Master thesis

MU ENG INDUSTRIAL

Urban navigation of a mobile platform

REPORT

Author: Joan Grimalt Oliver
Director: Alberto Sanfeliu

Date: September 2019

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Abstract

This master thesis presents a method for 3D navigation of a robotic platform in urban environments.

In autonomous navigation, the robot must know the localization of all the environment obstacles, so an algorithm for obstacle detection is developed and tested using a LiDAR and a camera as sensors, comparing the data points' height. This detection focus on objects the robot could collide with in urban environments, including negative obstacles such as holes or stairs. The navigation and detection algorithms are all integrated in ROS (Robot Operating System).

The simulation and experimental results show the effectiveness of the algorithm to detect those obstacles, being successful with the LiDAR as a sensor in urban environments, but not sufficient robust enough for the camera when the navigation is done outdoors with high sunlight.

Contents

List of figures

1. Introduction

Online shopping is increasing, augmenting considerably the direct transport of products to the purchaser or user. As these services grow, so does the need of reducing the costs of last mile transport in urban environments. This could be achieved by means of automatizing the service using robots for doing the deliveries. It is a complex task, involving autonomous navigation, people recognition, incidental problems overcoming, etc.

This master thesis is part of a national investigation project carried out at the Institut de Robòtica I Informàtica Industrial (IRI). The title of the project is: “ColRobTransp: Colaboración robots-humanos para el transporte de productos en zonas urbanas” (human-robot collaboration for products transport in urban zones). The project consists in improving and advancing in the design of mobile robots that transport products or goods at the last mile in urban environments. A mobile robot will be given something to deliver like a package with a goal to the location of the receiver. Then, the robot must safely navigate to the goal, and once there, identify the person that is to receive the goods and approach smoothly.

The challenges that may encounter the robot during the performance of its task are numerous, so the project is simplified by dividing it in several parts that must work together and be tested both separately and all together. These parts are:

- Autonomous navigation in urban environments. The robot must reach the goal following a safe path, without risking its integrity or the nearby people. Possible situations that the robot may face are people crossing its way, unexpected street repairing, badly parked vehicles, stairs, ramps, floor unevenness, gaps, etc.

- Human-robot collaboration. To solve complex situations, the robot asks help to nearby people for finding an alternative path in case it is stuck, or also can ask for an obstacle to be removed in order to clear the path.

- Person identifying. Once at the goal, the robot searches and identifies the receiver of the goods.

- Approaching to the receiver. Once the person is identified, the robot approaches nice and smoothly, as a real people would, without sudden movements.

- Package delivery interaction. A PIN code is introduced by the receiver to have access to the delivery.

- Task managing and recovery in case of error.

1.1. Objectives

This master thesis is focused on the autonomous navigation part. As it is said before, the robot must reach a goal following a safe path. For this, obstacle detection and recognition is done by reading and interpreting the data that the sensors installed in the robot provide. So, the main objective of this thesis is the development of algorithms for perception and 3D navigation in urban environments. By 3D navigation, we mean that the navigation is not only above a flat surface, but also through up and down ramps.

The obstacles that are likely to appear on urban environments and that we aim to detect are the ones above the floor level like walls or people, but also the ones beyond floor level, like holes and stairs. Also, ramps must be detected and classified as traversable or not traversable, depending on its inclination.

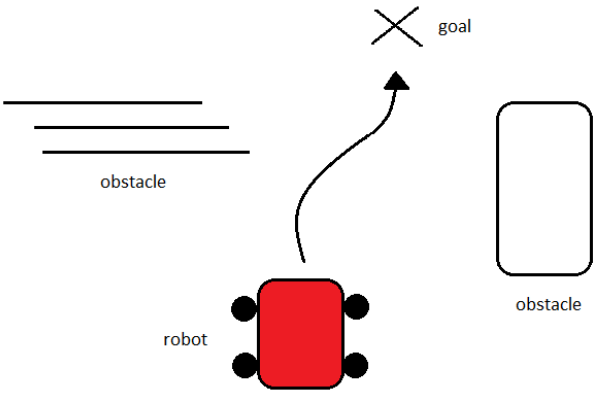


Figure 1-1.Autonomous navigation

2. Related work

A method for detecting negative obstacles like holes or gaps using a Kinect sensor is presented in [1]. The Kinect is a camera produced by Microsoft suitable for indoor environments. It provides a depth image that contains distance information of the scene with relation to the camera. In the article, they propose three algorithms that work together for detecting the location of non-traversable holes that the robot may encounter in its way. Once a hole is detected, the navigation algorithms treat it as an obstacle that must be avoided, so the robot changes its trajectory accordingly. The algorithms presented in the article are simple and can be implemented to operate on any camera that provides the depth image information.

The first algorithm, called farthest point method, detects cliff type holes, the ones that are far enough or deep enough that the sensor doesn't see anything below floor level. To detect this kind of holes, it looks for the farthest of points that are located at the floor level. If the sensor stops seeing anything at the floor level, the algorithm considers it is because there is a hole. In the article they consider all points between +3 cm and -3 cm in height on floor surface to belong to the floor level. The problem with this algorithm is that does not detect holes in the middle of the floor, so it has to work together with the other presented algorithm called virtual floor projection method. This later method uses trigonometry to project all points below floor level to the location of the hole. The final algorithm merges the information of the two others and produces an output that marks the location of holes obstacles and can be passed to the navigation algorithms. Summarizing, the method detects with a depth image camera both holes, the ones that are like a cliff and the ones in the middle of the floor.

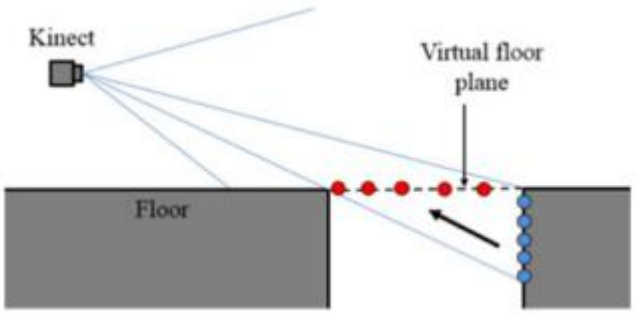


Figure 2-1. Floor projection method. [1]

In [2], an algorithm for incline detection and identification is presented. It is designed for being used on a powered wheelchair and using a depth image camera as the sensor, but it can be implemented on robots. The method not only detects the ramps in the scene, but also identifies its orientation, inclination and dimensions. The inclination information is necessary to know if the ramp is traversable, trying to navigate through a too inclined ramp could be very harmful for the robot.

What the algorithm does to identify ramps is taking as input the camera depth data, it calculates each point's normal looking to the neighbour points. By normal of a point, we mean the normal vector of a plane that fits the neighbour points. Then, all points with similar normal are segmented and then a RANSAC estimator is used to determine the equation of the plane. Then, the points of the ramp are reduced to its convex hull without loss of information, and information of the ramp is extracted, like the inclination and dimensions. Finally, the algorithm outputs this information to the navigation module so when a traversable ramp is detected it is not considered as an obstacle but a free path.

3. System overview

The robot is called Ana, and basically consists of a four wheel mobile platform with a fanless mini PC i7 computer inside to make all the calculations, perform the control of the robot, communicate with the outside, etc. It also has several sensors for obstacle detection during the autonomous navigation: a low cost IMU sensor, a 3D Lidar and a depth camera. It is powered with a custom 250 Wh motor battery and a custom 500 Wh payload battery, both from AMOPACK and with monitoring electronics, giving approximately 6 hours of autonomy.

Figure 3-1 shows how the whole system looks like. The laptop on top is for visualizing the sensor's data and to access the computer inside the robot.

Figure 3-1.IRI's Ana robot

3.1. Platform

Ana's mobile platform is a Pioneer 3 AT, a four wheeled, four-motor skid-steer base designed for exploration and autonomous navigation, very used in research. It can navigate through a wide number of different surfaces like inside floor, asphalt, mud or sand. It also can climb and go down ramps. For more complete specifications on physical characteristics, power specifications, mobility etc. see [3].

3.2. Sensors

Here, Ana's sensors are described briefly. These sensors calculate the distance from the robot to the surrounding obstacles. They are a LiDAR sensor and a depth camera.

3.2.1. LiDAR

LiDAR is an acronym for light detection and ranging. The sensor emits laser beams and sees the time it takes to return to the sensor to calculate the distance of an object to the sensor, so we have information of where are the objects and surfaces on the surrounding environment of the robot. This technology is used in diverse fields as robotics and autonomous driving, land surveying and cartography, spaceflight, meteorology etc.

The provided LiDAR sensor with the robot is the Velodyne LiDAR PUCK (VLP-16). It is a 3D sensor with 16 channels, that is, 16 emitter/receiver pairs. These channels are disposed vertically, each one with a fixed inclination, covering a vertical field of view of 30 degrees, from -15º to +15º as can be seen on Figure 3-2. As the sensor rotates horizontally at between 5-20 Hz, a full 360 degrees representation of the environment is obtained by with data consisting of approximately 300,000 points per second. It has a maximum sensing range of 100 m. and a minimum range of 0.45 m.

