled distance and maximization of perception quality simultaneously, for single and multi-robot scenarios. There are multiple possible directions for future work, but we believe four are the most interesting. The first involves improving the approximation of the robot-dependent visibility map. The second consists in enabling incremental updates to the robot-dependent map. The third involves using robot-dependent maps for opportunistic planning in dynamic environments. Finally, the forth consists of experimenting with some of our contributions on real robots and vehicles that can perform perception tasks.

Better Robot-Dependent Visibility Map Approximation. Introducing the concept of critical points proved valuable to calculate good approximations of the true visibility map quickly. However, if one wants to use robot-dependents maps as a pre-processing phase to speed up multi-robot task planning, these maps need to provide complete information on the feasibility of robot-task pairs. For the multi-robot coverage task planning, it was straightforward to integrate the robot-dependent actuation map, as it was an exact calculation and provided complete information on task feasibility. On the other hand, the robot-dependent visibility map we contributed is only an approximation. It proved to be a very good approximation, useful when improving the heuristics of perception planning. Nonetheless, if one wants to use the visibility map as a pre-processing of a multi-robot task planning problem, the robot-dependent visibility map should be a complete solution. The main challenge is then to extend the concept of critical point, so it would be possible to incrementally add visibility from different positions until an exact or much better approximation is achieved. Another challenge would be to maintain the efficiency of such an algorithm, minimizing the number of new added critical points needed to still be able to generate visibility maps quickly.

Incremental Updates of Robot-Dependent Maps. Even though we contributed an algorithm to compute robot-dependent maps quickly, our technique computes the full map transformations whenever a change in the map is detected. For small environments and simple planning problems, such as single robot scenarios, the computation is fast enough to be computed during online execution. However, for bigger and more realistic grid maps, and when a centralized planner needs to compute robot-dependent maps for different robot configurations, recomputing the full transformation could be quite inefficient. This limitation could make it challenging to compute the transformations online when the environment is dynamic. A logical next step to address this question would be to compute updates to the robot-dependent maps incrementally as changes in the environment are detected during robot execution. The first challenge would be to identify how local changes in a map propagate in terms of our map transformation, from changes in the partial morphological closing operation to changes in the frontiers of reachability. A small change in the environment might render some regions unreachable, and it will be necessary to construct an efficient algorithm to detect and propagate changes in the map, through the various steps in the calculation of robot-dependent maps. A second challenge would be to detect efficiently in which cases the algorithm needs to rerun ray casting, which is essential for online execution since ray casting is a computationally expensive operation.

Opportunistic Replanning in Dynamic Environments. When thinking of dynamic environments for a robot during online execution, it would be interesting for the robot to learn the probabilistic behavior of some obstacles. During execution, if changes are detected in the environment around the robot, planning is usually rerun to account for the last detections. However, for difficult planning problems, such as multi-robot path planning, it can be expensive to replan for those changes, and therefore it would be interesting to be able to use robot-dependent maps to quickly determine the effect of those changes in terms of task feasibility, and avoid expensive replanning when possible. The same reasoning applies for unexpected openings robots observe during execution, in places accounted as closed while planning due to learning of obstacles from past observations. In order to have this opportunistic replanning based on robot-dependent maps, the main challenge is to be able to determine the impact of each obstacle on every task feasibility using robot-dependent maps, and reason about it without knowing the state of possible obstacles still unobserved along the robots’ paths.

Experiments with Real Robots Demonstrating Perception Planning. We considered in one chapter a more realistic use case of perception planning, but we look forward to experimenting with our algorithm on real mobile robots or autonomous cars when they are available, performing perception tasks. We also look forward to using robot-dependent maps in different types of robotic planning problems so that our pre-processing technique can speed-up the planning process in those scenarios.

Bibliography

[1] E. U. Acar and H. Choset, “Sensor-based coverage of unknown environments: Incremental construction of morse decompositions,” *The International Journal of Robotics Research*, vol. 21, no. 4, pp. 345–366, 2002. 112

[2] E. U. Acar, H. Choset, and J. Y. Lee, “Sensor-based coverage with extended range detectors,” *IEEE Transactions on Robotics*, vol. 22, no. 1, pp. 189–198, 2006. 112

[3] N. Agmon, N. Hazon, G. Kaminka *et al.*, “Constructing spanning trees for efficient multi-robot coverage,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2006, pp. 1698–1703. 114

