Általános információk, a diplomaterv szerkezete

A diplomaterv szerkezete a BME Villamosmérnöki és Informatikai Karán:

1. Diplomaterv feladatkiírás
2. Címoldal
3. Tartalomjegyzék
4. A diplomatervező nyilatkozata az önálló munkáról és az elektronikus adatok kezeléséről
5. Tartalmi összefoglaló magyarul és angolul
6. Bevezetés: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása
7. A feladatkiírás pontosítása és részletes elemzése
8. Előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések
9. A tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása
10. A megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek
11. Esetleges köszönetnyilvánítások
12. Részletes és pontos irodalomjegyzék
13. Függelék(ek)

Felhasználható a következő oldaltól kezdődő LaTeXdiplomatervsablon dokumentum tartalma.

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmargóval készüljenek (mindenhol 2,5 cm, baloldalon 1 cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel, de ettől kismértékben el lehet térni, ill. más betűtípus használata is megengedett.

Minden oldalon – az első négy szerkezeti elem kivételével – szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellátni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellátni. A fejezeteket decimális aláosztással számozzuk, maximálisan 3 aláosztás mélységben (pl. 2.3.4.1.). Az ábrákat, táblázatokat és képleteket célszerű fejezetenként külön számozni (pl. 2.4. ábra, 4.2. táblázat vagy képletnél (3.2)). A fejezetcímeket igazítsuk balra, a normál szövegnél viszont használjunk sorkiegyenlítést. Az ábrákat, táblázatokat és a hozzájuk tartozó címet igazítsuk középre. A cím a jelölt rész alatt helyezkedjen el.

A képeket lehetőleg rajzoló programmal készítsék el, az egyenleteket egyenlet-szerkesztő segítségével írják le (A LaTeX ehhez kézenfekvő megoldásokat nyújt).

Az irodalomjegyzék szövegközi hivatkozása történhet sorszámozva (ez a preferált megoldás) vagy a Harvard-rendszerben (a szerző és az évszám megadásával). A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámozott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban mindig az eredeti nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben (a LaTeX-sablon a BibTeX segítségével mindezt automatikusan kezeli). Minden publikáció a szerzők után a következő adatok szerepelnek: folyóirat cikkeknél a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóiratok címét csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internetes hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal tulajdonosát és tartalmát (mivel a link egy idő után akár elérhetetlenné is válhat), valamint az elérés időpontját.

Fontos:

- A szakdolgozatkészítő / diplomatervező nyilatkozata (a jelen sablonban szereplő szövegtartalommal) kötelező előírás, Karunkon ennek hiányában a szakdolgozat/diplomaterv nem bírálható és nem védhető!
- Mind a dolgozat, mind a melléklet maximálisan 15 MB méretű lehet!

Jó munkát, sikeres szakdolgozatkészítést, ill. diplomatervezést kívánunk!

FELADATKIÍRÁS

A feladatkiírást a tanszéki adminisztrációban lehet átvenni, és a leadott munkába eredeti, tanszéki pecséttel ellátott és a tanszékvezető által aláírt lapot kell belefűzni (ezen oldal *helyett*, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell beleszerkeszteni ezt a feladatkiírást.

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Development of Navigation Methods in Dynamic Environment for an Intelligent Model Car

Bachelor's Thesis

Author
Soma Veszelovszki

Advisor
Domokos Kiss

November 21, 2019

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Veszelovszki Soma*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2019. november 21.

Veszelovszki Soma
hallgató

Kivonat

A diplomatervezési feladat egy, a tanszéken futó kutatási projekthez kapcsolódik, amelyben egy intelligens autó tesztplatform létrehozása a cél. A platform alapja egy csökkentett méretű (kb. 1:3 méretarányú) autómodell, melyet egy távirányítható játékautóból alakítunk át. A tesztplatform létrehozásának célja különböző navigációs algoritmusok fejlesztésének és valós környezetben történő tesztelésének elősegítése.

A sikeres navigáció és az ütközések elkerülése érdekében fontos, hogy a jármű valós időben detektálni tudja a környező objektumokat, és megfelelően reagáljon rájuk. Ennek elősegítése érdekében az autóplatformon elhelyezésre került egy-egy vízszintes síkban pásztázó lézeres távolságszkenner (lidar) a jármű elején és hátulján.

A feladat keretében ezekre a szenzorokra építve kell megvalósítani mozgó objektumok felismerését, azok méretének és sebességvektorának becslésével együtt. Fontos, hogy a jármű el tudja különíteni a statikus és a mozgó akadályokat egymástól. További feladat egy olyan akadályelkerülési módszer megvalósítása az autón, amely az objektumok mozgását is figyelembe véve hozza meg a megfelelő navigációs döntést minden időpillanatban.

A megvalósított algoritmusok működését mind szimulált, mind valós környezetben szükséges ellenőrizni. A valós tesztelés történhet a tanszéki járműplatformon, vagy egy kisméretű, egyedileg felépített modellautón is.

Abstract

The theme of the thesis is a subtask of a research project at the Department of Automation and Applied Informatics, with the purpose of designing an intelligent car testing platform. The platform is based on a decreased-size (1:3 scale) remote control car model, that has been modified to support self-driving program control. The platform aims to support the development of different navigation algorithms and helping the testing in real environment.

For a successful navigation and obstacle avoidance, the detection of the surrrounding objects and reacting accordingly are essential. For the purpose of detection, two horizontal distance scanning sensors (lidars) have been placed on the car - one on the front and one on the back.

Part of the task is to implement the detection of the moving obstacles and estimate their sizes and speed vectors. It is important for the car to be able to separate static and moving objects from each other. The other part is developing an obstacle-avoidance method, implemented as an application on the car, that takes the moving objects into consideration while making navigational decisions at every time step.

The implemented algorithms need to be tested both in simulation and in real environment, that may be executed on the department's vehicle platform or on a small-scale, custom-build model car.

Chapter 1

Introduction

Nowadays, self-driving cars gain more and more attention, both their technology and their effect on people's daily routines and their lives overall. Several articles are published every year about how these cars will change the way people commute to work, visit their friends or go on a family vacation. These articles often point out the decrease of the number of accidents, an optimized load of traffic and thus a reduced fuel consumption as the major advantages of this technology-to-come.

The release date of these cars, however, is still a matter of question. In 2015, Mark Fields, president and CEO of Ford at the time estimated their first fully autonomous car in 2020. 2 years later, at CES 2017 Nvidia announced that with the partnership of Audi they would develop a self-driving vehicle - also, in market by 2020. Both of these statements are considered too ambitious guesses today, as there is a high probability that we need to wait until at least 2025 for reliable fully autonomous vehicles to hit the roads. Claiming that there are no viable signs of these cars in traffic would be a false statement, as there are several companies who have been testing there vehicles on public roads for the last years, but these prototypes are very far from reliable products yet. The company that seems to be ahead of the competition in this race is Tesla. Their self-drving software is already in their products, but it still needs millions of hours of testing and the responsibility is still the driver's if an accident happens.

But why is this delay of release dates? One possible answer is that manufacturerers have the tendency to exaggerate when asked about new products, and thus the users' need and the other competitors development speed urged them to make such estimations they could not keep up with. Another theory is that the companies at that time didn't acknowledge how many hours and kilometers of testing is needed to finalize a self-driving product.

However, the spreading of autonomous vehicles is blocked by several legislational and technological obstacles. As this thesis describes an engineering problem an its solution, I will reflect on the technological blockers that both car manufacturers and other self-driving software developer companies need to face. Just to list some of these problems, we can name reliable object detecion, error insensitive, robust decision-making, fast, well-tuned physical control, and to meet the safety and quality requirements set by the market, determinant scheduling and redundancy throughout the whole software are also essential.

Chapter 2

Mapping

2.1 Method

This chapter decribes the process of building a separate static and a dynamic map (containing non-moving and moving obstacles, respectively) based on radial distance measurements (in this case provided by a LIDAR). Isolating the static and the moving obstacles from each other has its difficulties but also its advantages. First of all, the implemented local track planner calculates safer, more optimized paths if it is informed of both the static and the dynamic obstacles along the path. Secondly, SLAM (Simultaneous localization and mapping) algorithms work better if the their input contains only static objects in the space, because they build an internal map of the world. Passing detections of moving obstacles to a SLAM algorithm may lead to worse localization quality, as they may ruin this map.

As a first subtask of the mapping project, I checked if any implementation is avaiable already, that can handle dynamic objects. The only possible candidate was gmapping, which is a popular ROS package, used in a wide variety of applications that require map-building and localization. Its SLAM algorithm takes LIDAR measurements as its input and generates an occupancy grid (a 2D map) of the car's environment. By using this map it is able to make corrections to the car's odometry-based position and orientation, which is usually inaccurate. I tried out the package, and the result maps were promising, the generated map was insensitive to the car's longitudinal movements and its rotations. But unfortunately, gmapping's SLAM does not support dynamic objects. See Section 2.4.1 for further details. Therefore, gmapping could not be used as the producer of the static map. But that didn't mean it couln't be used for its second feature, localization. Note, that the mapping implementation I made is not a SLAM algorithm, it is not able to make corrections to the car's pose. So for that purpose, I still needed the help of gmapping, which proved to be very reliable at localization. But the static map-building needed to be implemented internally.