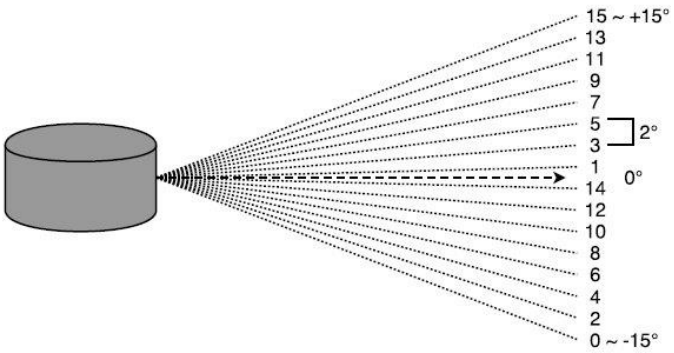


Figure 3-2. Vertical field of view. [4]

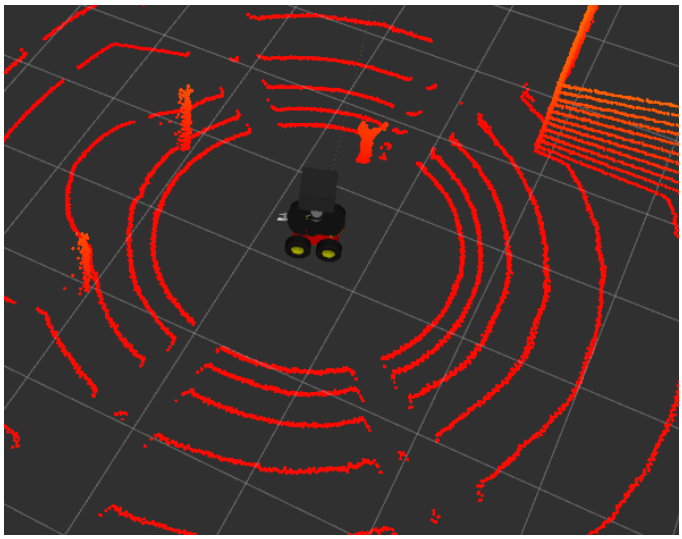


Figure 3-3. Point cloud obtained with the VLP-16.

Figure 3-3 shows a set of points obtained with the sensor. It is called a point cloud, and creates a sampled 3D representation of the surveyed environment. Each point holds the information of x, y and z coordinates referenced to the sensor frame.

In our implementation, the sensor is located on top of the platform, more or less on the centre. Because of this localization, there is a dead zone at ground level at a distance lower than about 1.4 m. In other words, the sensor is not able to detect holes that are a distance closer than 1.4 m, as the laser that is 15 degrees down can't reach this zone. This can be seen in Figure 3-3, as the circles surrounding the robot that correspond to the detection of the floor, appear at a certain minimum distance. This can be a problem, as a hole between the robot and this minimum distance for seeing the ground cannot be detected with this sensor. However, positive obstacles like a person can be detected closer than this distance, as the other laser beams in different angles can reach the obstacle.

Also, in Figure 3-3 it can be seen that there are four void zones, where the ground should be detected. This is because of the bars that hold the metallic box on the robot (Figure 3-1). Those bars block the sight of the sensor, but it is not very problematic, as these dead zones are not on the direction of movement of the robot and are small. The problem with the bars is they introduce noise on the measurements next to those regions because of reflections of the laser on them. This will be tackled later.

More detailed features and specifications of the VLP-16 can be found on its datasheet, available at [5]

3.2.2. Depth Camera

A depth camera is a sensor that provides distance information. It takes images of the scene composed by pixels, and also calculates the distance from each pixel to the sensor. From this, we can have a point cloud as the obtained with the LiDAR, but in this case the resolution is much larger.

The Stereo vision consists on obtaining images of the same scene from two different vantage points. Then, depth information can be extracted by comparing the relative position of the same object in the two different pictures. The quality of the results depends on the presence of points that can be matched between the two images. The presence of textures improves this matching.

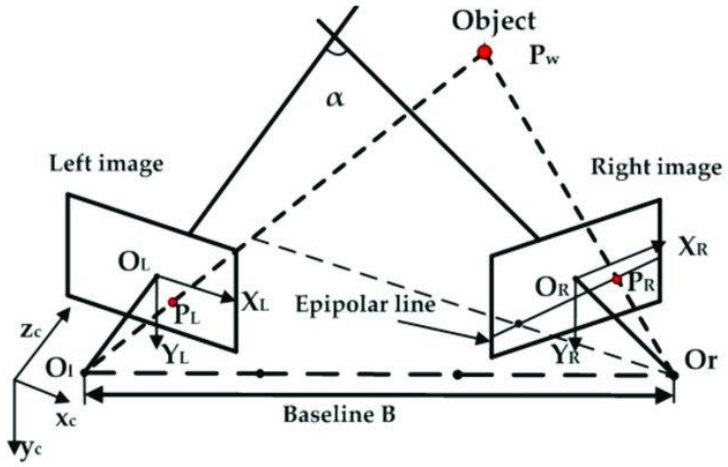


Figure 3-4. Stereo Vision [6] .

The available cameras that have been tested on the robot for this project are the Intel's Realsense D435, Zed stereo camera and Asus Xtion Pro Live.

a) Realsense D435 [7] b) Zed Stereo [8] c) Asus Xtion Pro Live [9]

Figure 3-5. Depth cameras

The Realsense camera uses Active Infrared stereo technology for extracting depth information of the scene. It consists on the stereo vision explained above with the addition of an infrared projector. This projects a static infrared pattern to improve depth accuracy in scenes with low texture, making it easier for the algorithm to do de matching [7]. Regarding to the specifications, it can be used both indoors and outdoors, and the presence of sunlight improves the performance. However, as it is explained at the real experiments chapter, the depth sensing suffers from bright light reflections in sunny days.

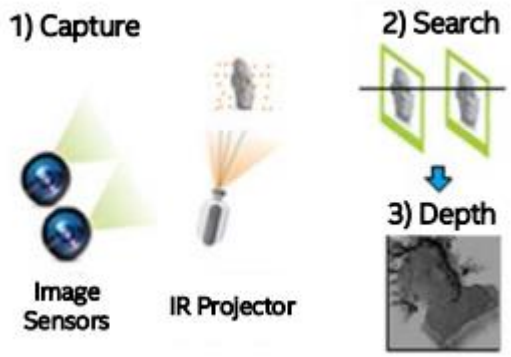


Figure 3-6. Active Infrared stereo [7]

The Zed camera is a passive stereo camera, that means it depends on the light of the environment, it doesn't use external light projectors. A passive stereo vision camera should work well in sunlight, but it has poor performance in low light and in non-textured scenes.

Finally, there is the Asus Xtion Pro live, another depth camera designed for indoors.

Table 3-1shows a comparation of the three mentioned cameras specifications.

Table 3-1. Depth cameras comparation

	Realsense D435	**Zed Stereo**	**Asus Xtion Pro Live**
Environment	Indoor and outdoor	Indoor and outdoor	Outdoor
Depth technology	Active infrared stereo	Passive stereo	Active infrared stereo
Field of view	85.2º	110º	58º
Minimum range	0.11 m.	0.5 m.	0.8 m.
Maximum range	10 m.	15 m.	3.5 m.

The camera is localized on the front of the robot, pointing to the floor. This way, we can have the close to the robot floor data missed by the LiDAR.

3.3. Software

The Ana's computer runs Ubuntu Xenial, and uses ROS Kinetic for operating the robot.

The Robot Operating System (ROS) is an open-source middleware specially designed for robotics, a standardized software framework for robot software development. Although it isn't exactly an operating system, it provides its services like hardware abstraction, hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management [10].

This allows us to easily integrate different hardware devices and software on a large number of applications without having to worry about each fabricant making different software. Code in ROS is organized in packages and stacks, making very easy its distribution and sharing with the ROS community. This is a very powerful feature of ROS, as we are able to reuse code made by someone else, saving a lot of time.

3.4. Robot frames

All the data taken with the sensors must be referred to a determined frame to have any sense. Each sensor has its own reference frame, normally located at the centre of the sensor, and all the data about the environment is referenced to that frame. Different parts of the robot also have its own frames so we can relate the position of anything to them. There are quite a lot of different frames on the system.

As we have mentioned before, both the camera and LiDAR presents the information of the scene with point clouds where each point has its XYZ coordinates. These coordinates are referenced to each sensor frame, so it has to be taken account the position or the orientation of the sensor is changed, the values of the coordinates of each point will change.

Because of that, a transformation to a fixed frame in relation to the robot has been applied to all the data coming from the sensors. The frame chosen is located at the robot's footprint, so a point located at the floor level will have a z coordinate close to 0. Ana's frames can be seen on figure Figure 3-7. Red green and blue colours are XYZ frames respectively.