[4] N. Agmon, N. Hazon, and G. A. Kaminka, “The giving tree: constructing trees for efficient offline and online multi-robot coverage,” *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2-4, pp. 143–168, 2008. 114

[5] D. Borrajo and S. Fernández, “MAPR and CMAP,” in *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15)*, Jerusalem (Israel), 2015. [Online]. Available: http://agents.fel.cvut.cz/codmap/results/CoDMAP15-proceedings.pdf 89

[6] L. Carlone, M. Ng, J. Du, B. Bona, and M. Indri, “Rao-blackwellized particle filters multi robot slam with unknown initial correspondences and limited communication,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2010, pp. 243–249. 111

[7] B. Coltin and M. Veloso, “Towards Replanning for Mobile Service Robots with Shared Information,” in *Proceedings of the AAMAS’10 Workshop on Autonomous Robots and Multirobot Systems (ARMS)*, 2013. 114

[8] M. Crosby, “ADP an agent decomposition planner,” *Proceedings of CoDMAP*, 2015. 89

[9] M. Dorigo *et al.*, “Swarmanoid: A novel concept for the study of heterogeneous robotic swarms,” *IEEE Robotics and Automation Magazine*, vol. 20, no. 4, pp. 60–71, 2013. 114

[10] R. Eidenberger and J. Scharinger, “Active perception and scene modeling by planning with probabilistic 6d object poses,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2010, pp. 1036–1043. 111

[11] S. Ekvall, P. Jensfelt, and D. Kragic, “Integrating active mobile robot object recognition and slam in natural environments,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2006, pp. 5792–5797. 111

[12] E. Fabrizi and A. Saffiotti, “Extracting topology-based maps from gridmaps,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 3. IEEE, 2000, pp. 2972–2978. 13

[13] J. Faigl, “Approximate solution of the multiple watchman routes problem with restricted visibility range.” *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 21, no. 10, pp. 1668–79, 2010. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/20837446 111

[14] M. Fox and D. Long, “PDDL2.1: An extension to PDDL for expressing temporal planning domains,” *JAIR*, 2003. 77, 80

[15] J. Gancet and S. Lacroix, “Pg2p: A perception-guided path planning approach for long range autonomous navigation in unknown natural environments,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3. IEEE, 2003, pp. 2992–2997. 115

[16] R. Haralick, S. Sternberg, and X. Zhuang, “Image analysis using mathematical morphology,” *IEEE transactions on pattern analysis and machine intelligence*, no. 4, pp. 532–550, 1987. 11

[17] A. Howard, “Multi-robot mapping using manifold representations,” in *Proceedings of the IEEE International Conference on Robotics and Automation.*, vol. 4. IEEE, 2004, pp. 4198–4203. 110

[18] P. Janousek and J. Faigl, “Speeding up coverage queries in 3D multi-goal path planning,” *Proceedings - IEEE International Conference on Robotics and Automation*, no. 1, pp. 5082–5087, 2013. 111

[19] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996. 110

[20] S. Koenig and M. Likhachev, “D* lite,” in *Eighteenth national conference on Artificial intelligence*. American Association for Artificial Intelligence, 2002, pp. 476–483. 113

[21] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," *Artificial Intelligence*, vol. 155, no. 1, pp. 93–146, 2004. 112

[22] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006. 111

[23] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in neural information processing systems*, 2004, pp. 767–774. 113

[24] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime Dynamic A*: An Anytime, Replanning Algorithm," in *ICAPS'05, the International Conference on Automated Planning and Scheduling*, vol. 5, 2005, pp. 262–271. 113

[25] D. V. Lu and W. D. Smart, "Towards more efficient navigation for robots and humans," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 1707–1713. 14

[26] N. Luis, T. Pereira, D. Borrajo, A. Moreira, S. Fernandez, and M. Veloso, "Optimal perception planning with informed heuristics constructed from visibility maps," *in submission to Journal of Intelligent & Robotic Systems*, 2018. 77, 79

[27] C. Luo, A. Espinosa, D. Pranantha, and A. De Gloria, "Multi-robot search and rescue team," in *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 296–301. 114

[28] F. S. Melo and M. Veloso, "Decentralized mdps with sparse interactions," *Artificial Intelligence*, vol. 175, no. 11, pp. 1757–1789, 2011. 114

[29] C. Muise, N. Lipovetzky, and M. Ramirez, "MAP-LAPKT: Omnipotent multi-agent planning via compilation to classical planning," in *Competition of Distributed and Multiagent Planners*, 2015. [Online]. Available: http://www.haz.ca/papers/muise_CoDMAP15.pdf 89

