UNIVERSIDAD POLITÉCNICA DE YUCATÁN

BIS Universities

Internship Report
Submitted for the degree of
Computational Robotics Engineer

Crowd navigation using a mobile robot

Carlos A. Alvarado Rostro

1. *External Supervisor* Dr. Jean-Bernard Hayet
Computer's Science Department
CIMAT

2. *Internal Supervisor* Dr. Ignacio Enrique Llanez Caballero
PhD in Renewable Energy
Universidad Politécnica de Yucatán

April 22, 2021

Contents

Acknowledgments v

Abstract vii

1 Introduction 1
- 1.1 State of art 2
  - 1.1.1 Velocity Obstacle 2
- 1.2 Problem Statement 4
- 1.3 Justification 5
- 1.4 Scope and limitations 6
  - 1.4.1 Work area 6
  - 1.4.2 Limitations 6
- 1.5 Objectives 6
  - 1.5.1 General 6
  - 1.5.2 Specific 7

2 Theoretical framework 9
- 2.1 Differential Drive Robot (DDR) 9
  - 2.1.1 Forward Kinematics of a DDR 9
- 2.2 Quadrilateral Velocity Obstacle 10
  - 2.2.1 Velocity Obstacle Chains 11

3 Methodology and development 13
- 3.1 Detection module 13
- 3.2 Prediction module 14
- 3.3 Navigation module 14
- 3.4 Tracking module 14
- 3.5 Software tools 14
  - 3.5.1 Qt Creator and openFrameworks 14
  - 3.5.2 ROS 15
  - 3.5.3 PCL 16
  - 3.5.4 OpenCV 17
  - 3.5.5 LibDAI 17
  - 3.5.6 MoCap OptiTrack 18

4 Results 21
- 4.1 User's manual 21
- 4.2 Experiments 21

5 Conclusions and recommendations 23
- 5.1 Conclusion 23
- 5.2 Recommendations 23

List of Figures

1.1 Movement and sensing equipment. 1
1.2 *VO* construction using two discs agents. Taken from [1]. 2
1.3 Velocity Obstacle vs Reciprocal Velocity Obstacle. Taken from [2]. 3
1.4 **a)** *VO*. **b)** *RVO*. **c)** *HRVO*. Taken from [3]. 4
1.5 Algorithm Simulation with different number of obstacles. 5
1.6 Trial environment with one fixed obstacle. 6

2.1 **a)** 3-DX8, pioneer of DDR. **b)** Parameters of a generic DDR. Taken from [4]. 9
2.2 Kinematic model of a DDR with a circular shape. Taken from [5]. 10
2.3 **a)** Pure translation. **b)** Pure rotation. Taken from [4]. 10
2.4 **a)** *VO* towards infinity. **b)** *VO* bounded. **c)** Construction of *QVO*. Taken from [1]. 11
2.5 Cluster of a trajectory. Taken from [1]. 11
2.6 **a)** *VOC*. **b)** Shape of *QVOC* model. **c)** New shape of *QVOC*. Taken from [1]. 12

3.1 ROS modules. 13
3.2 Example using RViz. Taken from [6]. 16
3.3 Use of Gazebo. Taken from [7]. 16
3.4 People tracking. Taken from [8]. 17
3.5 OpenCV library for object detection. Taken from [9]. 18
3.6 OptiTrack configuration. 19
3.7 Selecting rigid bodies to publish. 19

4.1 Analysis of the algorithm using Gazebo. 21
4.2 Applying QVOC prediction in 3D simulation. 22

Acknowledgments

I would like to sincerely thank the Centro de Investigación en Matemáticas A.C. (CIMAT) for provided me with financial support to conclude the bachelor's degree studies from the Universidad Politécnica de Yucatán (UPY).

I am grateful as well to the academic and administrative personal from both UPY and CIMAT, for the facilities provided and the infrastructure use, which was indispensable for the development of the project. To Dr. Jean-Bernard Hayet and Dr. Ignacio Enrique Llanez Caballero for the opportunity, trust and tutelage through the project.

Abstract

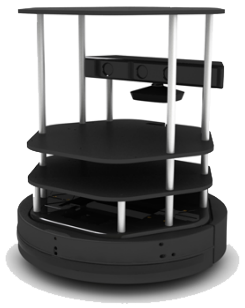
Robots have been used in many fields of industry and science. Since their first integration in those fields, there has been an increment in their demand along with efficiency. Along with robots many new fields and studies have come along and computational robotics is one of them. Computational robotics engineering approach is the comprehension and manipulation of the physical world through software engineering. In particular, it takes advantage of the high computational complexity required to exploit the integration capacity of sensors and actuators for advanced robotics. For this reason, as computational robotics student, this project presents a great opportunity using previous work of CIMAT master's degree students which particularly focused on the application of computational vision as well as navigation techniques within crowds for a mobile robot to freely move while avoiding any possible obstacle.

1 Introduction

As time goes by, technology specially the robotics area is developing in a tremendous way, and one of the biggest factors comes with the term *"autonomy"*. The word 'autonomous' has become widely used in many different areas such as: Artificial Intelligence (AI), robotics, etc. Most of the time, it is used to qualify the type of systems, agents, or robots being investigated. Despite this widespread use, the word does not always mean the same thing. Sometimes it has to do with intelligence, other times it simply means a mobile robot, as opposed to a stationary one, or it is used to refer to an agent that decides things for itself, intelligently or otherwise [10].

Therefore the concept of autonomy that is going to be handled on this project approaches more into the agent or mobile robot to act and decide by itself given that the only information that will be provided is the reaching point or goal; everything else is learnt constantly as it keeps moving. Unfortunately, one of the main issues is the motion planning given that the purpose (as previously mentioned) is to let the mobile robot without any previous information about the environment, obstacles (which can be either static or with constant velocities and trajectories) to reach a desired point in the exposed environment avoiding collision, prioritizing the previous over reaching the goal.

In the case of trials, we use the second generation of the TurtleBot named TurtleBot 2e. Before all, what is a TurtleBot? TurtleBot [11] is a low-cost, personal robot kit with open-source software. With TurtleBot, you'll be able to build a robot that can drive around with a single board computer such as the 96 Boards CE computer, the DB410c. The recognition is implemented using a Xbox Kinect Sensor [12] given that it contains an RGB color VGA video camera and a depth sensor. The camera detects the red, green, and blue color components. The depth sensor and infrared projector help to create the 3D imagery throughout the room.


(a) TurtleBot 2e

(b) Xbox Kinect Sensor

Figure 1.1: Movement and sensing equipment.