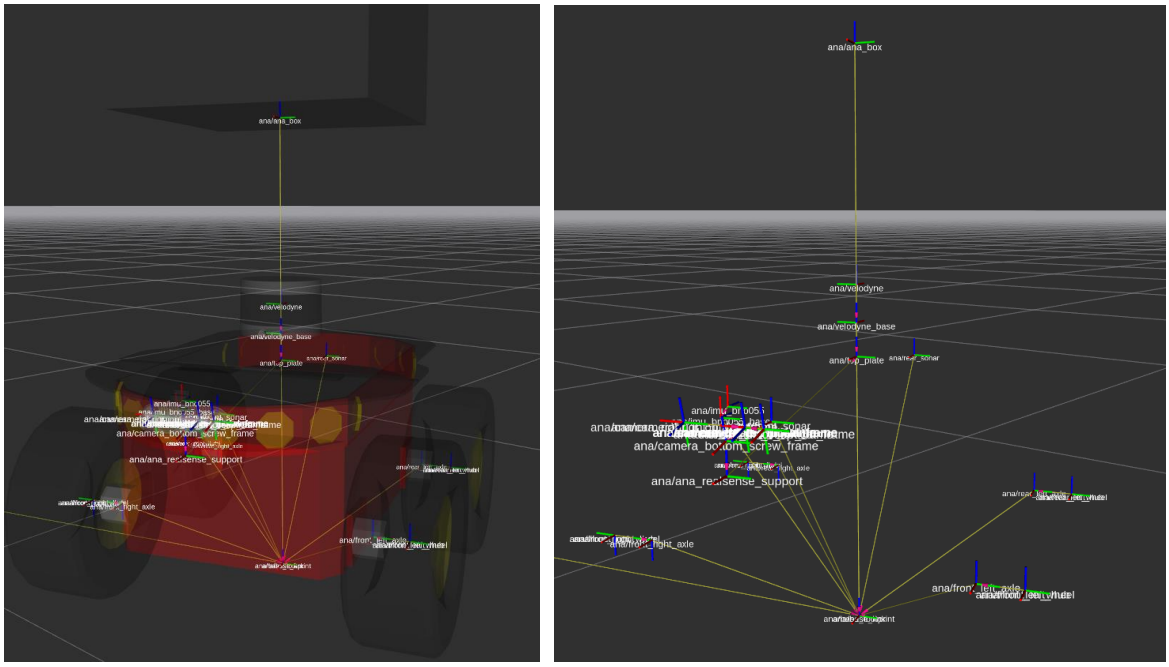


Figure 3-7. Robot's frames

4. Algorithm

This chapter presents an algorithm for detecting obstacles, including holes. It does so looking the height of near points in the sensed point clouds and comparing them. It detects either positive or negative obstacles, considering as positive obstacles the ones above ground level and negative ones those below ground level, like holes or downstairs. Also, it detects ramps and marks them as obstacles if its inclination is too big for the robot to safely traverse it. It can be used with data obtained from LiDAR sensors and depth image cameras, with just a few adjustments depending on the kind of sensor.

Figure 4-1. Positive and negative obstacle

4.1. General idea

For detecting holes, the proposed idea is simple. Comparing the height or z coordinate of neighbouring points in the point cloud, if the difference is higher than a gap height threshold, an obstacle is found.

If the sensor used is a LiDAR like the VLP-16, the first step is filtering the sensor data to reduce the number of points, so less calculation is needed, thus making the algorithm faster.

This filtering consists on removing points that are outside a determined window around the robot. In our case, the window's size chosen is the same size of the local cost map. The sensor has a wide range of detection, but for navigation purposes it is ok to only consider points that are not so far. Then, as each point belongs to one of the 16 rings of laser beams the sensor has, we can detect a positive or a negative obstacle by comparing each point's z coordinate with its two neighbours on the same ring and the ones on the same position but in next and previous rings.

In Figure 4-2 it can be seen a zoomed section of the point cloud obtained with the LiDAR sensor. It's easy to visualize that these points belong to three different laser rings, each of

them with a different angle, as it was commented in section 3.2.1. So, for detecting an obstacle, a recursion is doing for each point, comparing it with the points on its side and the corresponding point in previous and next rings. If there is a difference in height between the compared points that is larger than a certain threshold, an obstacle is found, being either a positive obstacle or a negative one.

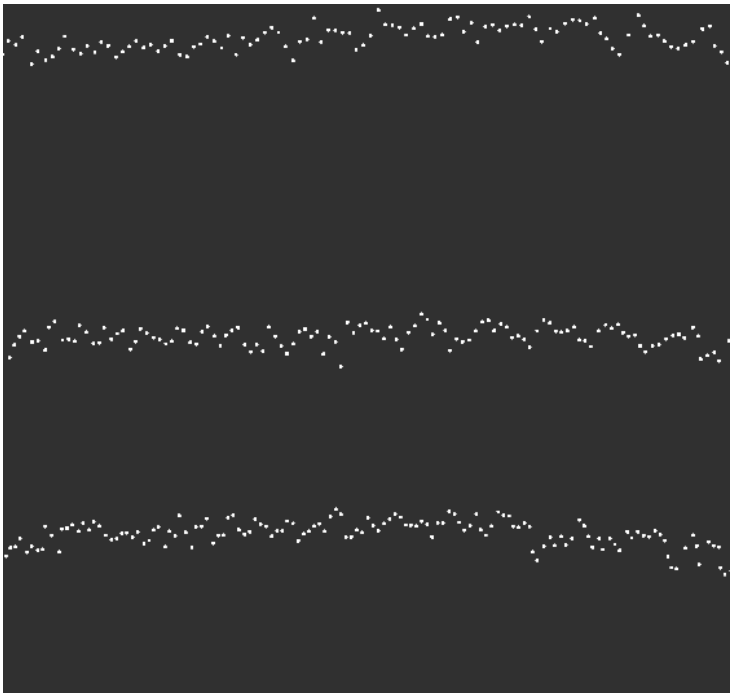
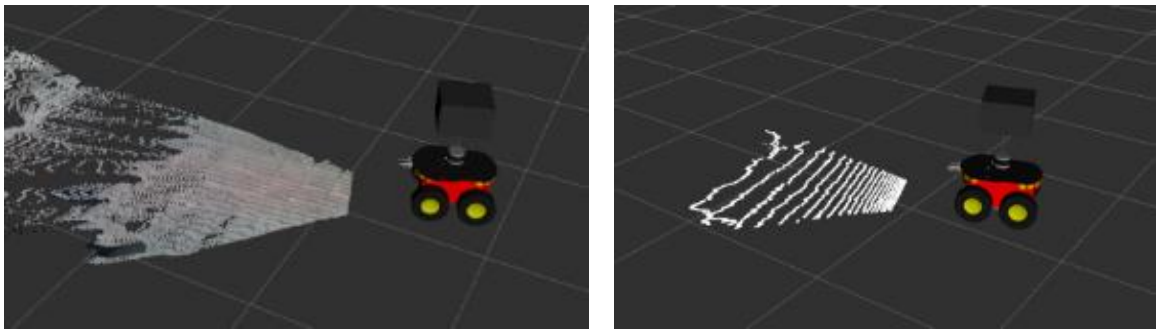


Figure 4-2 Neighbour points

On a camera, we have no longer a 360º field of view, but in exchange, the resolution is much bigger. Also, points in the point cloud are not organized in rings, rather than rows and columns. For the down sampling, we remove some rows so we have in a way a similar point cloud like the LiDAR one, as it is shown in Figure 4-3. Then, as in the LiDAR case, we compare the z coordinate of neighbour points, the ones than are in the same row and the ones that are on the next and previous rows.



a)Camera point cloud *b) Down sampled point cloud*

Figure 4-3. Down sampled camera point cloud

So far the negative and positive obstacle detection is done, but a difference in the z coordinate can also be due to a positive or negative ramp. If it is the case, we would be considering a ramp as an obstacle, but we want the robot to go through the ramp if it is traversable. So, while comparing the z coordinates of neighbour points, we also calculate the angle these two points form and obtain the inclination value. If the obstacle is a hole or stairs, the inclination will be bigger than the inclination threshold, but if it is a low inclined ramp, it won't be considered as an obstacle.

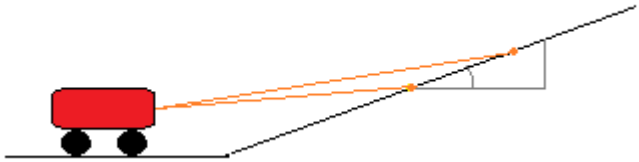


Figure 4-4. Ramp detection

Finally, once all obstacles are detected, the algorithm must output the location of those obstacles to the navigation module, so the path planner generates a feasible path to a set pose goal.

4.2. Pseudo code

Here it is the pseudo code for the proposed algorithm. Each point is compared with its two neighbours on the side and the one in opposite direction of the robot.

Algorithm 1. Obstacle detection

1. Get point cloud and downsample.
2. **for all** points in cloud **do**
3. calculate ramp inclination
4. **if** (point.z - neighbour.z > gap_threshold **and** incline>max_incline) **then**
5. mark negative obstacle
6. **end if**
7. **if** (point.z - neighbour.z < - gap_threshold **and** incline>max_incline) **then**
8. mark positive obstacle
9. **end if**
10. **end for**

4.3. Implementation

The algorithm has been written in C++ inside a ROS node. As we mentioned before, ROS allows us to easily integrate different parts of the robot, like the sensors or the navigation algorithms. A ROS node is a process that performs computation and communicates with other nodes. For this communication, we don't have to worry about how nodes made by someone else have been programmed, only the type of the information they exchange is needed if we want to use it.

In this case, our node is continuously running, doing calculations for detecting the obstacles. It takes the data from the sensors and outputs the location of the detected holes to the navigation module.

The overall scheme can be seen in Figure 4-5. Each arrow represents a communication between nodes, indicating the type of information. These types are built-in ROS msg types.