In order to create two disjunct maps, one static and one dynamic, the key element of the process is the separation of the moving and non-moving obstacles of the measured points. After determining these two disjunct set of points, the maps can be converted to any desired or required format. Static maps are usually published as occupancy grids, while dynamic obstacles need to present information about their speed vector. Occupancy grids do not group the grid points according to their probabilities, therefore they do not

know about the obstacles' borders and areas[1]. Dynamic obstacles however can be either represented as separate points with their own speed vectors, or groups of points, each groups having one speed vector. I chose the latter representation, thus publishing groups that contain a set of points (all the points, ideally) of the same obstacle. This way there is a one-to-one relationship between moving obstacles and groups. The separation and grouping methods of the mapping process is shown on diagram 2.1.

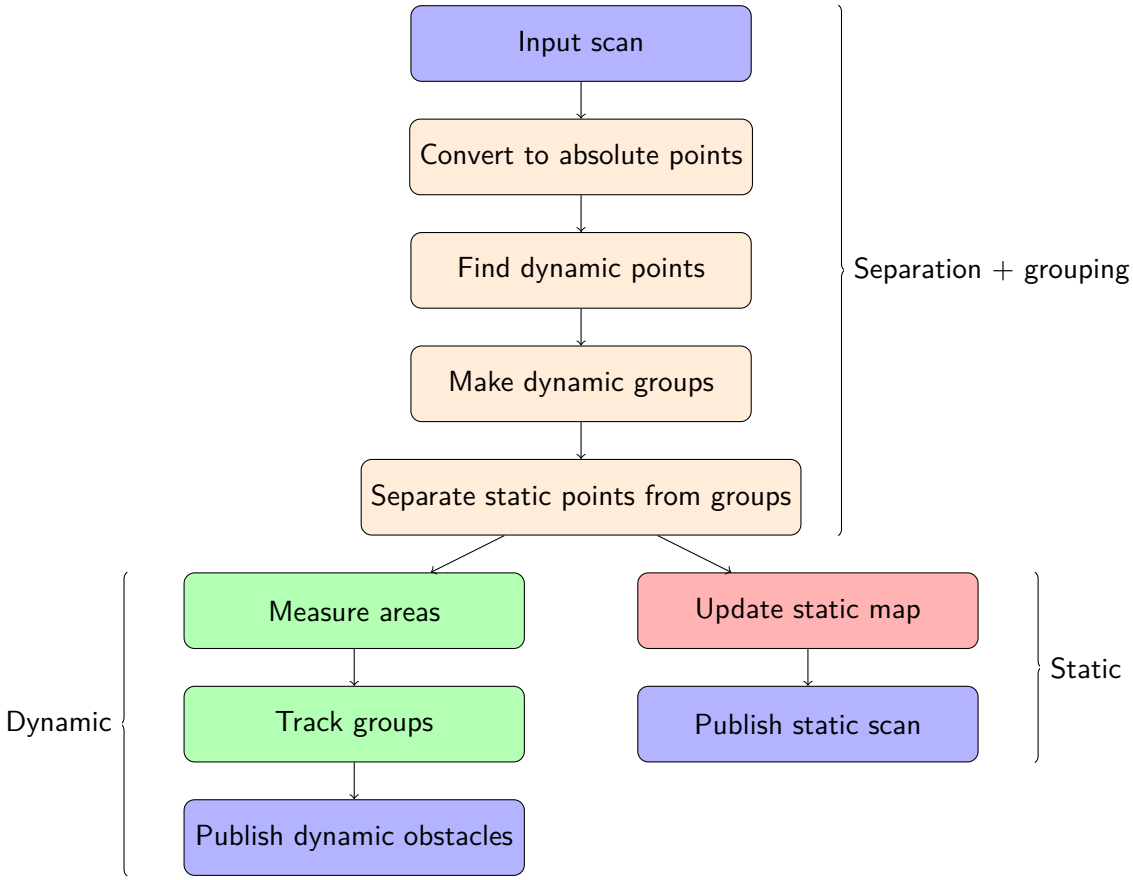


Figure 2.1: Mapping method

The diagram consists of 3 subgraphs - these are also marked on the diagram. The first, and most important is the separation and grouping of dynamic points. The second subgraph describes the additional calculations that need to be done for the dynamic obstacles, and consists the documentation for the message structure definining these dynamic obstacles. And the third one is about the static map that is built from the static points. The next sections are going to explain these subgraphs in detail.

2.2 Separation and grouping

This section describes the method of separating dynamic and static points of the input scan. This step is essential in the pipeline of creating two disjunct maps.

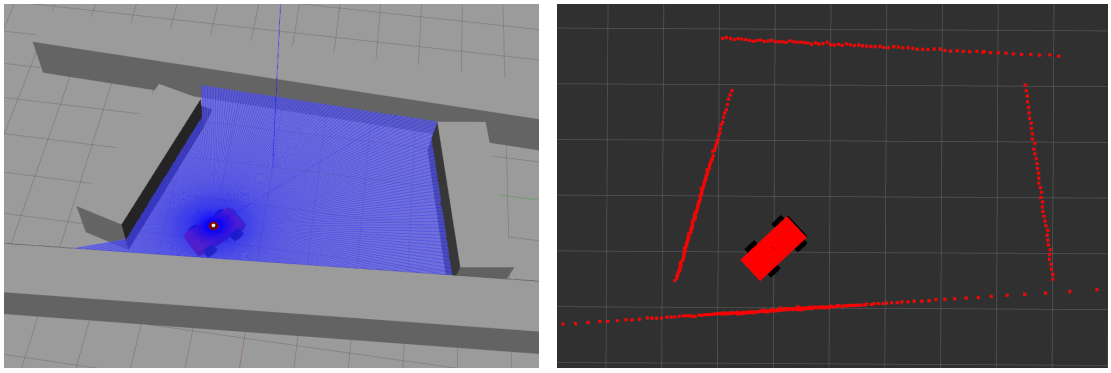
[1]In this project, 2D LIDARs were used, therefore the measured objects were seen as 2D shapes that have areas, not 3D objects that have volumes

2.2.1 Input scan

The input of the mapping algorithm is a 2D LIDAR scan, consisting of radial distance measurements. Two types of LIDARs were used in the project, RPLidar A1 and RPLidar A2. Both types have the following specifications:

Scan rate	10 Hz
Sample rate	8000 samples/sec
Distance resolution	0.2 centimeters
Angular resolution	1°
Detection range	12 meters

The devices proved to be reliable, and for a project of this volume, their frequencies, resolutions and ranges were adequate. Their output after each measurement sequence is an array of radial distances, that can be visualized easily using rviz.



(a) Gazebo simulation **(b)** rviz visualization

Figure 2.2: LIDAR scan

2.2.2 Absolute points

For static map-building, absolute points[2] are needed in the space, and static-dynamic point separation also uses absolute points, so firstly, these radial distances need to be transformed. However, the internal static map and the static-dynamic separation use different coordinate systems. In order to understand the need for this, let's take a look at figure 2.3.

As the figure shows, there are 3 coordinate frames that are important to the current case. The *odom* frame is the car odometry's coordinate system. It is calculated from values measured on the vehicle, such as servo position and speed. The *scanner* frame is the LIDAR's coordinate system. The transformation between *odom* and *scanner* is basically the pose of the LIDAR, relative to the car. The *map* frame is the output of gmapping's localization. Basically, gmapping takes the car odometry and the LIDAR scans as its input, and corrects the odometry using SLAM. As a result the difference between frames *map* and *odom* will increase with time.

The internal static map in my imlpementation uses this corrected *map* frame as the base for its points, so that its error is minimized. But unfortunately, the static-dynamic separation

[2] Absolute points are not relative to the car, but to a fix base point.