1.1 State of art

To achieve the autonomy on the mobile robot, there are several techniques for collision avoidance. There are two main categories for collision avoidance methods, namely: single-step and multi-step. In single-step [1], usually there is a global notion of the workspace, such as: dimensions and characteristics of the world and obstacles and their location.

The second category is multi-step [1]. The difference with single-step is that it may or may not need global knowledge of the workspace. They are characterized by being more local and reactive; they focus on solving the collision avoidance problem constantly (at every step) usually only around the robot. An advantage they have over one-step (single-step) methods is that they can avoid obstacles discovered in real time because they constantly update their status and relative positions of the obstacles in their world. This is very important because it supposes a more realistic performance and is able to face situations that single-step models could not solve.

1.1.1 Velocity Obstacle

Velocity Obstacle (*VO*) [13] is a geometrically-based, multi-step method consisting of generating an obstacle within the velocity space of the robot by combining dimensions of both the robot and the current obstacle and their velocities. This approach is very practical because it does not require much computational power for its development. *VOs* is a fundamental part of the algorithm given that with a few modifications it manages to handle multiple-agents environment. In addition, it is not necessary to know the workspace. Figure 1.2 provides a visual example on how two disc shaped agents (denoted A and B) and the respective *VO* is implemented. Figure 1.2(b) represents the formation of the collision cone of A induced by B, starting from A center. Lastly, Figure 1.2(c) shows the *VO* of A induced by B to approximate a possible collision.

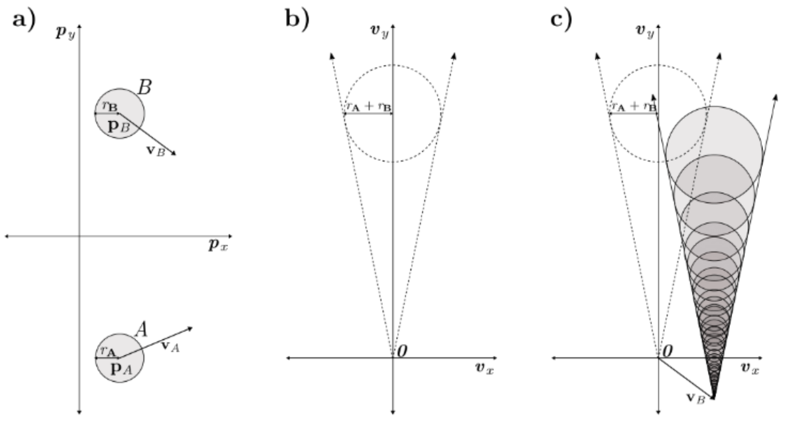


Figure 1.2: *VO* construction using two discs agents. Taken from [1].

Different types of VO so far

The use of *VO* in multi-agent configurations generates undesired oscillations in paths. These oscillations are generated in this approach because no robot takes into account that others also adjust their velocities in each time-step, making valid velocities that previously were not. To address this problem some approaches have been designed to reduce oscillations. A few are: **Reciprocal Velocity Obstacle** (*RVO*), **Hybrid Reciprocal Velocity Obstacle** (*HRVO*) and **Extended Velocity Obstacle** (*EVO*)

RVO

RVO implicitly assumes that other agents make a similar collision-avoidance reasoning. Under this assumption, is guaranteed to generate safe and oscillation-free motions [2]. This method is an extension of the *VO*. The approach inherits all of of its appealing properties, but a new capability is introduced to resolve the common oscillation problem in multi-agent navigation. Figure 1.3 presents a path followed by two agents that have opposite preferred velocities and are on a immediate collision course. Oscillation between the two is shown using the original concept of *VO*, nevertheless, implementing *RVO* this oscillation is gone.

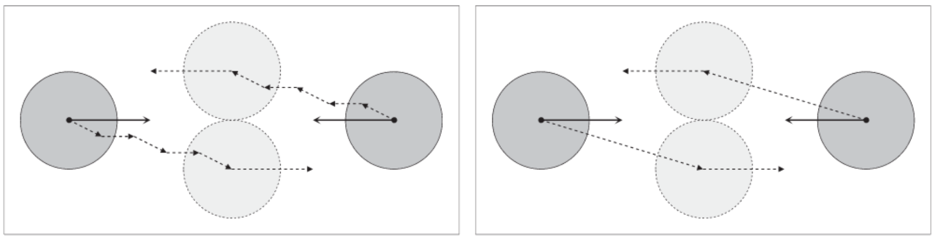


Figure 1.3: Velocity Obstacle vs Reciprocal Velocity Obstacle. Taken from [2].

HRVO

When a robot chooses a velocity outside the *RVO* induced by another robot, both automatically choose to pass through the same side (left or right) of each other, obtaining trajectories free of collisions and oscillations. This approach guarantees the above, as long as there is only one agent influencing another; otherwise, the robots will not necessarily choose the same side to pass. The presence of a third robot C may cause at least one of the robots to choose a velocity even farther from its current velocity. Unfortunately, this means that robots may not necessarily choose the same side to pass, which may result in oscillations known as “reciprocal dances”.

In order to solve the problem of “reciprocal dances”, the *HRVO* approach was designed. *HRVO* [3] consists in choosing one of the two sides of the other agent for consensus and apply reciprocity. Otherwise, if the other side is chosen, then the agent must take full responsibility to avoid collision. Figure 1.4 represents paths followed by five robots moving simultaneously across a circle comparing the behavior using *VO*, *RVO* and *HRVO*. Taken from [3]

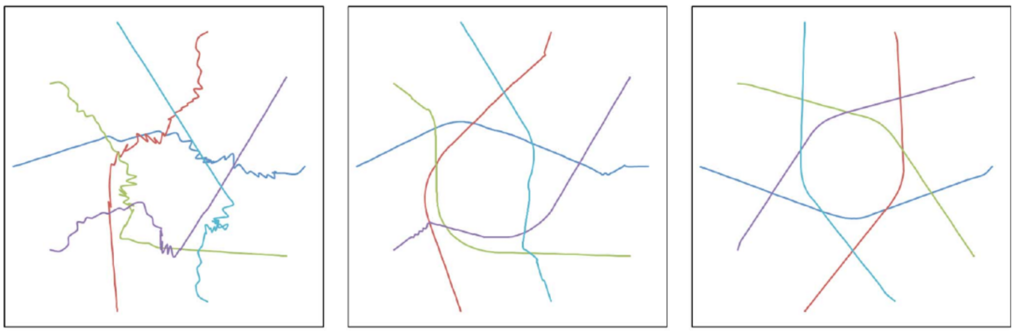


Figure 1.4: **a)** *VO*. **b)** *RVO*. **c)** *HRVO*. Taken from [3].