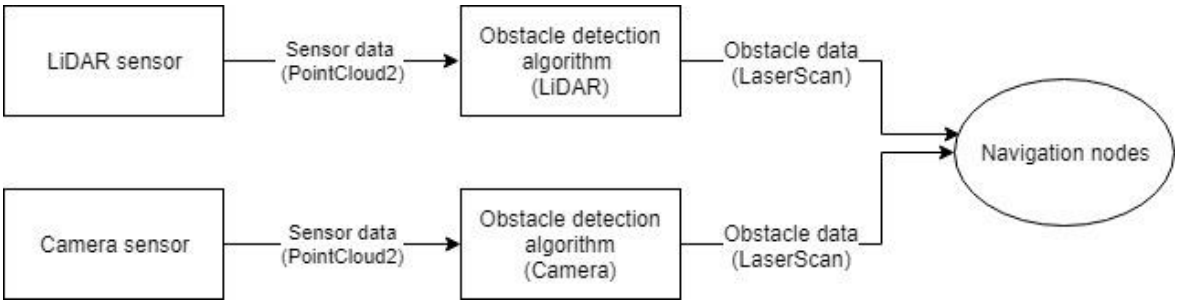


Figure 4-5. General ROS nodes scheme

So, each sensor sends the sensed information to the obstacle detection algorithm in form of point clouds. Specifically, the ROS message type is sensor_msgs/PointCloud2. Once the obstacles detection is done, the output obstacles information is passed in form of the built-in ROS message type sensor_msgs/LaserScan.

The scheme in Figure 4-5 is a general simplified of the actual system simplified, because for example it doesn't give details about all the communication between navigation nodes and doesn't show extra nodes needed for the Ana robot application. Navigation stack will be discussed in next chapter.

As it was commented in section 3.2.1, the metallic box of the robot is supported with 4 bars that block the vision of the LiDAR sensor located in the middle. This not only cause to have blank information on certain parts of the scene, but also to have displaced points in the point cloud that are in the vicinity of the blocking bars due to light reflections on those bars. Because of that, those points are filtered in a node before the point cloud is passed to de detection algorithm.

Regarding to the C++ implementation of the holes detection algorithm, for process the information received in form of point clouds, the open-source library PCL is used. PCL provides algorithms and classes written in C++ for point cloud processing. We create a pcl::point_cloud object from the point cloud received, do all the calculations and filterings with this new object and finally create a ROS msg Laser scan type object to send to the navigation nodes the obstacles information.

A laser scan type of message is used for sending the obstacles information because less data is needed than a point cloud. In a ROS msg Laser scan, the data is sampled by the number of readings, each one separated a fixed angle, and we only have to indicate the range the obstacle is to the robot for each sample. For navigation purposes, each laser scan sample is initialized at a range larger than the minimum obstacle range for marking an

obstacle and at a lower range than the minimum obstacle range for clearing. This will be discussed later on next chapters.

Also, for the hole detection, we explained the points comparison, but we didn't mention how to do it. Organization of the points in point clouds can vary depending on the sensor, so we have implemented the neighbour point's comparison by calculating angle, step and range for each point in the point cloud, as it is done in [1].

$$Angle = atan2\frac{y}{x} \qquad \textit{(eq. 1)}$$

$$Step = \frac{Angle - Angle_{min}}{Angle_{inc}} \qquad \textit{(eq. 2)}$$

$$Range = \sqrt{x^2 + y^2} \qquad \textit{(eq. 3)}$$

Atan2 refers to the arctan function that return a value between $-\pi < \theta < \pi$, and $\mathrm{Angle}_{\mathrm{min}}$ and $\mathrm{Angle}_{\mathrm{inc}}$ are parameters.

So, each point is given a step and we compare points with the same step. By sorting all points by its range value, iteration of all points in the sorted point cloud is done, and then the z coordinate of the point is compared with the z coordinate of the previous point with the same step value. Also, all neighbours located at the point sides are found by looking to the ring value in the case of the LiDAR and to the point index in the case of the camera.

Finally, laser scan data is filled. The laser scan has to have the same number of samples as the number of steps. If an obstacle is found, the laser scan sample corresponding to the step number is equal to the range value of the obstacle found.

Also, all configurable parameters like threshold values, filter values, modes of operation etc. are written as a dynamic reconfigure parameter. This allows us to change the value of the parameters while the node is running, and prevents from having to recompile the code each time we want to change it.

5. Navigation

In this chapter we describe in short the navigation module. For the Ana robot implementation, the navigation module used is the provided by the ROS navigation stack. It takes the obstacle information from the obstacle detector algorithm and a desired goal pose to give commands to the mobile base to reach that goal following a safe path. The overall scheme of the navigation stack is shown in Figure 5-3.

In autonomous navigation, the robot keeps track of the obstacles it has seen by building a map. This is called mapping. The map can be built from scratch while navigating, but it also can be previously added to the navigation stack if it is known previously. It is also continuously been actualized with new information from the sensors.

Besides, the robot must know it's localization in relation to the map as it moves. This localization can be done with the odometry information for estimating the position of the robot in the map knowing the starting location. However, localizing the robot relying only on odometry is error prone, as there are many sources of error caused for example by wheel sliding. This error is being accumulated while the robot is moving, and the error turns unacceptable.

To avoid this error in localization, the ROS navigation stack has an implementation of the AMCL (Adaptive Monte Carlo Localization) algorithm. It uses a particle filter to track the pose of a robot against the map [11]. This way, simultaneously localization and mapping can be done at the same time. This is also known as SLAM.

Regarding to mapping, in the ROS navigation module we use there are two maps, the global and local cost map. A cost map is a map defined as a sized grid, marking each cell with a cost. Depending on the cell cost, it is considered as occupied, free or unknown space. Then, the path planner uses this cost map to find a feasible path and avoiding occupied cells, thus avoiding collision or dangerous situations. The cost on each cell is determined by the sensors information, inflating the values on cells that contain obstacles. In other words, cells with obstacles have very high cost, and cells on their neighbourhood decrease its cost with distance.

For building this occupancy grid, the ROS costmap_2d package is used. This means that the cost map is projected on a 2D grid, the height of the obstacle doesn't matter. In our implementation all detected obstacles are marked with a laser scan, so we don't care about the height of the obstacle anyway, with its xy position it is enough.

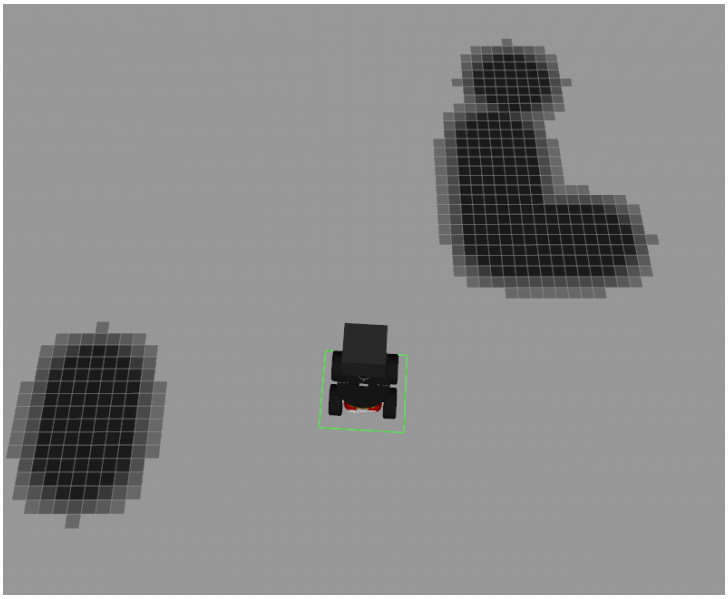


Figure 5-1Cost map grid

The configuration of the cost map grid filling is done with plugins. Some parameters that can be configured are, among others: the maximum and minimum distances to take account an obstacle, the relations between layers, the inflation ratio … but most important parameters are the ones regarding the specification of the marking and clearing of obstacles.

Marking an obstacle on the cost map grid means that the cost map subscribes to the obstacle topics published by the detection nodes and change the cost value of the corresponding cells. On the other hand, the clearing operation consists of removing an obstacle from the cost map. It is done by raytracing from the sensor to the detected obstacle. This clearing is essential if the robot moves, especially if there are mobile obstacles like for example a crossing person, because cells which no longer include an obstacle must be considered as free.

This marking and clearing procedure can be better understood by looking to Figure 5-2. In case a), an obstacle is detected in front of the robot, so the marking is done by assigning a cost to the corresponding cells of the cost map, considering them as occupied.

Then, in case b), the obstacle has moved and the corresponding cells are also marked. Notice that the previous marked cells remain considered as occupied, despite the fact that the sensor no longer detects an obstacle there. This is done for safety reasons, it is not sure that there is no longer an obstacle on that first position.

Finally, in case c), the obstacle moves again, this time behind its first position. Again, the corresponding cells are considered as occupied, but the cells that were marked in case a) have been cleared. This clearing is done by raytracing. It means that only previously marked cells that are not currently being marked are cleared if its location is in between a ray from the sensor to another obstacle.