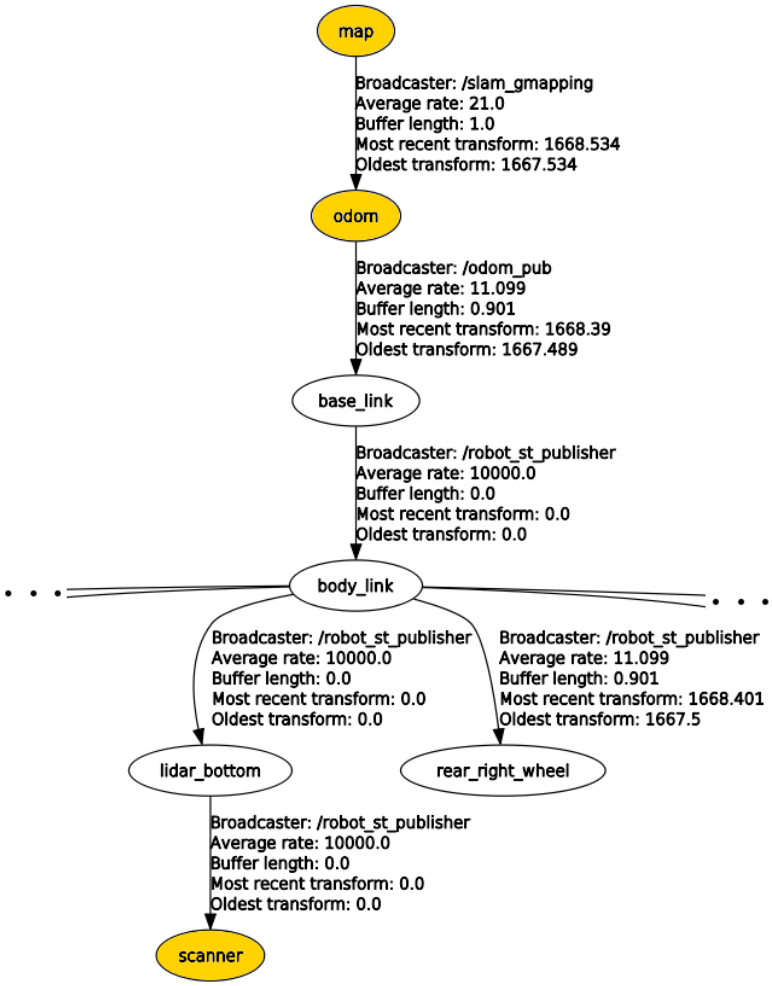


Figure 2.3: Coordinate frames

method cannot use this frame. The reason is that this frame does not get updated on every new scan, but with a much slower frequency. Therefore there are 'jumps' in the *map* frame, which would cause false dynamic point detections (see Section 2.2.3). To avoid this undesired situation, the static-dynamic separation uses the *odom* frame as its base, which is more continuous than the *map* frame.

Therefore, two transformations are needed for each input point. The transformations are calculated using tf, which maintains the relationship between coordinate frames in a tree structure (see figure 2.3) buffered in time, and makes the transformation of points, vectors, etc possible at any desired point in time.

2.2.3 Dynamic points

The next step is similar to a creating a subtraction image in image processing, where subtracting two images, taken from the same position but at a different time, results in an image that amplifies the movements between the snapshots. Previous methods[1] that I studied before starting my implementation are also based on this step. The aim of this algorithm unit is basically the same: selecting the points from the input that are likely to be part of a moving mass. This is done by finding the points among the current measurements that have not been present in the previous ones.

This step is best explainable in practice. Let's assume that the position and orientation of the LIDAR is fix, and one object (e.g. a car) in the detected area is moving. Figure 2.4 shows this situation. On the image, the pale contour represents the previous pose of the object, and the current state is blue.

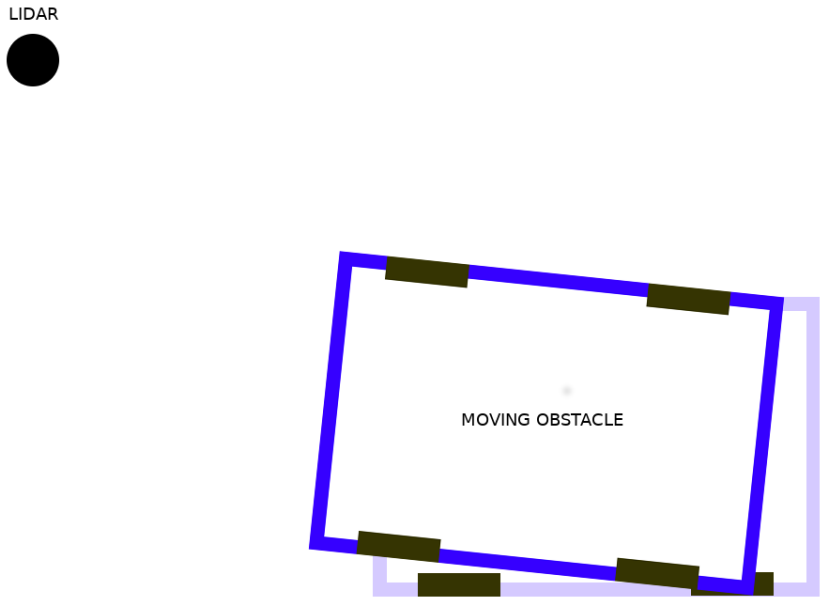


Figure 2.4: The obstacle is moving

Figure 2.5 shows how the LIDAR detects the moving object in two different timesnaps. I marked the points corresponding to the current position with red color, and the ones of the previous measurement are pale red. As it is suggested in the figure, the measurements are far from ideal, the detected points are noisy.

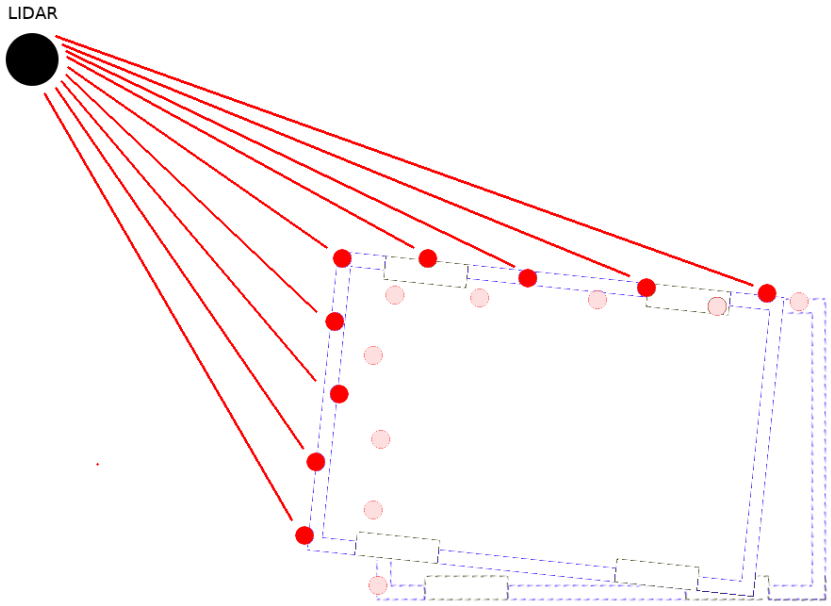


Figure 2.5: The detected points of the moving obstacle

To understand the method of dynamic point detection, I removed the LIDAR, its virtual rays and the obstacle's contour from the image, thus leaving leaving only the measured points of the two timesnap. The result, which is basically a time-buffered array of 2D

points, is presented on figure 2.6. The algorithm iterates through each point in the current measurement, and checks if any point in the previous measurements[3] is within its compliance radius[4].

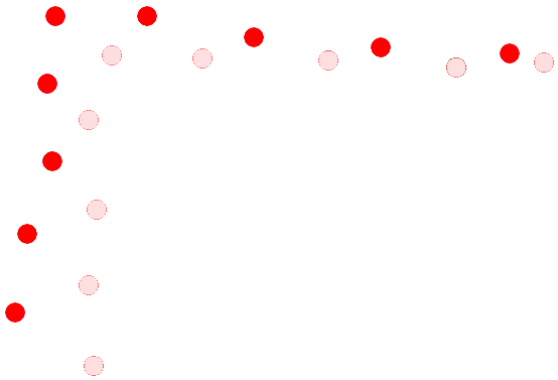


Figure 2.6: The detected points of the two timesnaps

The measured points with their compliance radiuses are presented in Figure 2.7. The possible dynamic points, that have no points from the previous measurement within their radius, are marked with blue, and their compliance radius with yellow. The possible static points are marked with red, along with their radiuses. Note, that these points are *possible* dynamic and static points. The final classification will be preceded be multiple filtering mechanisms and point grouping, but this is the base for finding dynamic objects.

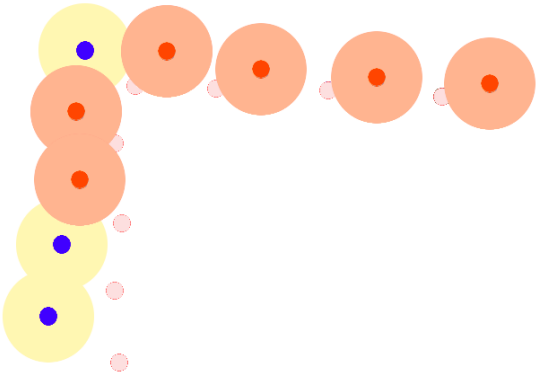


Figure 2.7: The compliance radiuses and the dynamic points

Several conclusions can be drew from Figure 2.7. The one, and probably most important is that with right parameterizing (number of 'look-up' measurements, compliance radius, etc) this method amplifies the positive[5] and negative[6] changes of the scans. The second remark is that the method does not detect all the points of a moving object. However, due to measurement noise, false detections may happen, and non-moving points may be marked as dynamic.

[3]The number of measurements to 'look up' is configurable.