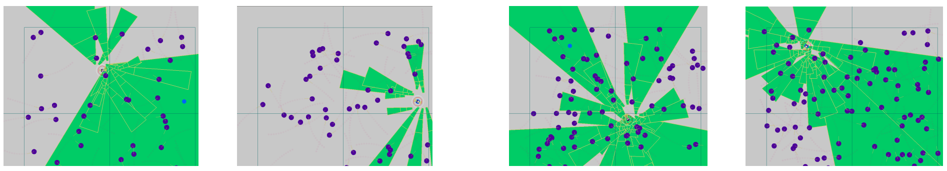
EVO

A modification of the *VO* approach has been made to handle collision avoidance and oscillations by enlarging the collision cone. *EVO* [14] uses predicted, communicated, or derived information about the agents' future intentions. It prevents collisions not only with the current velocity, but with an interpolation of the future velocity. That way, the robot anticipates the behavior of the other agent and takes actions to obtain a smooth, collision-free and blockage-free trajectory.

1.2 Problem Statement

The algorithm implementation has been carried out through simulations in 2D and 3D. The 2D simulation was done using openFrameworks (OF) and Qt Creator. In it, it is possible to manipulate many factors of the simulation such as: the number of moving obstacles (which represent humans), the environment size, the total simulation time, among others.
Next, for the 3D simulation the experiments were accomplished using simulated TurtleBot robots using the Gazebo simulator which contained not as many simulated humans as in the 2D, but it is able to recognize and avoid the simulated humans as well as static obstacles all thanks to the integrated library with a control based on ROS. Figure 1.5 presents four different sets of simulations. The green lasers represent the prediction for collision avoidance which tell the robot when a collision is certain, so the best approachable path is one outside the green marked area.

As previously mentioned, the project has been tested trough simulations in 2D and 3D. Unfortunately, the algorithm has not yet been tested in a real environment, with actual humans moving along or static obstacles. Even tough the simulations are great proofs to prove and show how the algorithm works, there are situations with real-life trials with all the variables that apply to a not fully controlled environment. This can lead to a new problematic given that even though the trials would be done in a "controlled environment", there can always be issues with either the floor friction, its irregularity with cracks, style, the TurtleBot, etc.



(a) 2D simulation with 50 moving obstacles.

(b) The simulated mobile robot has reached the goal but still prevents any possible collision.

(c) 2D simulation but now with 100 moving obstacles.

(d) Same results with 100 obstacles.

Figure 1.5: Algorithm Simulation with different number of obstacles.

To perform experiments in a real environment, to be able to compute the robots movements and position as well as for the obstacles within the environment, the OptiTrack motion capture system will be used given its facility to perform all previous tasks. This will be possible using 13 cameras which would mark off the limits of the environment and capture the location of the OptiTrack markers mounted on the robot and obstacles.

1.3 Justification

It is not a fuss that big technological companies have been working on autonomous robots capable of handling themselves in the real world with all its uncertainty due to the entire lack of information. And so, experiments in this field can involve robots exposed to a not controlled or studied environment, for example: a street full of people in which the robot does not the actual know the number of person or where all are walking in one direction but there can be moments in which someone stops for checking the time, remembering something they missed, etc.

Te development of new theories in collision avoidance capable of deal with error or limitation of previous is essential so this field can keep growing and gain even more importance. Just like is briefly explained in section 1.1, collision avoidance theories might sound flawless at the beginning, but by going off what was tested each one has one or few areas of opportunity. That is the reason why from *VO* came *RVO* dealing with oscillations; at the same time from *RVO* arrived *HRVO* managing to eradicate reciprocal dances.

Is important to start the trial in a real environment being either controlled or not controlled. Real life trials can be helpful to determine how the TurtleBot really behaves with such algorithm with different obstacles classified by maybe size, color, velocities, etc. All with the purpose of analyzing and determine what can be improved. There is a huge gap between a controlled environment to being exposed into the real world. There is much more to deal with, random behavior, ground, obstacles and many more.

1.4 Scope and limitations

1.4.1 Work area

The trials are performed in a floor composed by flagstone, each one divided by lines of approximately 0.4 mm. The area is roughly 7x5 meters. There is also the possibility to work with a mat of the same dimensions to make the movement smoother. Figure 1.6 allows to have an estimate on the size of the area compared to the size of a TurtleBot; it as well displays the infrared cameras which work along the OptiTrack which has already been talked about.

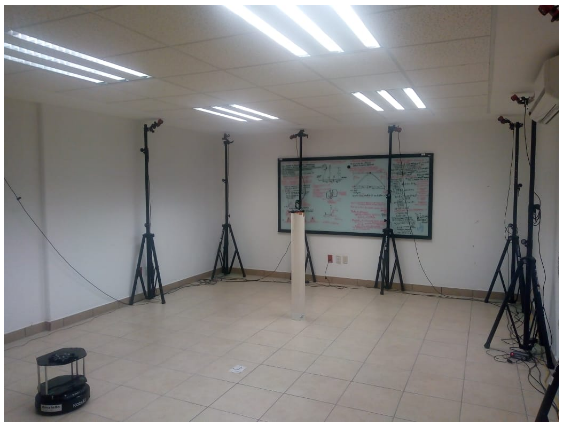


Figure 1.6: Trial environment with one fixed obstacle.

1.4.2 Limitations

Unfortunately, it is not as easy to provide many moving obstacles or make the environment size as desired. The project is bounded by the previous area of 7x5. In that space it is not possible to introduce more than 2-3 moving obstacles (in this case, actual humans). The trials are bounded by what is accessible in the limited space due to the necessity of work in a controlled area for the time being.

Furthermore, another potential limitation is the composition of the floor. As mentioned before, its flagstone divided by lines as appreciated in Figure 1.6. This can become an error factor for the actual movement performance of the TurtleBot, given that there can be situations in which it might get stuck or that each time it passes through them, its velocity can get compromised as well as its avoidance reaction and movement.

1.5 Objectives

1.5.1 General

The aim of the TurtleBot either simulated or in real life during the testing is to be able to navigate around without any previous information about the environment, the obstacles (should not matter if dynamic, static or even both) avoiding any probable collision and finally reaching a specific point in the area, in other words its goal.

In addition, we aim at providing a specific guide-line in how to set-up, operate and work with the previous project made by CIMAT's masters students given that it has been untouched since 2017. The point is to create a tool to simplify future work progress on both projects.