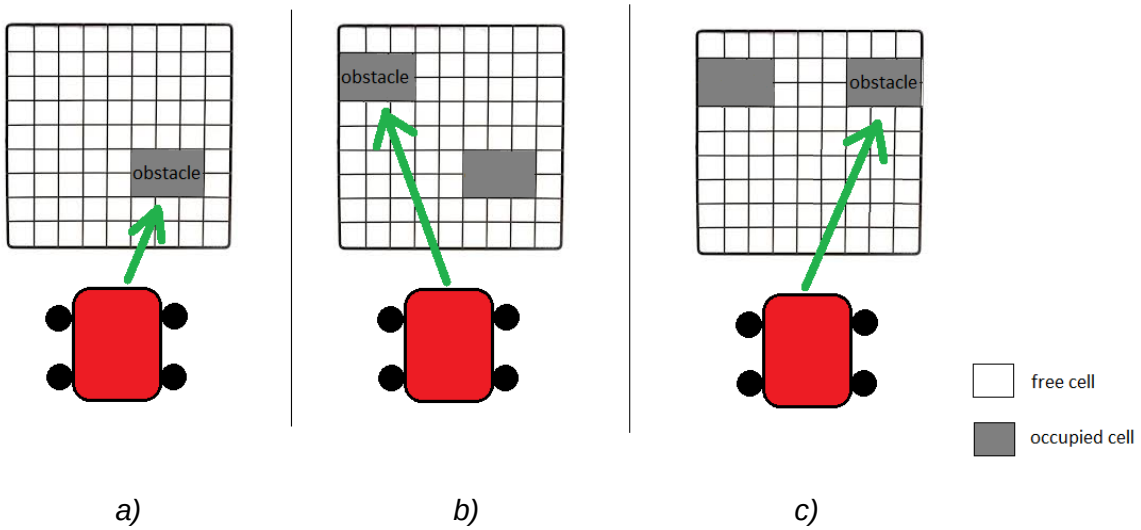


Figure 5-2 Marking and clearing

In our configuration, two obstacle layers are used. One is for obstacle detection with the LiDAR and another for detection with the camera. It is done this way to avoid that one layer clears the other's detected obstacles in case it doesn't see them because of the dead zone.

The global planner generates a long-term plan, regarding the global cost map and the desired goal. On the other hand, local planner makes short-term plans, generating velocity commands to the mobile base.

Finally, the move_base node links together the global and local planners, and maintains the local and global cost maps. Also, it provides recovery behaviours in case the robot is stuck. In case those recovery behaviours are needed, the system may perform an action or a sequence of them, as rotation of the robot, clearing obstacles of the cost map in a controlled way, aborting the attempt, etc.

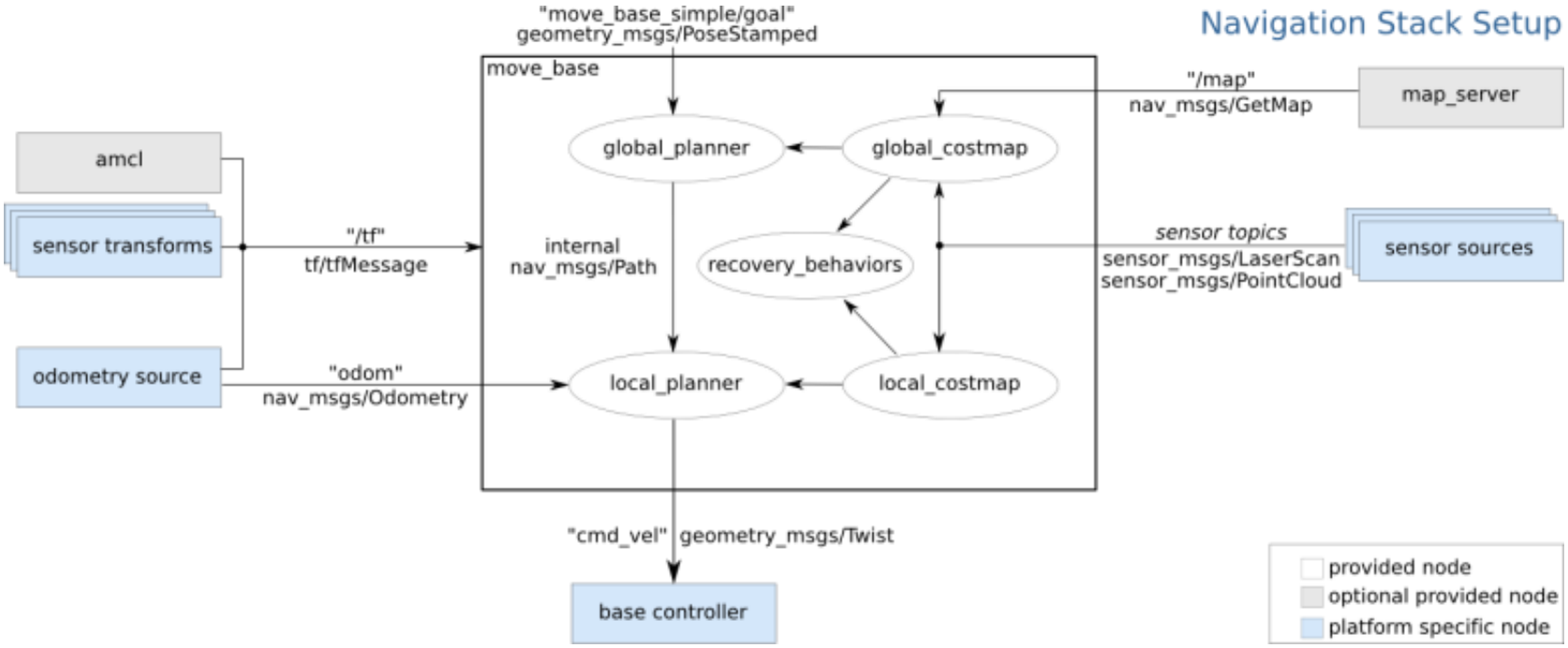


Figure 5-3. ROS navigation stack.[10]

6. Simulation

This chapter shows the results of simulating the whole system in a Gazebo simulated environment.

6.1. Detection algorithm simulation

For testing and developing the detection algorithm, several Gazebo worlds with ramps, holes and stairs were created. An example is presented in Figure 6-1, where it is shown a launched Gazebo environment with the simulated Ana robot. In the next figures, it can be observed the visualization of the data received from the sensors when the robot is at the position shown in Figure 6-1. The simulated sensors on the robot are the LiDAR VLP-16 and the depth camera Realsense D435.

In Figure 6-2, the white points are the point cloud obtained with the LiDAR. The ones that form circles near the robot correspond to the ground, while the walls and the box are easily identified by comparing Figure 6-1 and Figure 6-2. The red points correspond to the laser scan generated by the obstacle detection algorithm. As it was commented in previous chapters, the detection of obstacles is indicated by this laser scan. So, the obstacles detected are the box on the left of the robot, the walls on its front and the ramps behind it. Notice that the ramps situated at the back of the robot are detected as obstacles, meaning it's a traversable path. This is because the ramps have larger inclination than the incline threshold, so they are considered as an obstacle. Finally, another obstacle is detected, the hole on the front. The algorithm compares the z coordinate of the points and detects a sufficient leap for considering an obstacle.

Another interesting test is to position the robot close to the stairs. As it can be seen in Figure 6-3 and Figure 6-4, the LiDAR is not able to detect this obstacle at its entirety, as only its sides are marked with the laser scan. This is because the LiDAR sensor has the mentioned dead zone. In this case, the obstacle is detected with the camera, and the detection can be visualized with a yellow laser scan.