[4]The compliance radius is a maximum allowed distance between measurements representing the same physical point but measured in different timesnaps. If the distance between two measurements is greater than this value, the measurements are assumed to represent different points of the space. The compliance radius is calculated for each measurement separately, and it is proportional to the distance of the LIDAR and the measured point.

[5]Previously a distant background was detected, but now a closer object appears.

[6]Previously a close object was detected, but now it disappears, and the distant background becomes visible.

Therefore, the result of this separation step needs to be filtered. I implemented a filtering step that only keeps those samples in the set of dynamic points, of which the left and right neighbour in the scan has also been marked dynamic. It can be easily compared to erosion in image processing, which is able to remove peaks from the image. Let's take a look at Figure 2.8. The possible dynamic points are marked with blue. The filtering mechanism iterates through each of them, and check both neighbours. As it can be easily read from the figure, point A has two neighbours that have been marked static, therefore it will be removed from the set of dynamic points. However, points B and C both have dynamic neighbours (each other), so they will not be filtered out. This algorithm removes the single-size false detections from the result of the separation.

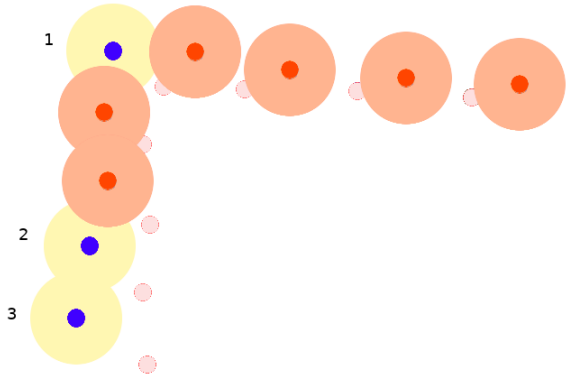


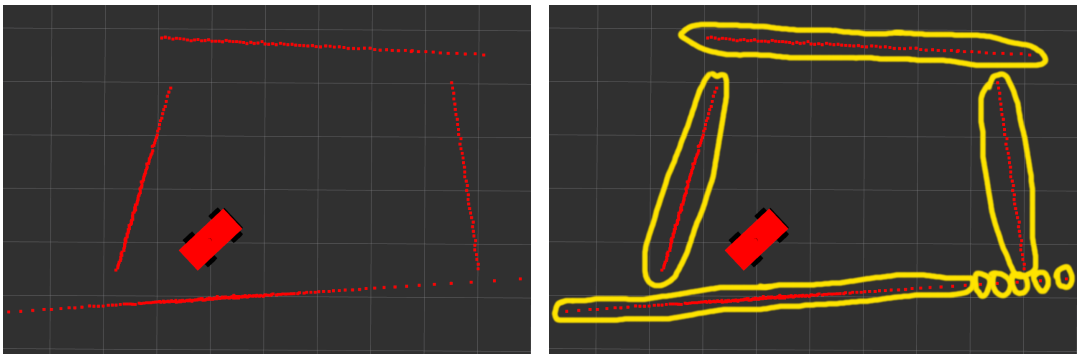
Figure 2.8: Point A will be filtered out

2.2.4 Grouping

After the filtering, we can safely assumed that the elements in the dynamic point set are in fact points of a moving obstacle. The next step is grouping the corresponding dynamic points, and also adding those points to these groups that were prevoiusly marked as static, but they are part of a moving group. For example in Figure 2.8, after the separation and filtering, only points B and C were marked dynamic, but actually, all the points in the image are points of the same object, that is moving. The grouping algorithm needs to be able to add these other points to the group as well.

The grouping algorithm is based on the supposition that coherent points are close to each other, while the distance between points of separate objects is larger. With a few exceptions, this statement holds its stand, by practice it proved to be a reliable base for grouping. The algorithm itself is quite simple, it separates the list of measurements (including the static and dynamic poitns as well) into subsets - groups. A new group is started when the distance between two adjacent points is larger than the permitted maximum within a group. This maximum is an empirical constant, with experiments of valid use cases. The result of the step is shown in 2.9b.

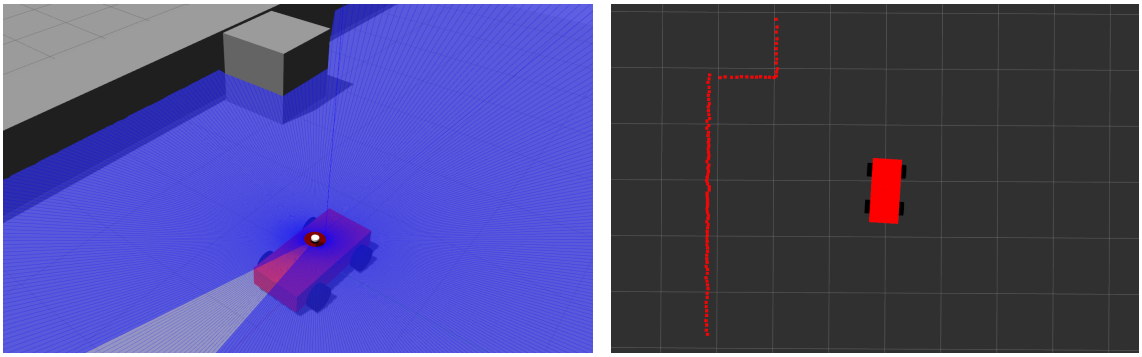
As it is clearly visible on the image, far points that are correspondent to the same object may be separated from each other, and grouped one by one - see the bottom right corner of the picture. However, the further an object is from the LIDAR, the less important it is for the local trajectory planner, which needs to avoid close obstacles. But still, they are not valid groups, so as the last step of the grouping mechanism, these single-point groups will be removed from the list of groups. But before doing that, there is still one special case that needs to be handled by the grouping unit - the *wall following car*. Let's imagine the situation represented in Figure 2.10. The detected obstacle (represented by the box)



(a) Gazebo simulation **(b)** Groups

Figure 2.9: Distance-based grouping

is moving alongside the wall, and within the maximum group point distance. Therefore, the grouping algorithm will join the groups. This wouldn't be a problem if the box wasn't moving, but as soon as it changes its position, grouping it together with the wall would ruin the mass center and speed vector calculation, described in Section 2.3.2.



(a) Gazebo simulation **(b)** Lidar scan

Figure 2.10: The *wall following car*

To avoid this problem, the algorithm detects if an obstacle is unusually large, and cuts off long 'tails' from the group. Figure Figure 2.11 explains this step in practice.

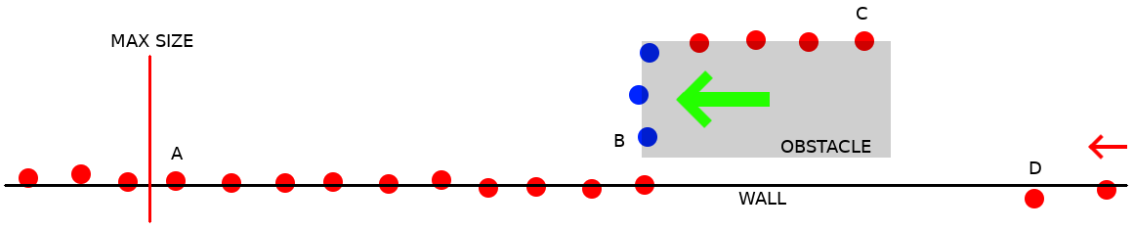


Figure 2.11: Points on the left of B will be cut off

Let's assume that the algorithm iterates through the points visualized in the figure from the right. The distance between points C and D is larger that the allowed maximum group point distance, therefore point C starts a new group. Until the program reaches point A, all the points are added to this group, because the distance between adjacent points is always lower than the maximum group point distance, but after point A, the algorithm detects that the group is larger than the permitted limit. So the iteration turns

around, and all the points are getting removed from the group, until the first dynamic point is reached - in the current example this would be point B. As a result, only the points between (and including) point B and point C will be grouped together.

As an undesired side-effect, this step may also remove some of those points from the group that are in fact parts of the moving object but were marked as static, because the removal only stops at the first *dynamic* point.

2.2.5 Separation of static points and dynamic groups

The previous step has successfully sorted the measured points into groups, but it is undecided yet if these groups are moving or not. This decision is made by checking the number of dynamic points in each group, which must exceed the required minimum in order to the group being marked as dynamic.

This was the last step of the separation of the groups, from this point, static points and dynamic groups will be handled differently.

2.3 Dynamic obstacles

After the dynamic groups have been separated from the static points, additional information needs to be calculated for them, so that the local track planner algorithm can use their data for its obstacle-avoidance feature. 3 pieces of information are needed for each object, its position, size and speed vector. All of these are calculated from the object's mass center, which is the average of the group points. This mass center is considered to be the object's position in space.