1.5.2 Specific

- To allow future work to have a smoother and faster start through the user's manual.
- Deal with launching issues of the project.
- Implementation of dynamic libraries.
- Test the algorithm

2 Theoretical framework

This chapter consists of information that will help the reader understand theories that support our project. First, forward kinematics to comprehend a little more about the structure and movement of the TurtleBot. Secondly, the collision avoidance approach implemented in the algorithm that allows the simulated robot to perfectly avoid dynamic obstacles.

2.1 Differential Drive Robot (DDR)

Most indoor mobile robots do not move like a car. For example, consider the mobile robotics platform shown in Figure 2.1. This is an example of the most popular way to drive indoor mobile robots. It has two main wheels, each of which is attached to its own motor. A third wheel (not visible in Figure 2.1) is placed in the rear to passively roll along while preventing the robot from falling over, this third wheel does not possess a motor thus it can be ignored in the mathematical model. The TurtleBot which has been mentioned corresponds to this type of robot model given that it presents the same characteristics.

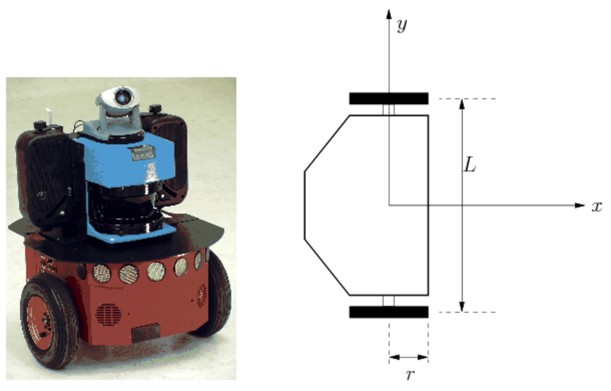


Figure 2.1: **a)** 3-DX8, pioneer of DDR. **b)** Parameters of a generic DDR. Taken from [4].

2.1.1 Forward Kinematics of a DDR

Just to avoid confusion, Figure 2.2 represents a more accurate kinematic model of the TurtleBot. As stated, it has two main wheels, fixed in orientation, each attached to its own motor such they can act independently from each other at different speeds if necessary. The third wheel, which is not shown because it only adds support, does not affect the model.

The DDR configuration [5] is represented by the position (x, y) and orientation θ. The distance between the two main wheels will be defined by L, left and right wheel speeds are V_l and V_r, respectively, making the configuration transition equations as:

$$\dot{x} = \frac{V_l + V_r}{2} \cos\theta \tag{2.1}$$

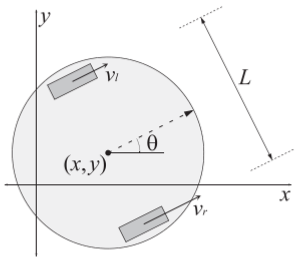


Figure 2.2: Kinematic model of a DDR with a circular shape. Taken from [5].

$$\dot{y} = \frac{V_l + V_r}{2} \sin\theta \tag{2.2}$$

$$\dot{\theta} = \frac{V_l + V_r}{L} \tag{2.3}$$

Furthermore, the wheel speeds are constrained to a set maximum V_{max}, such that:

$$-V_{max} \leq V_l \leq V_{max}, -V_{max} \leq V_r \leq V_{max} \tag{2.4}$$

The wheels speed are the control input of the robot. When $V_l = V_r > 0$, the robot will move straight; when $V_l > V_r > 0$, it will arc towards the right and to the left if vice versa; and when $V_l = -V_r \neq 0$, it will spin in its place. Figure 2.3 shows what is has been explained more graphically.

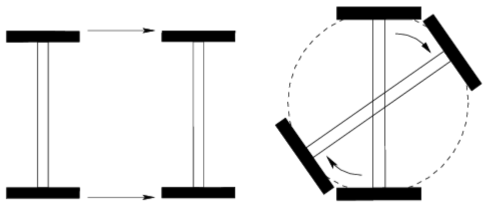


Figure 2.3: **a)** Pure translation. **b)** Pure rotation. Taken from [4].

2.2 Quadrilateral Velocity Obstacle

Velocity Obstacle as has been explained in section 1.1, is defined by line segments and arcs of circles as the result of the approximation of the collision result of two agents given by a certain period of time. Furthermore, this can be modified in order to create a simpler and more computational practical model by generating a trapezoid of minimum area circumscribed to the to the corresponding VO. This new approximation is called **Quadrilateral Velocity based Obstacle** (QVO). Figure 2.4 represents a basic but illustrated form of the creation of a QVO model [1].

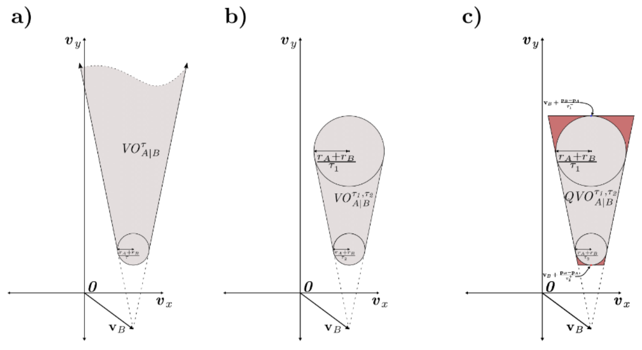


Figure 2.4: **a)** *VO* towards infinity. **b)** *VO* bounded. **c)** Construction of *QVO*. Taken from [1].

2.2.1 Velocity Obstacle Chains

The QVO model mentioned before, can be also usefully applied for creating a **Velocity Obstacle Chain** (VOC). To apply it, is necessary for A to be induced by a mobile agent B, from which the trajectory has already been predicted or known. This predicted trajectory is by approximation represented using line segments between points on it. Each sub-path (line segment) corresponds to the velocity and position of the dynamic obstacle along different periods of time. For instance, see Figure 2.5 [1].

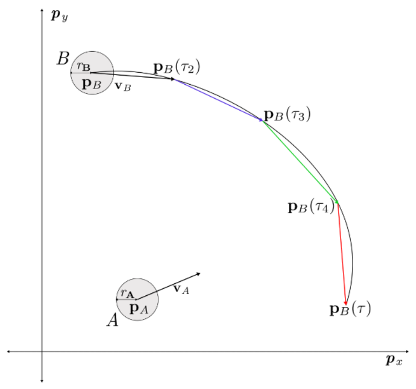


Figure 2.5: Cluster of a trajectory. Taken from [1].