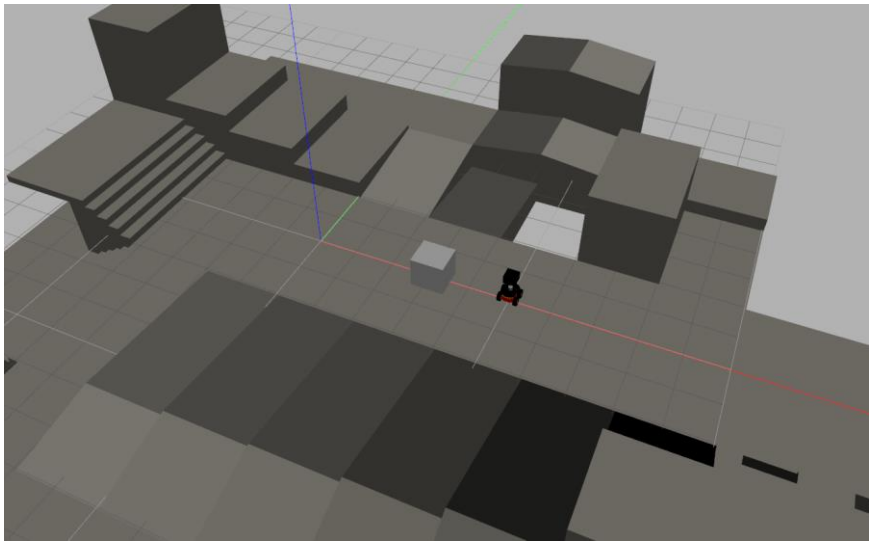


Figure 6-1.Gazebo ramps world

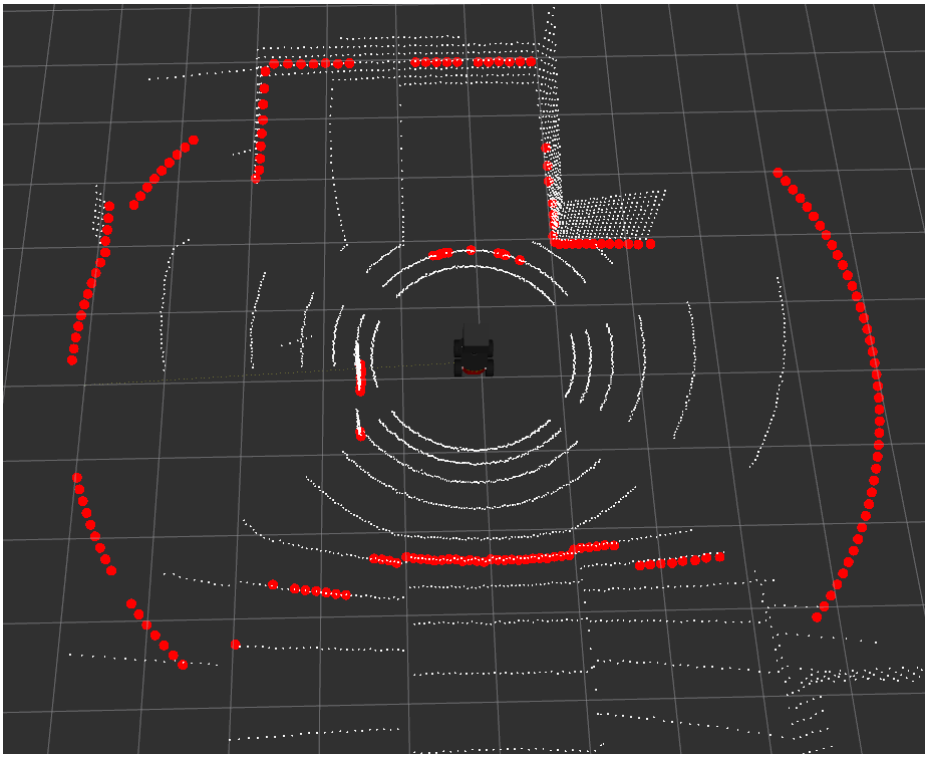


Figure 6-2 Obstacles detected with LiDAR

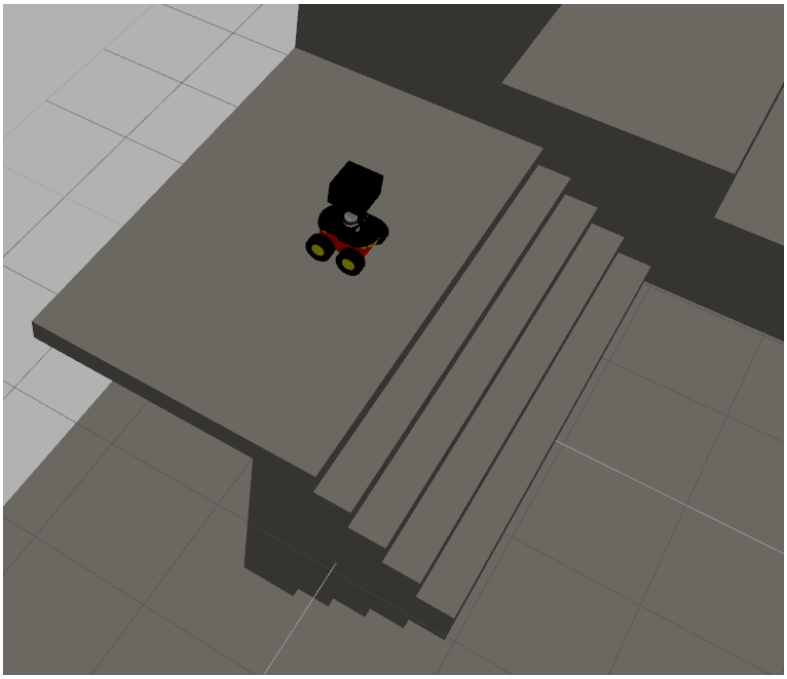


Figure 6-3 Ana in front of down stairs

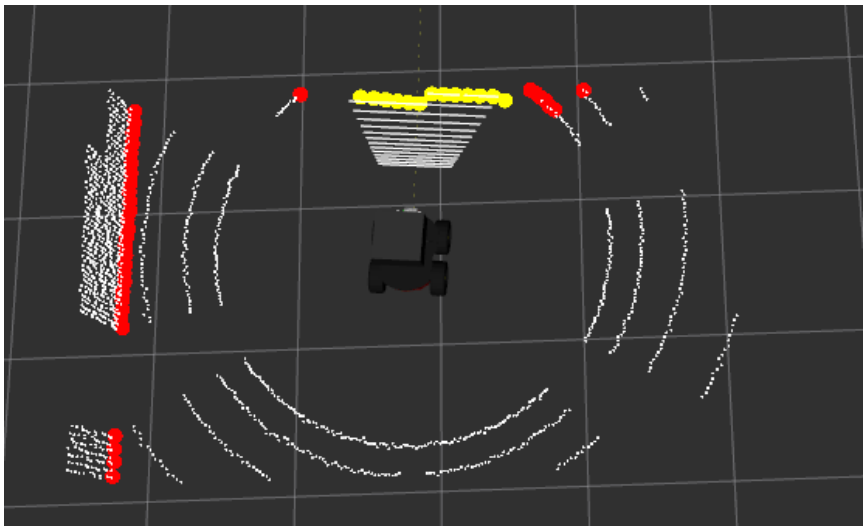


Figure 6-4 Stairs seen with the camera but not with the LiDAR

6.2. Navigation simulation

Once the obstacle detection worked in simulation, the navigation stack was launched for simulating the autonomous navigation of the robot. For this, a series of destination goals near the robot were given, to see the detection of obstacles in motion, the correct filling of the cost map and the new planning of a path to reach the desired goal.

In Figure 6-5 right, it is shown the inflated obstacles detected with the LiDAR. In Figure 6-5 left, the same Figure 6-2 is repeated for easier comparison. As it can be seen in Figure 6-5 left, laser scan samples that don't represent an obstacle remain at its initialization range. Because the minimum range for the cost map to mark an obstacle is set lower than this initialization range, no obstacle is marked, as it is expected. This is done for allowing proper clearing of moving obstacles.

In Figure 6-6 it can be seen that the robot plans a path to the desired goal. Instead of going straight to the goal, it follows a path that avoids obstacles.

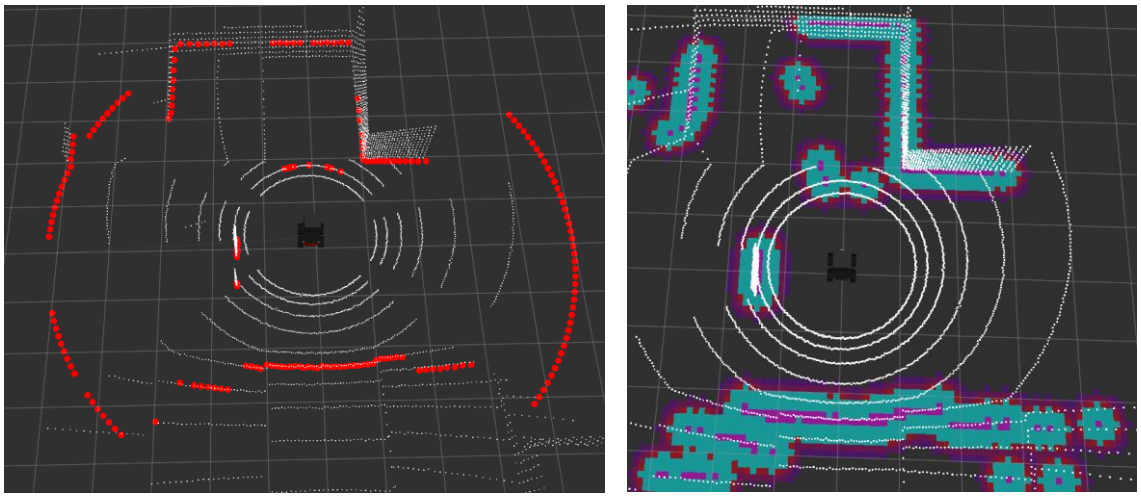


Figure 6-5 Obstacles in local cost map

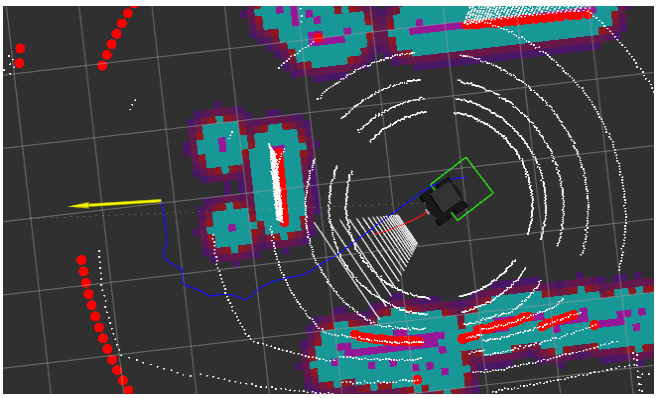


Figure 6-6 Path to reach the goal

Finally, another simulation is shown, this one in a gazebo world representing Campus Nord. On this new simulation, it was used a map of the environment as a static map, to allow the setting goals far from the robot.

The navigation worked as expected, detecting correctly problematic obstacles like the stairs and climbing and descending ramps without problem. However, there was a location were the robot seemed to have a little trouble to create the path. It consisted in two ramps converging in opposite directions and the given map didn't exactly match the gazebo world. In this case, setting again the goal was needed.