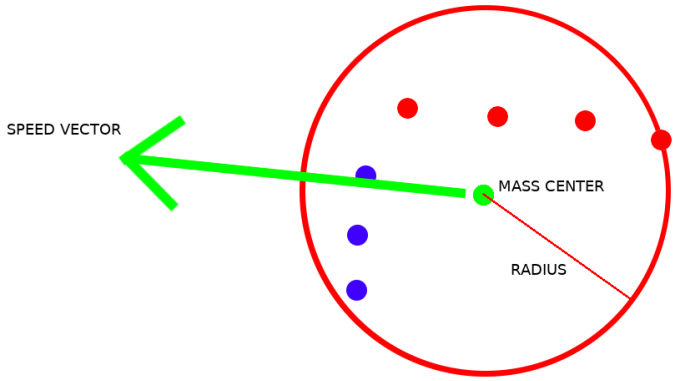


Figure 2.12: The calculated information for dynamic obstacles

2.3.1 Areas

First, the groups are substituted by bounding circles. The circle's radius is the distance between the farthest group point and the mass center. This simplification results in easier path calculation in the local planner, but also drops every information known about the obstacle's shape. For long objects, for instance, this method provides a very unrealistic image of the object, because the bounding circle is much larger than the obstacle itself. But for ususal shapes such as cars, the loss that the representation causes is less than the processing time we save with the simplification.

2.3.2 Tracking

Speed vectors are calculated from the change of the mass center between measurements. But in order to be able to check the mass center of a group in any previous measurement, object tracking is needed. The tracking is done by estimating each obstacles' current position based on their previous positions and speed vectors, and finding the best fit among the current groups. Filtering is also needed for such false detections when a previously detected object is not recognized for a few cycles. Without filtering, these obstacles would disappear from the list of groups for a period of time and then they would reappear, but tracking would fail. Therefore the algorithm keeps these objects alive for a given number of measurement periods, updating their positions with their last known speed vector in every cycle.

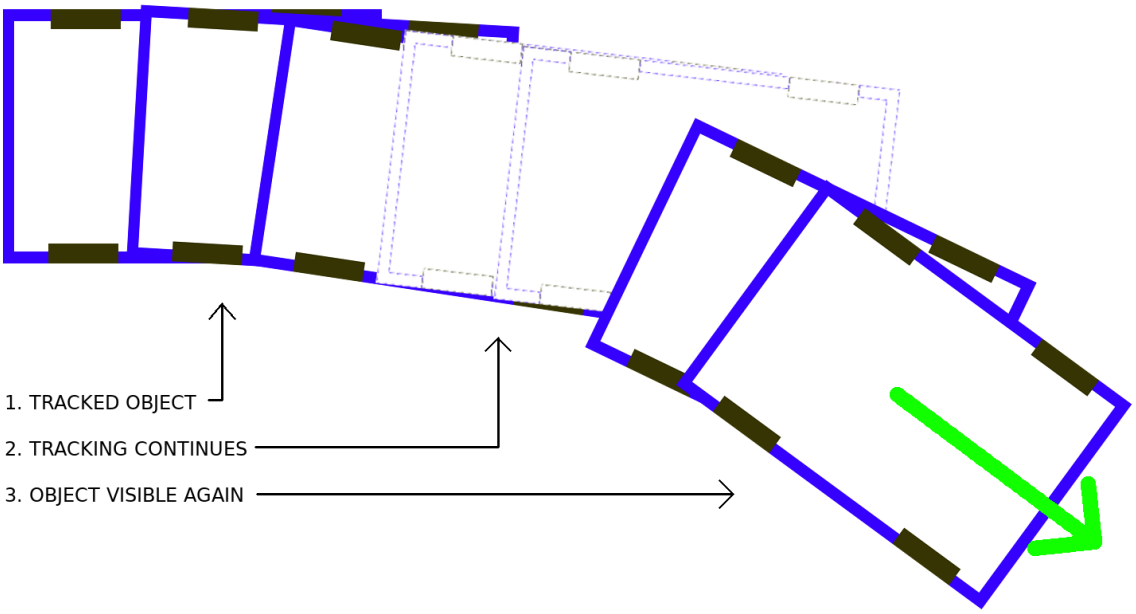


Figure 2.13: Object tracking

After the objects have been tracked, their speed vectors can be calculated. A filter is applied here, as well, for smoother obstacle paths.

2.3.3 Publishing dynamic obstacles

Dynamic obstacles - as outputs of the mapping node - are published on several topics, responsible for containing information for either further processing or visualization. The standard dynamic object structure in which moving objects are published to the local planner is not defined yet. The visualization messages, however, are implemented and in active use.

The published messages contain *visualization_msgs::Marker* instances with either *SPHERE* or *ARROW* instances with either *SPHERE* or *ARROW* type, thus representing the areas and speed vectors of the objects.

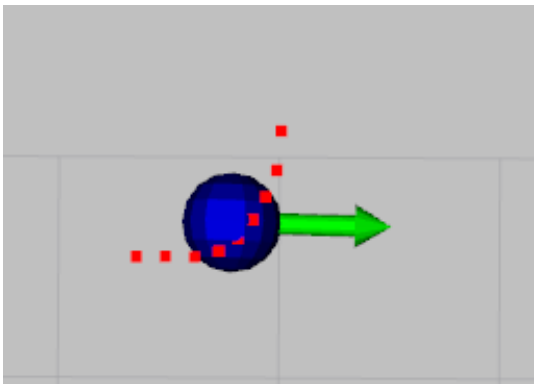


Figure 2.14: Visualizing moving objects

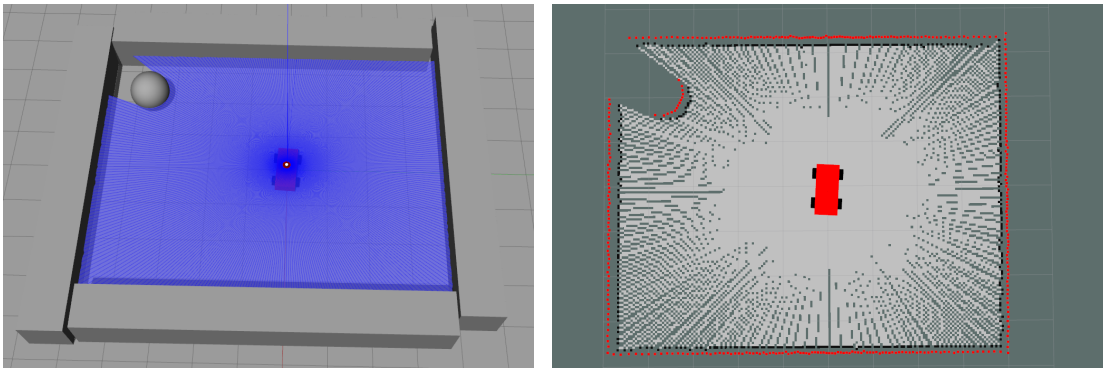
2.4 Static points

Static points used for two separate purposes. The first one is building a static map, the second is localization with the help of gmapping.

2.4.1 Static map

After separating the dynamic groups from the scan points, only the static points are left. These points do not change their positions with time[7], so they can be added to the static map.

For static map-building, gmapping has already been mentioned as a candidate, but unfortunately, its algorithm does not recognize changes in the environment. In practice, if an object has been detected and placed in its map, it will not be erased from there, even after the object has moved away. Figures 2.15 and 2.16 demonstrate the problem.



(a) Gazebo simulation **(b)** Map created by gmapping

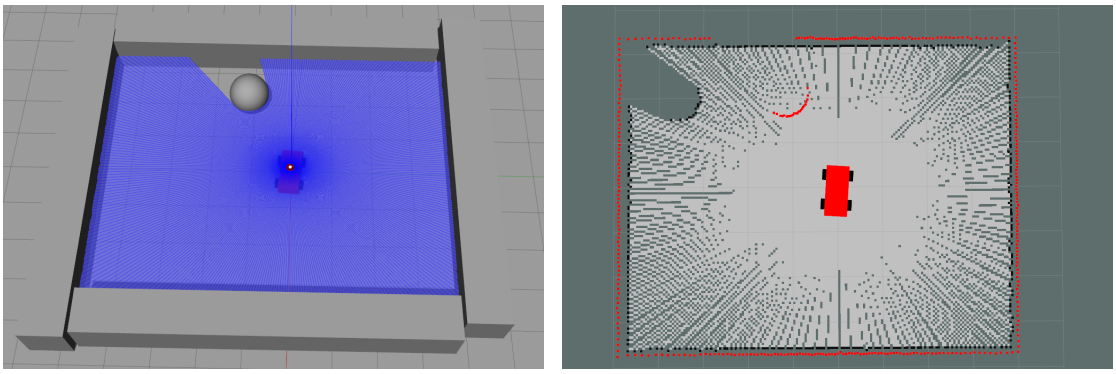
Figure 2.15: Initial state - ball is still

Figure 2.15 shows a situation where the ball is standing still. The mapping algorithm of gmapping successfully creates a map (in the form of an occupancy grid[8]) that marks the

[7]Except those obstacles that start moving only after they have already been detected as sets of static points. But these objects do not ruin the quality of the map, either.