Each mobile obstacle is being associated with a continuous probability-distribution that predicts its trajectory for a few time steps in the future. Depending on the predictor and observed data, these models are updated, and we can deduce a deterministic prediction of the trajectory within the next time span.

Then, VOC can correspond to the union of each VO within the desired time. As a result, **Quadrilateral Velocity Obstacle Chain** ($QVOC$) can be defined as the corresponding VOC by making the analysis and approximations necessary.

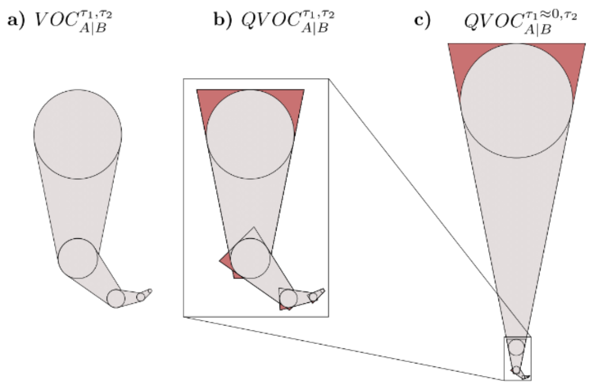

a) $VOC^{\tau_1,\tau_2}_{A|B}$
b) $QVOC^{\tau_1,\tau_2}_{A|B}$
c) $QVOC^{\tau_1\approx 0,\tau_2}_{A|B}$

Figure 2.6: **a)** *VOC*. **b)** Shape of *QVOC* model. **c)** New shape of *QVOC*. Taken from [1].

Figure 2.6 graphically shows a *QVOC* construction. In Figure 2.6(b) the red areas are constrained to maintain the *QVO* model. Lastly, Figure 2.6(c) is when the lower time is approximated to 0, making the red area of little significance thus, be ignored.

3 Methodology and development

The following chapter presents information about the ROS modules used in the project algorithm. These modules go from the connection of the TurtleBot to ROS to the navigation and prediction; essential to perform the required collision avoidance. Additionally, it also gives a brief explanation of all software implemented along the progression of the project.

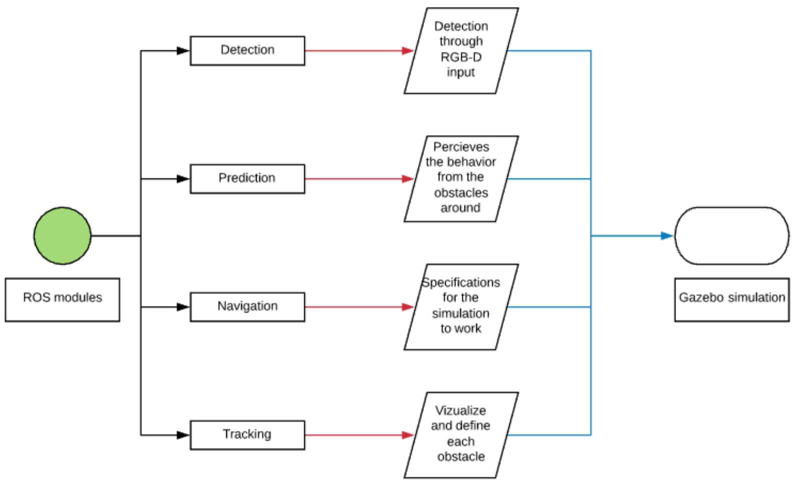

ROS modules
Detection
Prediction
Navigation
Tracking
Detection through RGB-D input
Percieves the behavior from the obstacles around
Specifications for the simulation to work
Vizualize and define each obstacle
Gazebo simulation

Figure 3.1: ROS modules.

Figure 3.1 represents in a very simple way the purpose of each module in the ROS project to run the 3D simulation. Each module is based on cpp libraries and headers files that allow to complete the required function.

3.1 Detection module

The sensor we use to collect input data to apply the tracking algorithm is a Kinect sensor (mentioned in the introduction). All the essential complements and/or libraries come from OpenCV, PCL and the package Spencer People Tracking. All these combined with the algorithm of the project are able to fairly detect people or objects using a RGB-D camera.

Ground-based RGB-D

This ROS package is a wrapper around a slightly modified version of the ground-based RGB-D people detection module from the Point Cloud Library. In contrast to the original implementation in PCL, the following features have been added in this version:

– Subscribe to an RGB-D stream via ROS, publish detection as *spencertrackingmsgs/DetectedPersons.*

– Publish extracted ROIs as *visualization_msgs/MarkerArray* for visualization in RViz.

– Transform the input cloud from input coordinate frame such that the ground plane (given by its ground-plane coefficients) is correctly aligned. This allows to use sensor setups in which the sensor is not perfectly horizontally aligned.

3.2 Prediction module

This module implements the use of the ROS library "ros_interface". The ROS Interface duplicates the C/C++ ROS API (Application Programming Interface) with a good fidelity. This makes it the ideal choice for very flexible communication via ROS, but might require a little bit more insight on the various messages and the way ROS operates.

The main purpose is to predict the behavior produced by the corresponding obstacle (call it human or robot). The prediction can vary from current position to a new predicted one, to get a distance between a set of points and to assign an ID to each obstacle.

3.3 Navigation module

The navigation modules fully depend of the QVOC library denominated "libqvoc". They were developed by CIMAT students' Andrei Raya and Emmanuel Ovalle. This library is located in the directory. By compiling them it creates and install the binaries of QVOC which allow the prediction and navigation. The libqvoc elements used are "human.h", "lineObstacle.h", "robot.h", "structures.h" and "Simulator.h" which create the representations of the objects behavior (call it human or robot) in the simulation. They also help the simulated robot to distinguish between obstacles.

3.4 Tracking module

One of the main theories implemented in tracking is "Near-Online Multi-target Tracking" (*NOMT*) given its efficiency in target dynamics, appearance similarity, and long term trajectory regularization. In this module it also happens the *Point Cloud processing* which is the collection, measurement and use of point clouds (collection of data points defined by a given coordinates system) to design 3D CAD models of a target object or surface.

3.5 Software tools

3.5.1 Qt Creator and openFrameworks

Qt Creator is an IDE which stands for *Integrated Development Environment* that helps the user to design and develop coding, applications and interfaces. Qt is designed for developing applications and user interfaces once and deploying them to several desktop, embedded, and mobile operating systems or web browsers. Qt Creator provides the tools for accomplishing your tasks throughout the whole application development life-cycle from managing projects, designing user interface, coding, building and running, testing and publishing to the target platform.