[30] B. Nabbe and M. Hebert, "Extending the path-planning horizon," *The International Journal of Robotics Research*, vol. 26, no. 10, pp. 997–1024, 2007. 115

[31] A. K. Pandey and R. Alami, "Mightability maps: A perceptual level decisional framework for co-operative and competitive human-robot interaction," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2010, pp. 5842–5848. 111

[32] T. Pereira, M. Veloso, and A. Moreira, "Visibility maps for any-shape robots," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 4205–4210. 11, 19

[33] T. Pereira, A. Moreira, and M. Veloso, "Improving heuristics of optimal perception planning using visibility maps," in *International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE, 2016, pp. 150–155. 43

[34] T. Pereira, M. Veloso, and A. Moreira, "Multi-robot planning using robot-dependent reachability maps," in *Robot 2015: Second Iberian Robotics Conference*. Springer, 2016, pp. 189–201. 11

[35] T. Pereira, M. M. Veloso, and A. P. Moreira, "Pa*: Optimal path planning for perception tasks." in *ECAI'16*, 2016, pp. 1740–1741. 26, 95

[36] T. Pereira, A. P. G. Moreira, and M. Veloso, "Multi-robot planning for perception of multiple regions of interest," in *Iberian Robotics conference*. Springer, 2017, pp. 275–286. 93

[37] T. Pereira, N. Luis, A. Moreira, D. Borrajo, M. Veloso, and S. Fernandez, "Heterogeneous multi-agent planning using actuation maps," in *IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE, 2018, pp. 219–224. 77

[38] T. Pereira, A. Moreira, and M. Veloso, "Optimal perception planning with informed heuristics constructed from visibility maps," *Journal of Intelligent & Robotic Systems*, vol. 93, no. 3-4, pp. 547–570, 2019. 43

[39] R. A. Peters, "A new algorithm for image noise reduction using mathematical morphology," *IEEE Transactions on Image Processing*, vol. 4, no. 5, pp. 554–568, 1995. 11

[40] M. Phillips, B. J. Cohen, S. Chitta, and M. Likhachev, "E-graphs: Bootstrapping planning with experience graphs." in *Robotics: Science and Systems*, vol. 5, no. 1, 2012. 110

[41] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009. 110

[42] D. Portugal and R. P. Rocha, "Cooperative multi-robot patrol with bayesian learning," *Autonomous Robots*, vol. 40, no. 5, pp. 929–953, 2016. 115

[43] C. Potthast and G. S. Sukhatme, "A probabilistic framework for next best view estimation in a cluttered environment," *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 148–164, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.jvcir.2013.07.006 111, 115

[44] S. Richter and M. Westphal, "The LAMA planner: Guiding cost-based anytime planning with landmarks," *JAIR*, 2010. 89

[45] R. B. Rusu, I. A. Şucan, B. Gerkey, S. Chitta, M. Beetz, and L. E. Kavraki, "Real-time perception-guided motion planning for a personal robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2009, pp. 4245–4252. 111

[46] F.-C. Shih and O. R. Mitchell, "A mathematical morphology approach to euclidean distance transformation," *IEEE Transactions on Image Processing*, vol. 1, no. 2, pp. 197–204, 1992. 12

[47] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes, "Coordination for multi-robot exploration and mapping," in *Aaai/Iaai*, 2000, pp. 852–858. 113

[48] E. Sommerlade and I. Reid, "Probabilistic surveillance with multiple active cameras," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2010, pp. 440–445. 111

[49] D. Thakur, M. Likhachev, J. Keller, V. Kumar, V. Dobrokhodov, K. Jones, J. Wurz, and I. Kaminer, "Planning for opportunistic surveillance with multiple robots," in *IROS'13, the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013. 115

[50] J. Velez, G. Hemann, A. Huang, I. Posner, and N. Roy, "Planning to Perceive: Exploiting Mobility for Robust Object Detection." *Icaps*, pp. 266–273, 2011. [Online]. Available: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS11/paper/download/2707@misc/3177 111

[51] V. Vidal, "YAHSP3 and YAHSP3-MT in the 8th International Planning Competition." in *8th International Planning Competition (IPC-2014)*, 2014. 89

[52] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 3260–3267. 114

[53] P. Wang, R. Krishnamurti, and K. Gupta, "View planning problem with combined view and traveling cost," in *ICRA'07, the IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 711–716. 111