[8]An occupancy grid represents a map in an evenly spaced field of probability values representing if the grid points are occupied by an obstacle

place of the ball as occupied. However, when the ball starts moving (see Figure 2.16), the map does not change, because the algorithm cannot handle moving obstacles.



(a) Gazebo simulation **(b)** Map created by gmapping

Figure 2.16: Moving state - ball has changed position

Due to the above problem, static map-building needed to be implemented internally, which is basically a 2D occupancy grid. The grid points' possible values are the following:

Occupied (value: 100)
Unknown (value: 50)
Free (value: 0)

I chose the most straight-forward way of map-building - each static measurement introduces an occupied point and a free ray in the map. Figure 2.17 visualizes the situation. The green and red grids are the car's current position and the measured point.

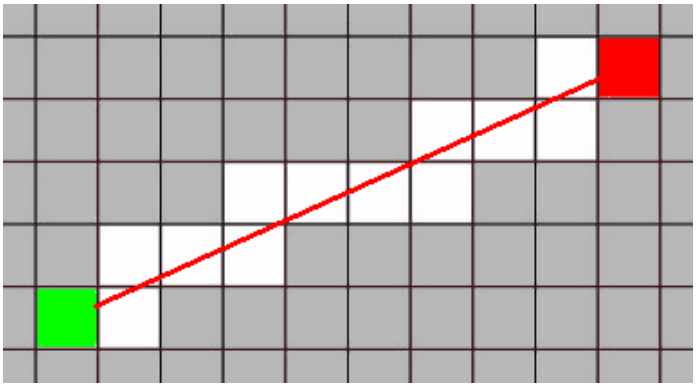


Figure 2.17: Introducing a free ray

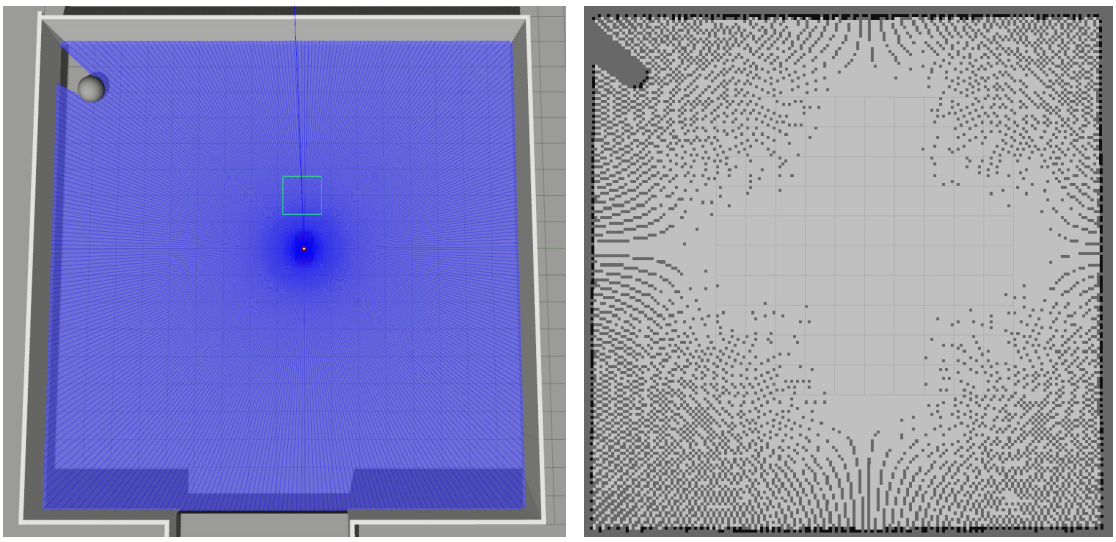
2.4.2 Publishing static scan

As I mentioned already, gmapping is not used as a primary map-builder node, but its localization is still useful, therefore LIDAR measurements still need to be published for it. As the algorithm already separated the static points from the dynamic ones, it is handy to publish only the static measurements to gmapping. So in this step, a new scan structure is created, but the dynamic points are not added to it, only the static ones.

2.5 Conclusion

As a conclusion of the mapping sub-project, I present an example with a valid use-case. The scenario is the following. The car is standing in a closed room, the distance from the walls are smaller than the LIDAR's range. There is one other object in the room, in the simulation, this is a ball, its size similar to the car's. The ball is standing in one place at first, but after a while, it start moving in a straight path with constant speed. Finally, the car starts moving as well - first in a straight path, then turning.

Figure 2.18 presents the initial situation of the demonstration. The left image shows the simulation - the car is standing in the middle of the room, the object in the top left corner is not moving yet. On the right image, the created static map is visible. The number of unknown (grey) grid points is large at this state of the simulation, it will dramatically decrease once the car starts moving. This effect is caused by the LIDAR's finite angular resolution and ray characteristics. The black grid points represent statically occupied positions. All the walls are marked static, and the ball as well.



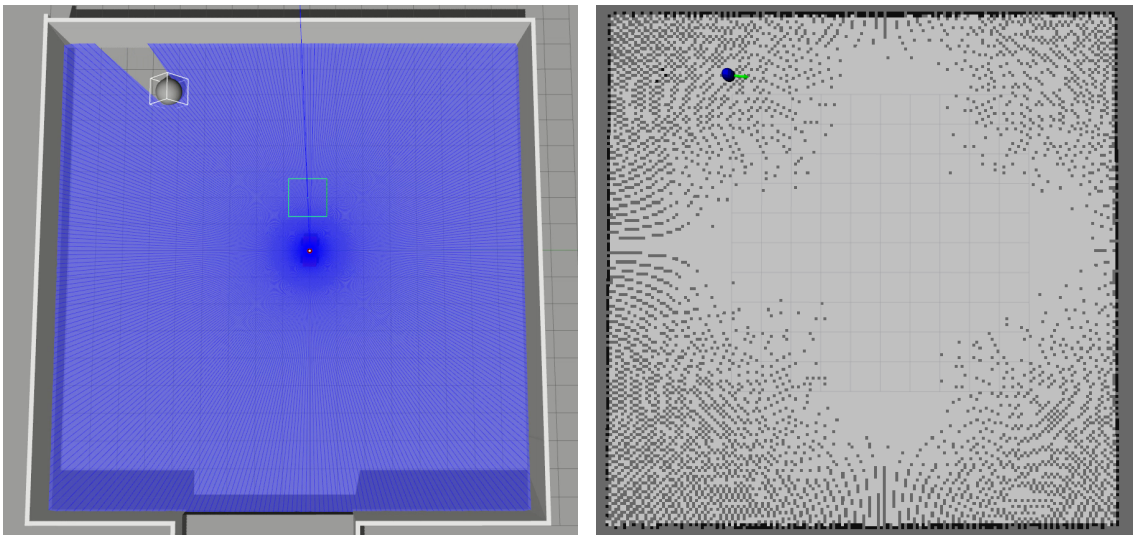
(a) Gazebo simulation **(b)** rviz visualization

Figure 2.18: Initial state - ball is still

Once the ball starts moving (Figure 2.19), it stops being marked as static, but is handled as a moving object. The static block at the top left corner has been removed, and the ball is displayed using the dynamic visualization toolset - a sphere and an arrow, representing the ball's area and speed vector.

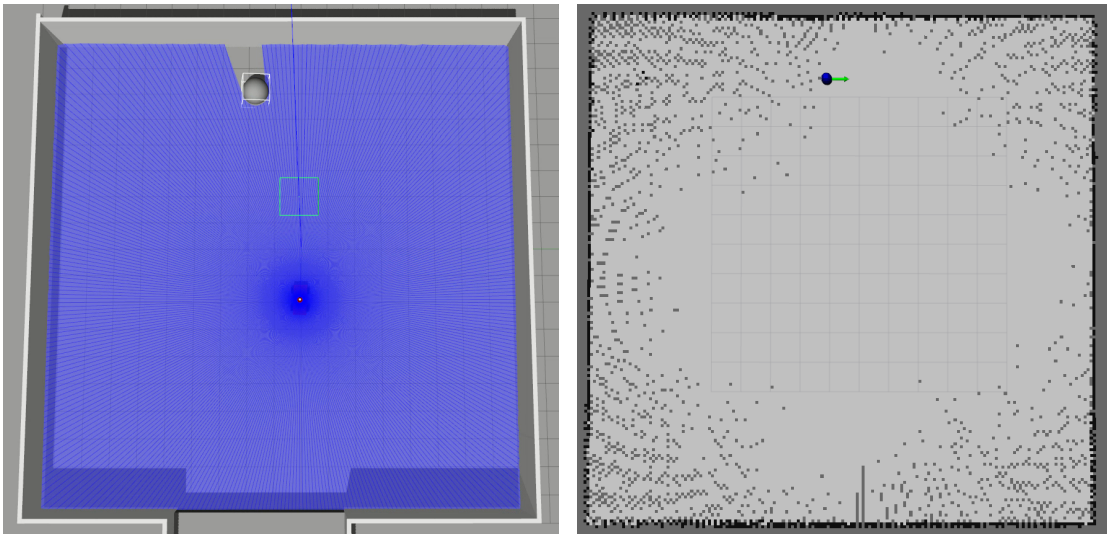
When the car starts moving, the map immediately starts to clear out. If we compare Figure 2.19 and Figure 2.20, it is clear that the latter contains less unknown grid points.