In a similar case, openFrameworks is an open source C++ toolkit designed to assist the creative process by providing a simple and intuitive framework for experimentation. It is designed to work as a general purpose glue, and wraps together several commonly used libraries, including:

- OpenGL, GLEW, GLUT, libtess2 and cairo for graphics.
- OpenCV for computer vision.

Both are used in the 2D modeling allowing to first see how the algorithm is supposed to work and give samples of the robot behaviour.

3.5.2 ROS

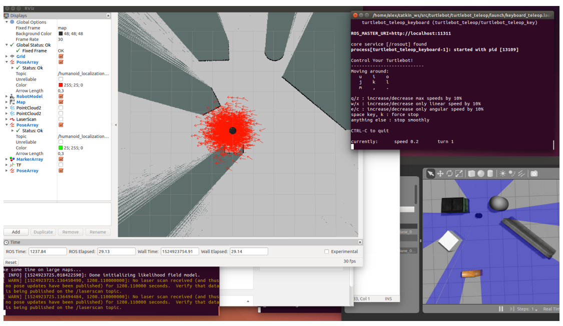
ROS is the acronym of Robot Operating System and it is a collection of tools, libraries and conventions to simplify the task of creating complex and robust robot behaviour. With ROS, it is easy to separate the code base into **packages** which are composed of programs called “nodes”.
ROS posses three main communication tools:

- Topics: Mainly used for sending data between nodes. Topics are always sending data.
- Services: Allow to create a client/server communication between nodes. Instant communication between client and server, useful when a specific change setting is needed or specific action.
- Actions: Is actually based on topics. They grant a client/server communication and in contrast with services, the client can send a request that might take an excessive amount of time.

To implement these communication tools, only the appropriate libraries and specific messages (which is the data structure of the information you send) are needed in the code.

RViz

RViz or ROS visualization is a ROS package for 3d visualization in ROS applications. It allows the user to have a view of the robot model working on, capture sensor information from the robot and replay the previous mentioned data. It can display data from camera, lasers, 3D and 2D devices including photos.



```
/home/alex/catkin_ws/src/turtlebot/turtlebot_teleop/launch/keyboard_teleop.lau
    turtlebot_teleop_keyboard (turtlebot_teleop/turtlebot_teleop_key)

ROS_MASTER_URI=http://localhost:11311

core service [/rosout] found
process[turtlebot_teleop_keyboard-1]: started with pid [13109]

Control Your Turtlebot!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:      speed 0.2       turn 1
```

ROS Time: 1237.84 ROS Elapsed: 29.13 Wall Time: 1524923754.91 Wall Elapsed: 29.14

```
ke some time on large maps...
[ INFO] [1524923725.018422590]: Done initializing likelihood field model.
[ WARN] [1524923725.136450490, 1208.110000000]: No laser scan received (and thus
 no pose updates have been published) for 1208.110000 seconds.  Verify that data
 is being published on the /laserscan topic.
[ WARN] [1524923725.136494484, 1208.110000000]: No laser scan received (and thus
 no pose updates have been published) for 1208.110000 seconds.  Verify that data
 is being published on the /laserscan topic.
```

Figure 3.2: Example using RViz. Taken from [6].

Gazebo

Gazebo lets you build 3D worlds with robots, environments, and other objects the user might want to add. It also has a physics engine for modeling illumination, gravity, and other forces providing the user with a good controlled simulation that can adapt to any real life scenario. Robotics developers use Gazebo to evaluate and test robots in different scenarios, sometimes even more than using physical robots. Gazebo also makes it easier to test other aspects of your robot like error handling, battery life, navigation, and machine learning algorithms.

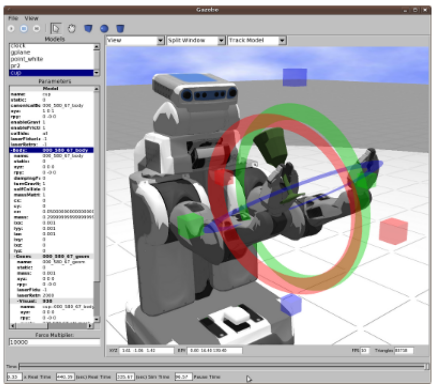


Figure 3.3: Use of Gazebo. Taken from [7].

Spencer People Tracking

The meaning behind the development of the *Spencer People Tracking* [15] algorithm and ROS module was to allow service robots to guide groups of people through crowded and highly unexpected environments, airports, malls for instance and at the same time not disturbing the people there. To reach this goal, robust and computationally efficient components for the perception and recognition of humans in the robot's surrounding needed to be developed or improved.

3.5.3 PCL

The Point Cloud Library [16] (PCL) is a standalone, large scale, open source project for 2D/3D image and point cloud processing. From an algorithmic perspective, PCL is meant

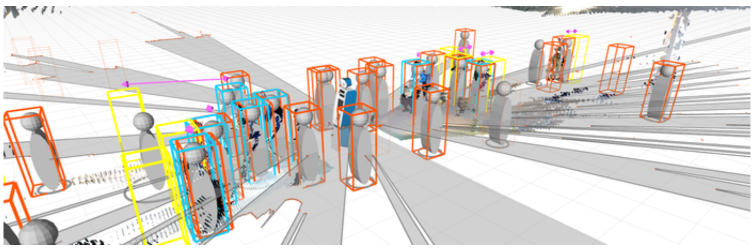


Figure 3.4: People tracking. Taken from [8].

to gather together a multitude of 3D processing algorithms that operate on point cloud data, including: filtering, feature estimation, surface reconstruction, model fitting, segmentation, registration, etc. Each set of algorithms is defined via base classes to combine all the common functionality used throughout the entire pipeline, thus keeping the implementations of the actual algorithms compact and clean. The basic interface for such a processing pipeline in PCL is:

- Create the processing object (e.g., filter, feature estimator, segmentation.)
- Use setInputCloud to pass the input point cloud data-set to the processing module.
- Set some parameters.
- Call compute (or filter, segment, etc) to get the output.

3.5.4 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

Accordingly, the library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. Their purposes go to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects as shown in Figure 3.5, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality and many more.

3.5.5 LibDAI

libDAI is a free/open source C++ library that provides implementations of various (approximate) inference methods for discrete graphical models. libDAI supports arbitrary factor graphs with discrete variables; this includes discrete Markov Random Fields and Bayesian Networks [17].