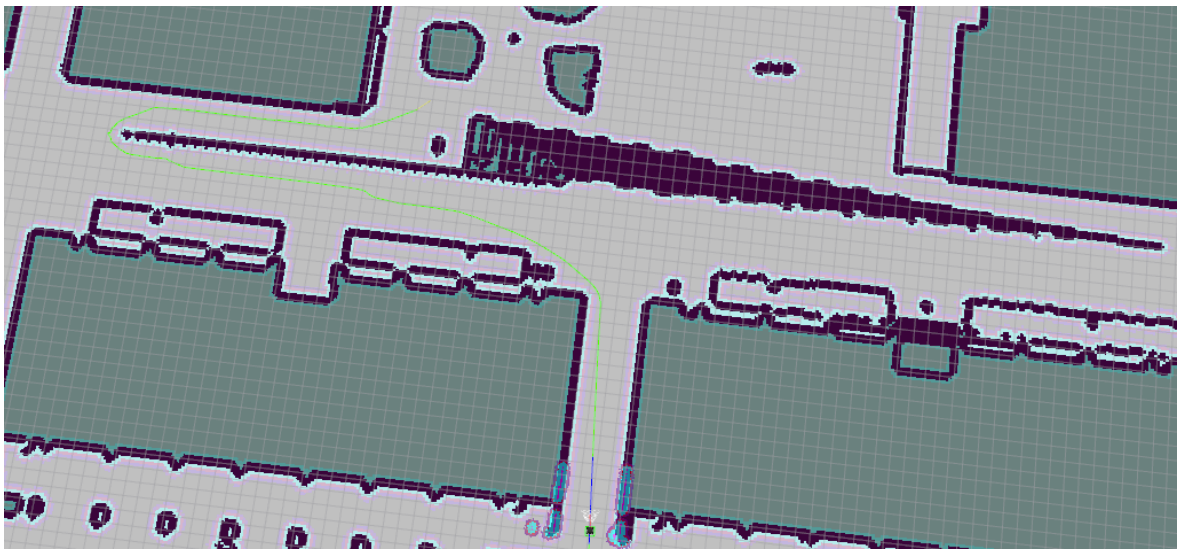


Figure 6-7 Campus Nord simulation

7. Experiments

This chapter shows the results of experiments done with the real robot outdoors. The main difference with the simulated environment is the presence of errors caused by the noise of the sensors.

As to obstacles detection, the algorithm works fine. It was tested in several situations with common obstacles in urban environment, like walls, people, holes and ramps. In all situations, it marked the obstacles correctly.

In Figure 7-1, Figure 7-3 and Figure 7-2, it is shown the obstacles detection in a case when the robot is near the edge of the sidewalk. Figure 7-2 shows the output of the detection algorithm in form of two laser scans, one for each sensor (red for the LiDAR and yellow for the camera). As it was mentioned, the VLP-16 LiDAR provides 360 degrees of vision except the occlusions due to the bars, and the camera only sees the ground in front of the robot.

Results are as expected, all obstacles are detected with the LiDAR except the hole, because it is inside the dead zone of the sensor. So, this hole only can be detected with the camera.

Figure 7-1 Edge of a sidewalk

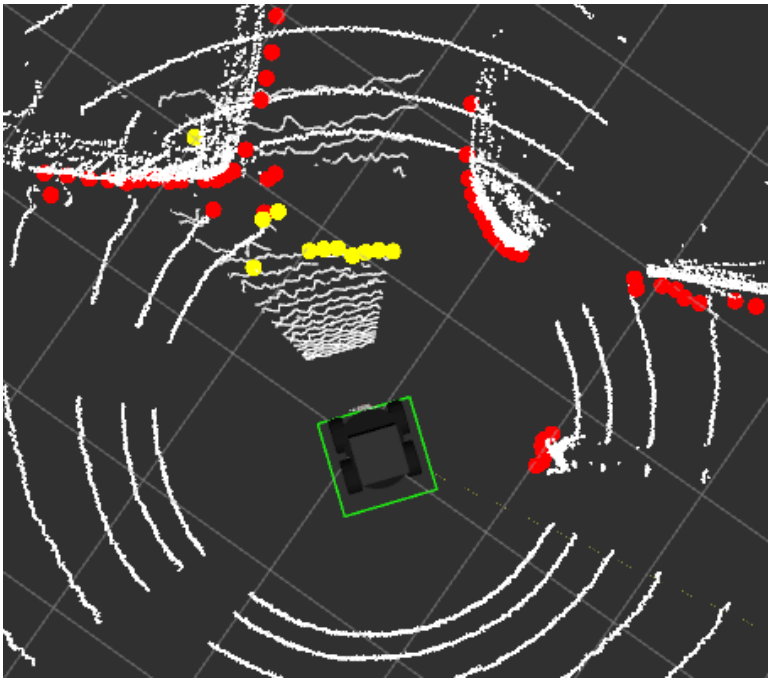


Figure 7-2 Obstacles detection. Hole

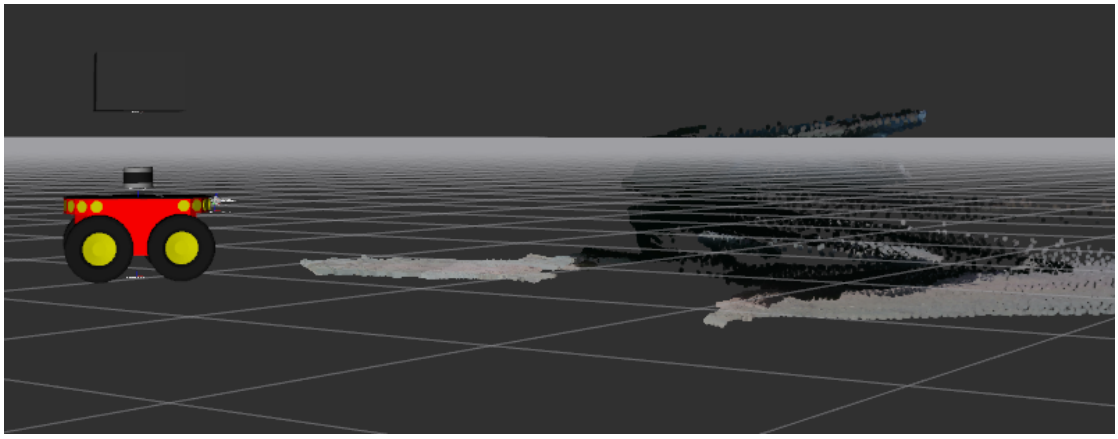


Figure 7-3 Point cloud of the hole seen by realsense camera

Another interesting experiment is the shown in Figure 7-4 and Figure 7-5. It can be seen that the ramp is not marked as an obstacle, only the two pylons at its sides.

Figure 7-4 Ana in front of a ramp

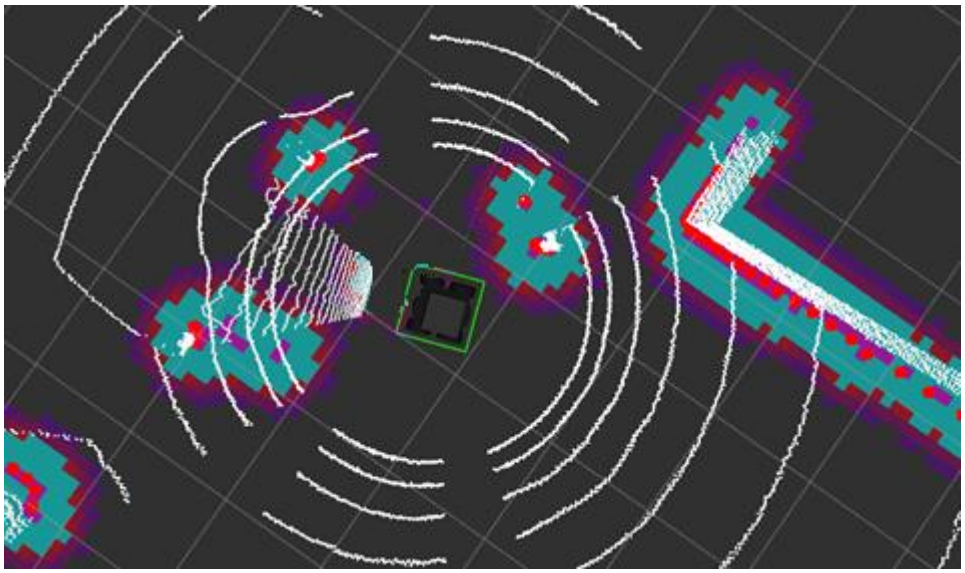


Figure 7-5 Obstacle detection. Down ramp.

However, although all obstacles were detected, there is a downside that makes the system not robust enough for autonomous navigation. It have been detected the presence of incompatible noise in the camera depth data with this approach. This happens due to bright reflections of light in a sunny day.

As it can be seen in Figure 7-6 and Figure 7-7, the camera sees the ground, so no obstacle should be marked. However, the yellow laser scan corresponding to the detected obstacles by the camera marks some locations as an obstacle because of this noise. This makes it impossible to navigate with the camera, because these false positive detected obstacles located in front of the robot makes that it gets stuck very often.

Figure 7-6 Ana on the sidewalk.