Due to the map implementation being very simple, and updating each map point without handling its history, the mapping node can only be used with certain limitations regarding obstacle speed and the car's angular velocity. But the main task for the mapping algorithm was not to create a static map with the best possible quality, but to separate the static points from the dynamic objects.



(a) Gazebo simulation **(b)** rviz visualization

Figure 2.19: The ball is moving



(a) Gazebo simulation **(b)** rviz visualization

Figure 2.20: The car is moving

Chapter 3

Motion planning

3.1 Existing local planner algorithms

After building a map of the static and dynamic objects surrounding the vehicle, the next step is motion planning, which consists of two sub-tasks - global trajectory planning and local obstacle avoidance. As a result of global planning, a trajectory is created that - without taking the moving obstacles into consideration - leads the car to the target configuration, without making it collide with any static objects in the way. Local obstacle avoidance takes this trajectory as its input, and updates the car's actuators (acceleration and steering) to follow this trajectory, while also preventing collisions with dynamic objects. As a part of my diploma project, I implemented the latter, a local obstacle avoidance algorithm, that relies on the previously created static and dynamic maps, and gets its input trajectory from a global planner node[1]. According to [4], there exists a wide range of local planner algorithms, and they can easily be grouped by their complexity:

- Velocity-based methods, most of the times combined with the dynamic window approach
- Predictive and probability-based methods
- Complex methods based on visual detection and AI
- Other methods

Out of these methods, velocity-based algorithms are by far the most popular, due to their being relatively easy to comprehend and implement and because of their low hardware requirements. As the mapping node and the local planner that I designed need to run on a processor with limited resources, I also chose a velocity-based obstacle avoidance method, using velocity obstacles, with dynamic windowing.

3.2 Method

The planner algorithm I implemented uses velocity obstacles to determine which configurations of the car are safe. Velocity obstacle methods (descibed in [2] and in [3]) are

[1] Implementing a global motion planner was not part of my diploma project, but is considered a necessary condition for the local planner to work properly.

based on the following statement: *Assuming a given a given vehicle configuration (position, orientation, speed and wheel angle), the time until collision with a given static or dynamic object in the map can be calculated.* Therefore, velocity obstacle-based methods calculate this collision time for all surrounding objects and for all vehicle velocities. thus creating a velocity obstacle map, or forbidden velocity map (also referred to as dynamic velocity space in [4]). Using the dynamic window approach is advised in this step,as it filters out non-reachable velocities before collision-time calculations, thus decreasing the execution time of the algorithm. Given this velocity obstacle map, the algorithm has a basic knowledge of the reachable velocities and their level of safety. These safety levels provide a good starting point for further calculations, trajectory and collision estimations. My algorithm converts the forbidden velocity map's collision times to safety factors, but also takes the destination point's position and the target speed into consideration when selecting the next actuator outputs. The next graph shows all the sub-tasks of the local trajectory planner algorithm.

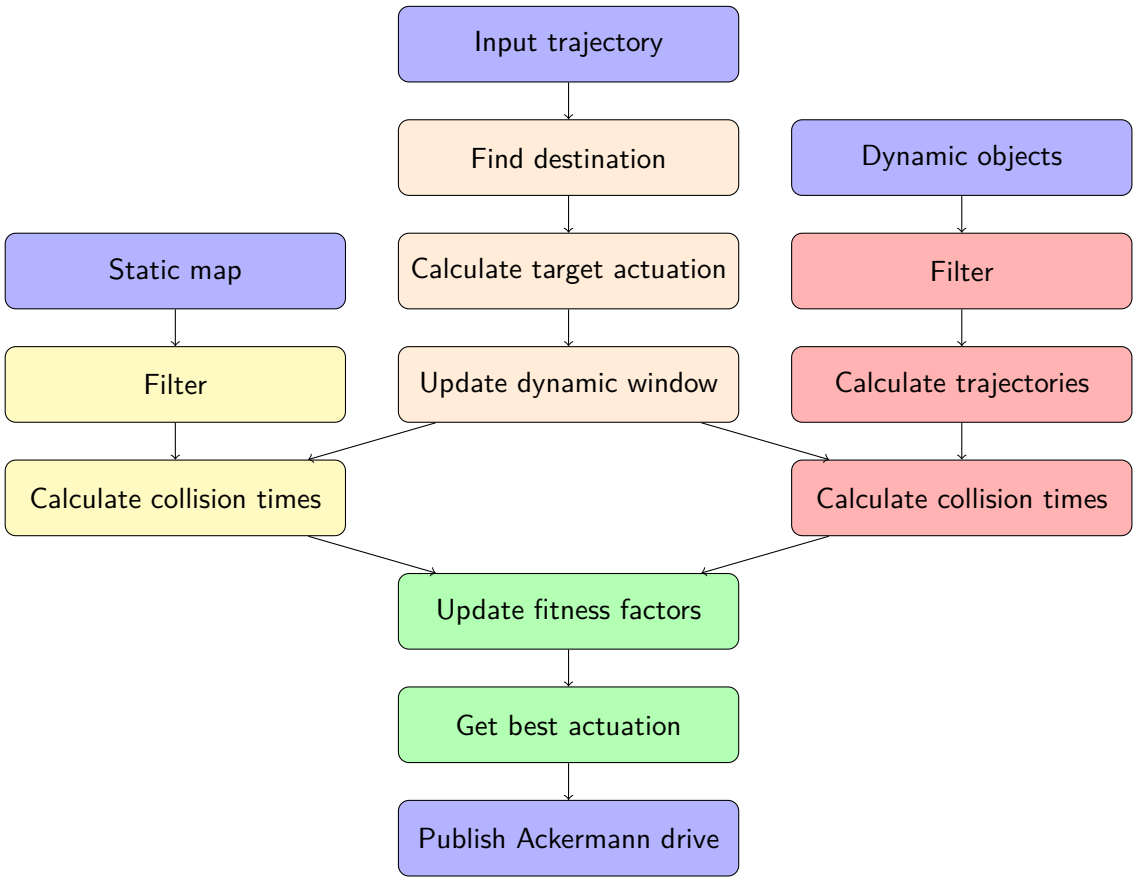


Figure 3.1: Motion planning method

The diagram consists of 4 subgraphs - these are also marked on the diagram with different colors. The orange subgraph describes the environment-independent sub-tasks of the algorithm - finding the next destination point, updating the target actuation and the dynamic window. The yellow one is about the method of reducing the size of the static map by filtering out those points that are not needed for the trajectory planning. The red side subgraph shows that for dynamic objects, trajectory calculation is also necessary besides filtering. And the last one (populating the bottom area of the graph, marked with green colour) contains the main planning logic. The next sections are going to explain these subgraphs in detail.

3.3 Target actuation and dynamic window

In every loop (at each new incoming map data), the target actuation and the dynamic window are updated. The target actuation describes the desired speed and steering wheel angle for the next step, therefore the desired linear and angular speed vectors. The dynamic window is independent from the target actuation, but depends on the current actuation. It contains a set of speeds and wheel angles that are reachable by the car within the next step. The width and height of the window (the maximum speed and wheel angle change in one step) are defined by the mechanical characteristics of the car. So basically, the aim of this subtask is to provide the algorithm a target actuation, and a set of reachable actuations. When I implemented the target actuation update mechanism, that leads the car according to the input trajectory, I had several options to choose from. The first option was to split the trajectory into line-segments, and implement and tune a steering angle controller (e.g. a PD controller) to follow this line. I dropped this idea to prevent the re-tuning that would have been needed for each target vehicles (simulation, real-life test vehicles). Another option was the 'invisible string' the 'dog bone' method, where a target destination point is always in front of the car, and the car is always trying to reach this point, as if the point was pulling it by a string. The target actuation's wheel angle defines a trajectory curve, that goes through the destination point. If the destination point is always on the trajectory, and in front of car (within the right distance range), the car is goint to follow the trajectory. I chose the latter (the 'invisible string') method.

3.3.1 Finding the next destination point

This method requires a destination point in each iteration. This point is preferably on the trajectory, in front of the car, 'pulling' the car to the right direction. The destination point search start with finding the point in the trajectory nearest to the car's current position. The destination point needs to be in front of the car, but its distance from the car (which is nearly equal to its distance from the previously found nearest point) is not trivial (see figure 3.2).