The library is targeted at researchers. To be able to use the library, a good understanding of

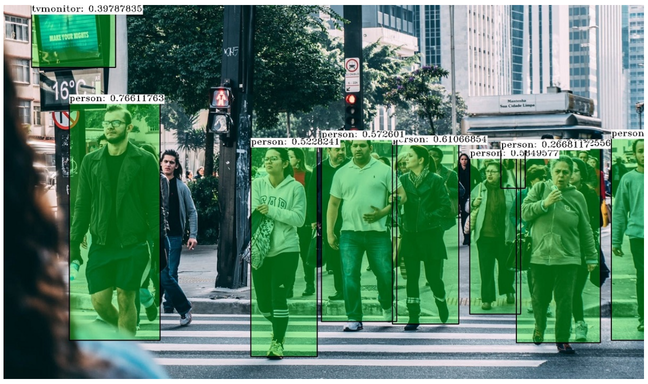


Figure 3.5: OpenCV library for object detection. Taken from [9].

graphical models is needed. libDAI can be used to implement novel (approximate) inference algorithms and to easily compare the accuracy and performance with existing algorithms that have been implemented already.

3.5.6 MoCap OptiTrack

This package contains a node that translates motion capture data from an OptiTrack rig to tf transforms, poses and 2D poses. The node receives packets that are streamed by the Tracking Tools software, decodes them and broadcasts the poses of configured rigid bodies as tf transforms, poses, and/or 2D poses.

Configuring tracking tools

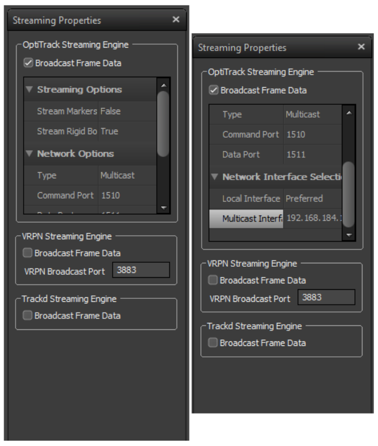
To configure the MoCap software for streaming of rigid bodies to the (virtual) machine it is necessary to run the mocap_optitrack ROS node.

1. Open the “Streaming Properties” pane in Tracking Tools and enable the “Broadcast Frame Data” checkbox. Set the “Stream Rigid Bodies” option to “True” if it’s not set already.
2. Set “Type” to “Multicast”, the ports should be left as-is.
3. Unless the machine running Tracking Tools has multiple network interfaces, the “Local Interface” may be set as “Preferred”. The “Multicast Interface” should be set to the address of the machine running the mocap_optitrack node.

Configuring mocap_optitrack

Once motion capture data is being streamed to the mocap_optitrack node, the mapping of trackables to ROS topics must be defined. A sample configuration file called “mocap.yaml” is included with the package, you can find it in the config directory of the package.

Since multiple robots may be tracked by the OptiTrack system, the mocap_optitrack configuration allows for publishing poses and transforms for each robot separately. First, the robot must be set up as a trackable in Tracking Tools. Next, note the Trackable ID for each trackable in Tracking Tools.



OptiTrack Streaming Engine: ✓ Broadcast Frame Data

Streaming Options	
Stream Markers	False
Stream Rigid Bo	True
Network Options	
Type	Multicast
Command Port	1510

VRPN Streaming Engine: Broadcast Frame Data; VRPN Broadcast Port 3883

Trackd Streaming Engine: Broadcast Frame Data

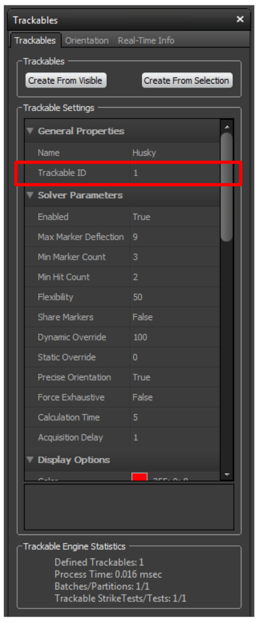
OptiTrack Streaming Engine: ✓ Broadcast Frame Data

Type	Multicast
Command Port	1510
Data Port	1511
Network Interface Selecti	
Local Interface	Preferred
Multicast Interf	192.168.184.1

VRPN Streaming Engine: Broadcast Frame Data; VRPN Broadcast Port 3883

Trackd Streaming Engine: Broadcast Frame Data

Figure 3.6: OptiTrack configuration.



General Properties	
Name	Husky
Trackable ID	1
Solver Parameters	
Enabled	True
Max Marker Deflection	9
Min Marker Count	3
Min Hit Count	2
Flexibility	50
Share Markers	False
Dynamic Override	100
Static Override	0
Precise Orientation	True
Force Exhaustive	False
Calculation Time	5
Acquisition Delay	1
Display Options	

Defined Trackables: 1
Process Time: 0.016 msec
Batches/Partitions: 1/1
Trackable StrikeTests/Tests: 1/1

Figure 3.7: Selecting rigid bodies to publish.

The published topics for each trackable can now be configured in mocap_optitrack. Each trackable that should be published in ROS can be specified by its Trackable ID - the desired topics for the published Pose, 2D Pose and the frame_id of the transform in tf can be configured.

4 Results

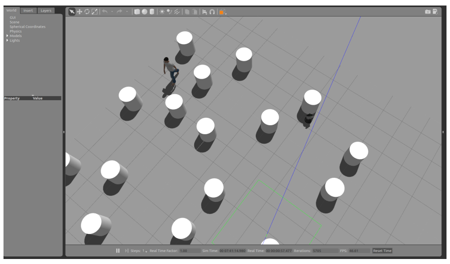
4.1 User's manual

Some modifications had to be done in order to properly run all ROS modules along with Gazebo and RViz for the project package. Given that the project was without updates, many of the programs or packages required were discontinued or no longer existed. For some time, provoked delays, however my partner and I were able to finally properly run it and thus we wrote the manual for future work and to avoid any unnecessary trouble that could waste time.

Consequently, the manual is composed of all modifications that were necessary given updates, tips found on forums or websites that worked for us because as it was specified in the previous paragraph a few of the principal programs versions were no longer available or it was completely necessary to use a different version. Secondly, at the beginning there was a missing ROS package which was giving trouble.

4.2 Experiments

In Figures 4.1(a) and 4.1(b) we show two different situations. 4.1(a) allows to observe how the mobile robot has stopped after reaching its minimum available distance of an object (which in this case is one of the static objects). Lastly, 4.1(b) permits to see how it is orientated towards the person, which in this case its function is to follow. Once the robot reaches its target or goal, it does not stop, it starts to spin in its own axis to carefully analyze if an object is approaching and (in case of possible collision) leave its position.