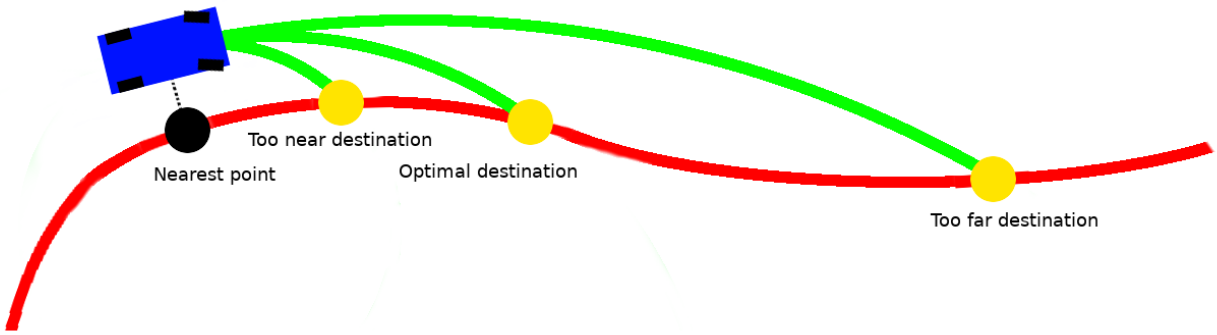


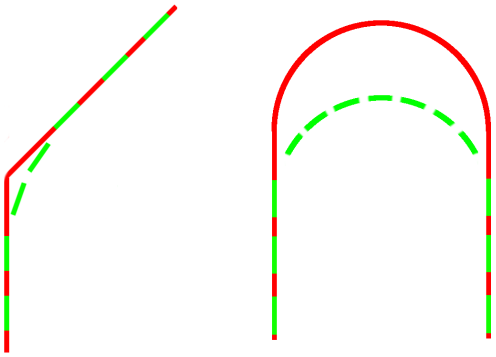
Figure 3.2: Updating the destination point

Choosing a too near destination will result in the car starting to oscillate around the target trajectory with an increasing amplitude, until a point where the angular difference between the car and the trajectory will be too large for the car to find the path again. This situation is displayed in figure 3.3.

However, the destination point must not be chosen too far, either, because it straightens the curves of the trajectory, rounds its sharp edges, and adds an offset to the long circular

Figure 3.3: Oscillation around the target trajectory

sections. Figure 3.4 demonstrates these effects. The effect shown on the left side might even be advantageous, but the righ-hand side effect is certainly not.



(a) Rounding sharp peaks **(b)** Rounding curves

Figure 3.4: Trajectory rounding

After empirical testing, the optimal distance of the destination point and the nearest trajectory point (marked with yellow and black dots in figure 3.2) proved to be around 4 times the distance between the car's front and rear axles. Note, that this value largely depends on the dynamic window's maximum angular velocity limit and the car's mechanical characteristics (e.g. using a faster steering servo or a 4-wheel steering car would reduce the optimal distance).

3.3.2 Calculating the target actuation

After the destination point has been selected, the next step is to calculate the target actuation - the speed and steering angle pair that would lead the car to the destination point. First, the target wheel angle is calculated using linear algebra (see figure 3.5). The target speed calculation is based on two factors. The first is the target speed limit, which we expect the car to keep, given no extraordinary circumstances. This should be set to a safe speed that the car can keep up during its travel. The actual target speed will never exceed this limit. But the target steering angle may require this target speed to be lower. Trivially, the car can manage to reach higher speeds safely when going straight, than when in a sharp bend.

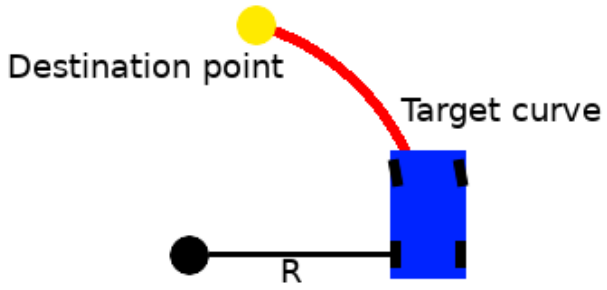


Figure 3.5: Calculating the target steering angle

3.3.3 Updating the dynamic window

3.4 Target actuation

3.5 Static velocity obstacle map

Static points in the map populate a very large percent of the car's surroundings, so handling them efficiently can boost up the algorithms runtime performance. Therefore, they are not joined with the dynamic points, but filtered separately, and the collision times are calculated in a way that is optimized for non-moving objects to gain performance.

3.5.1 Filtering static points

Separating the static points from the dynamic objects has both advantages and disadvantages for local motion planning. The most important advantage is that the collision check with static points is much simpler and faster than with dynamic obstacles. A great disadvantage, however, is that the static map contains lots of points, and running a collision check on all of them would cause the algorithm to be unacceptably slow. To prevent this undesired behaviour, a preparatory filter is applied to the map, removing those points from the map, that are too far from the car's current position to be worth keeping. Depending on the environment's characteristics, this step may reduce the runtime of the static collision times drastically - even to zero, if all the static points are far from the car (e.g. in a large room).

Figure 3.6 shows that the mapped environment of the car is a much larger territory than what the motion planning algorithm needs to check for collisions. The circle marks the area that is kept after the filtering. In this example, this area contains the two square-shaped obstacles, all the other points in the static map are thrown.

3.5.2 Static collision times

Collision times Due to the fact that the number of static points is usually high compared to the dynamic points, the collision times are calculated separately, so that the calculation can be optimized for static objects. Collision check with static objects is executed the following way. The

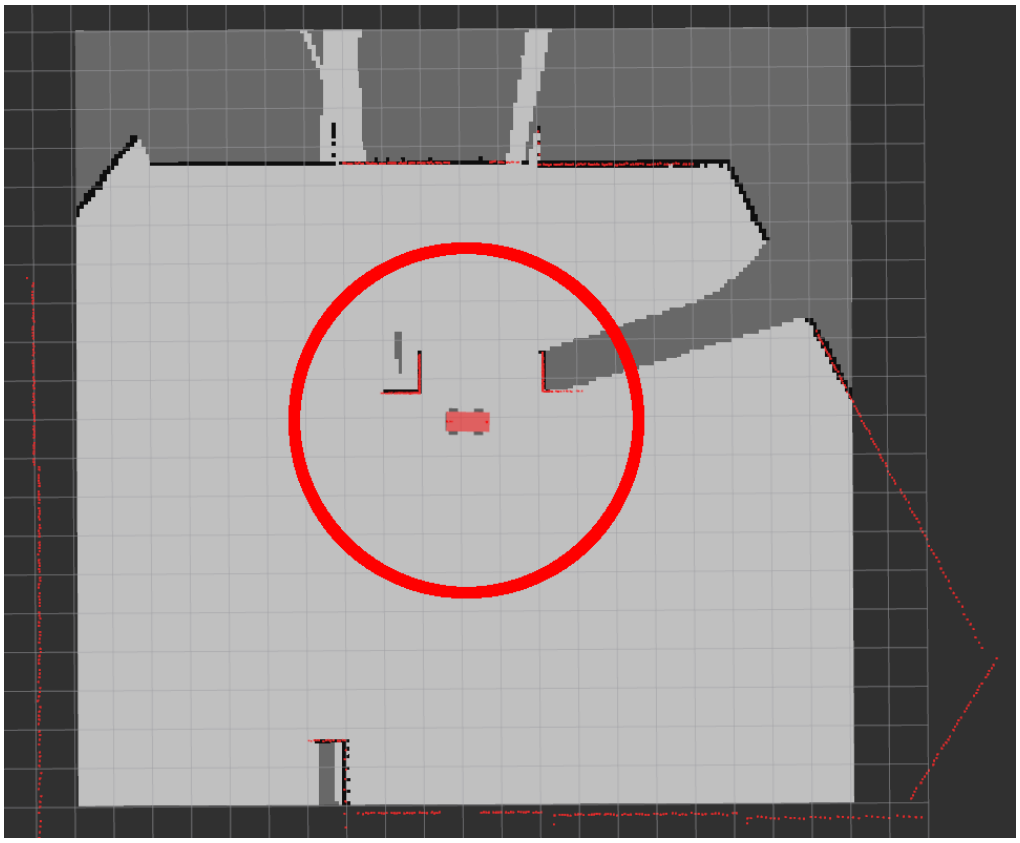


Figure 3.6: Static filtering

3.6 Dynamic velocity obstacle map

3.6.1 Filtering dynamic objects

3.6.2 Trajectory calculation

3.6.3 Dynamic collision times

3.7 Velocity obstacle map evaluation

3.7.1 Fitness factors

3.7.2 Best actuation

3.7.3 Publishing Ackermann driving control

Bibliography

[1] Baifan Chen, Zixing Cai, Zheng Xiao, Jinxia Yu, and Limei Liu. Real-time detection of dynamic obstacle using laser radar. 5(1):1–5, 11 2008. DOI: 10.1109/ICYCS.2008.357.

[2] Bruno Damas and José Santos-Victor. Avoiding moving obstacles: the forbidden velocity map. 6(1):1–6, 10 2009. DOI: 10.1109/IROS.2009.5354210.

[3] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. 13(1):1–13, 7 1998. DOI: 10.1177/027836499801700706.

[4] M. G. Mohanan and Ambuja Salgoankar. A survey of robotic motion planning in dynamic environments. 15(1):1–15, 10 2017. DOI: 10.1016/j.robot.2017.10.011.