(a) TurtleBot completely stopping.

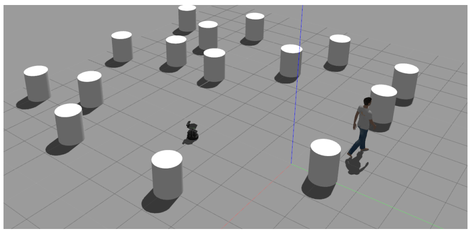
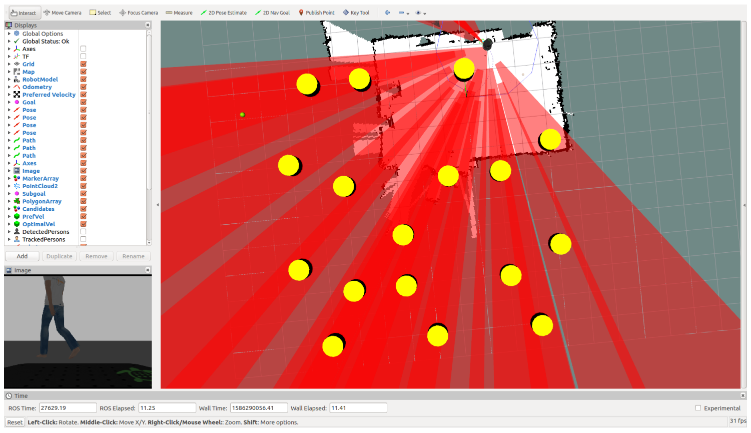

(b) Tracking and following the simulated human.

Figure 4.1: Analysis of the algorithm using Gazebo.

Figure 4.2 illustrates the performance of the **Quadrilateral Velocity Obstacle Chain** model using Gazebo and RViz. The present image, shows the “behind scenes” allowing to see linear and angular velocity of the robot; it also shows how the *QVOC* would appear in that environment with static obstacles which represent the yellow figures on the red-laser coming from the robot. The camera simulation provides sight on the target (which in this case is a moving simulated human).



- Global Options
- Global Status: Ok
- Axes ☐
- TF ☐
- Grid ☑
- Map ☑
- RobotModel ☑
- Odometry ☑
- Preferred Velocity ☑
- Goal ☑
- Pose ☑
- Pose ☑
- Pose ☑
- Pose ☑
- Path ☑
- Path ☑
- Path ☑
- Axes ☑
- Image ☑
- MarkerArray ☑
- PointCloud2 ☑
- Subgoal ☑
- PolygonArray ☑
- Candidates ☑
- PrefVel ☑
- OptimalVel ☑
- DetectedPersons ☐
- TrackedPersons ☐

ROS Time: 27629.19 ROS Elapsed: 11.25 Wall Time: 1586290056.41 Wall Elapsed: 11.41 ☐ Experimental

Reset **Left-Click:** Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options. 31 fps

Figure 4.2: Applying QVOC prediction in 3D simulation.

5 Conclusions and recommendations

5.1 Conclusion

Recently, multi-agent systems have been gaining increasing attention, especially to carry out tasks that can be done more efficiently and effectively with a team of agents in many different areas that can go from industry to even military. Such tasks can be: assembly, moving important or heavy objects, search and rescue, etc. Other than control and coordination of multiple agents, one of the central problems in this area is motion planning among multiple moving agents.

We are getting closer to a well-balanced motion planning to deal with dynamic or statics obstacles. $QVOC$ is the proof, showing how is possible to deal with these two different types of obstacles. The simulations in Gazebo and Qt Creator have shown how the algorithm can deal with a large number of obstacles in a limited space without creating oscillations, efficiently avoiding these obstacles and allowing the robot to reach the desired goal.

5.2 Recommendations

Unfortunately, due to the shortening of the internship, it was not possible to reach actual trials on real life. Even though the algorithm show how well the TurtleBot can move and maneuver, this might not be the same result on a real-life experiment. There are many factors that differentiate a simulation from a real environment. That is why even when in simulations the robot achieves the desired behavior, the experimentation in real world should start from having static and dynamic in different trials, see how the TurtleBot reacts to each one separately.

Bibliography

[1] RICARDO ANDREI RAYA ORTEGA. Robot navigation incrowds using queadrilateral velocity obstacles (qvo). 2017.

[2] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935. IEEE, 2008.

[3] Jamie Snape, Jur Van Den Berg, Stephen J Guy, and Dinesh Manocha. The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, 27(4):696–706, 2011.

[4] Steven M. LaValle. A differential drive. http://planning.cs.uiuc.edu/node659.html, 2006.

[5] Jamie Snape, Jur Van Den Berg, Stephen J Guy, and Dinesh Manocha. Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5917–5922. IEEE, 2009.

[6] Ros world, robot, sensor simulation, mapping, localization. http://grauonline.de/wordpress/?page$_id = 2769$.

[7] simulator_gazebo. http://library.isr.ist.utl.pt/docs/roswiki/simulator_gazebo.html.

[8] Achim J. Lilienthal and Kai Arras. Spencer people tracking. http://www.spencer.eu/index.html.

[9] David Naranjo. Opencv una biblioteca para el reconocimiento de objetos en imágenes y cámaras. https://blog.desdelinux.net/opencv-una-biblioteca-para-el-reconocimiento-de-objetos-en-imagenes-y-camaras/, Jan 2020.

[10] Tim Smithers. Autonomy in robots and other agents. *Brain and cognition*, 34(1):88–106, 1997.

[11] Turtlebot2. https://www.turtlebot.com/turtlebot2/.

[12] Robert Cong and Ryan Winters. How does the xbox kinect work. https://www.jameco.com/Jameco/workshop/howitworks/xboxkinect.html.

[13] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using the relative velocity paradigm. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 560–565. IEEE, 1993.

[14] Amichai Levy, Chris Keitel, Sam Engel, and James McLurkin. The extended velocity obstacle and applying orca in the real world. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16–22. IEEE, 2015.

[15] Timm Linder and Kai O Arras. People detection, tracking and visualization using ros on a mobile service robot. In *Robot Operating System (ROS)*, pages 187–213. Springer, 2016.

[16] RB Rusu and S Cousins. 3d is here: Point cloud library (pcl),[in:] international conference on robotics and automation. *Shanghai, China*, 2011, 2011.

[17] Joris M. Mooij. libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, 11:2169–2173, August 2010.