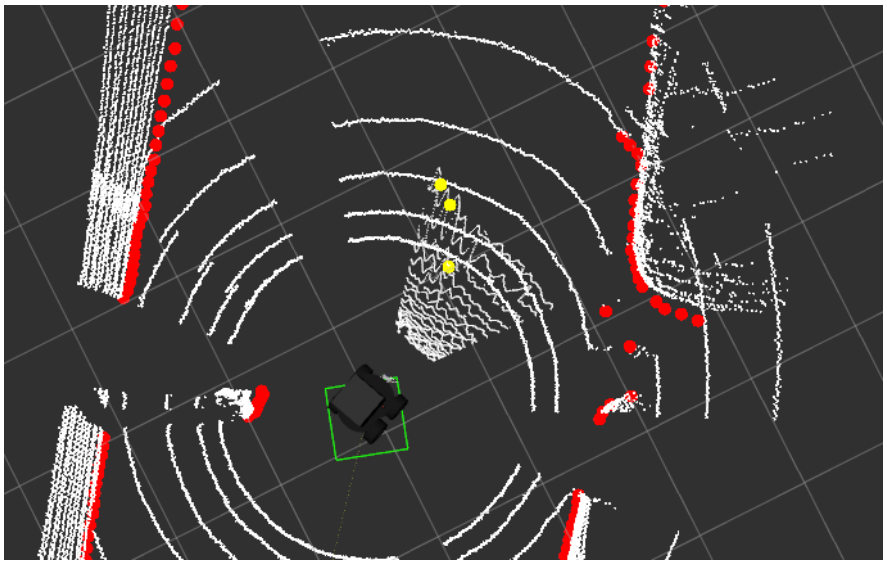


Figure 7-7 False positive obstacle detection

The camera was unable to get correctly depth values in regions of the scene where there are bright light reflections, very common in a sunny day. The three different cameras presented in chapter 3 were tested both indoors and outdoors, and none of them were accurately enough in an outdoors environment.

The point cloud obtained with the realsense infrared camera and zed stereo camera had displaced points on the mentioned bright regions, and they were located on an unpredictable location above or beyond the actual location. This makes the detection algorithm find an obstacle were there isn't. The use of an outlier removal filter and voxel grid filter to eliminate the disturbances in obstacle detection due to the presence of such displaced points turned out to be good enough.

On the other hand, with the Asus Xtion camera the point cloud obtained was very accurate indoors and outdoors in the shade, but in an outdoors environment with sun the camera wasn't able to get depth information of the scene.

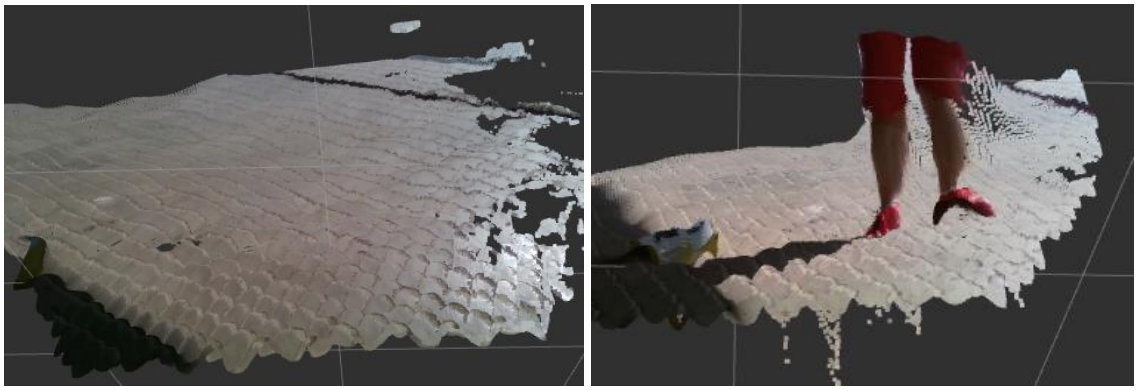


Figure 7-8 Realsense noise

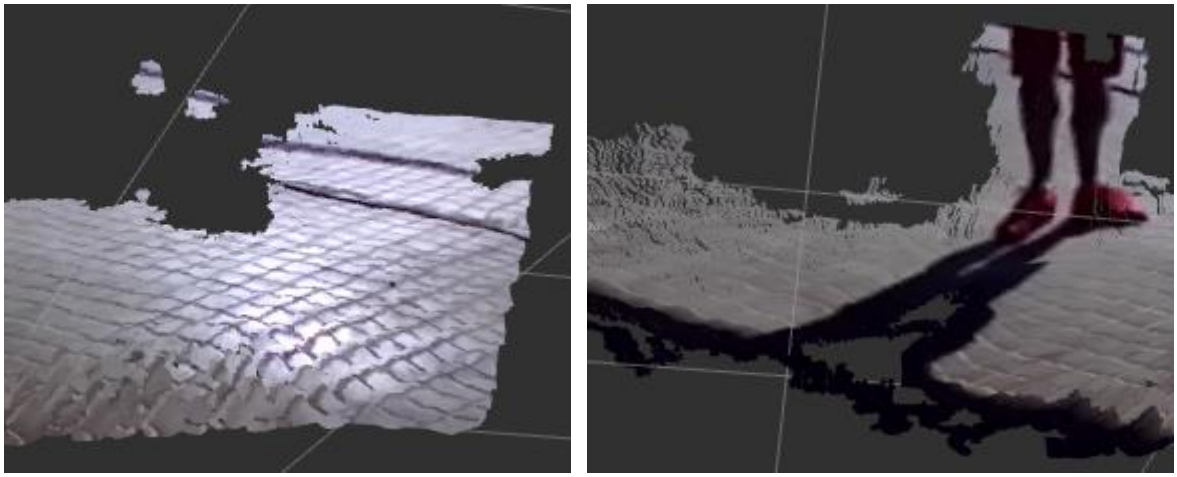


Figure 7-9 Zed stereo noise

8. Budget

In this chapter, the costs of the project are detailed. The time of realization of the project is approximately 5 hours every working day during 12 months. Taking account the hours dedicated and material used, the total cost of the project is 36,689 €, taxes included.

8.1. Engineering costs

Engineering costs cover the number of dedicated hours to this project including design, implementation and testing. The salary for an engineer have been estimated to 30 €/h.

Engineering costs	Hours	€/hour	Amount
Design	500	30	13,500 €
Implementation	300	30	9,000 €
Testing	450	30	12,000 €
Total			34,500 €

8.2. Equipment costs

Equipment costs cover a 5 year amortization of all the equipment used for the realization of this project. The equipment used is the computer equipment with all its accessories, the robot and the sensors. The amortization of this equipment has been calculated in relation to the duration of the project, being 12 months.

The robot item includes all its hardware and software. It includes the pioneer platform, batteries, computer, electronics and structure.

Computer equipment costs include a laptop and a desktop computer with Ubuntu Operating system. No software licenses were needed for the development, as only open software was used.

Equipment costs	Real cost (€)	Year Amortization (€)	Cost
Velodyne VLP-16	5.850	487.5	487.5 €
Realsense D-435	228	45.6	45.6 €
Zed Stereo	439	36.58	36.58 €
Ana robot	6,000	1,200	1,200 €
Computer equipment	2,100	420	420 €
Total			2,189 €

8.3. Total cost

Here the total cost is shown as the sum of the engineering and equipment costs.

Total cost	Amount
Engineering costs	34,500 €
Equipment costs	2,189 €
Total	36,689 €

9. Impact of the project

Here a brief description of the impact that can cause the project to the environment and to society.

9.1. Environmental impact

The realization of this project doesn't have much environmental impact, as it consisted in developing algorithms and testing them, both in simulation and in real experiments. All experiments were done in a safe environment, without any peril for nearby people, the robot itself or urban furniture. However, there are minor aspects that, regardless of not having a great impact by itself, can cause environmental damage if not actuated properly.

As it was explained in chapter 3, where it was described the used robot, the robot is powered by two lithium electrical batteries that could harm the environment if not handled properly. Those batteries have toxic components inside, and an escape of those is to be avoided. There is no need to say that on this project the batteries were handled responsibly and no dangerous situations were presented in relation to this.

The electrical consumption of the robot itself and the equipment used in the realization of this project is negligible as is not much more than the consumption of a person on its day to day, and is good to remark that no harmful emissions for the environment were emitted excepting the indirect ones regarding to the used electricity production and fabrication and transportation of the equipment.

Another thing to take into account is the necessity of proper handling of the other equipment used, taking it to proper recycling establishments when its lifetime comes to an end. It is important also if able to reuse the components of the equipment, as the fabrication of new ones involves a cost not only economical, but also to the environment.

9.2. Social Impact

Regarding of the social impact of the project, if we look to the bigger picture and autonomous navigation for goods delivering becomes a thing, it will affect urban spaces and even the conception of cities as we know it nowadays. It will become normal to have robots surrounding us in our lives, so people we will have to be used to its presence in the streets and learn to interact with them.

It is hard to imagine how cities and urban environment will change in the future, but it is a fact that robots will be an important cause for all of these changes. However, this won't happen until the technology is mature enough for ensuring the safety of people first and then safety of the robots and environment.

Another point of view is looking to the economical aspect, as a costs reduction for transportation of goods at the last mile could propitiate new paradigms, even some of them that we haven't thought of.

In relation to jobs, automatization of transportation could sadly imply the jobs destruction on the particular tasks of drivers and dealers, but is something that affects a lot of other sectors too. This is not very alarming though, as even the fact that some jobs can become useless is being countered by the new possibilities and jobs that will be created. People will have to get used to be continuously recycling themselves and be prepared for new opportunities.

10. Conclusions and future work

In this master thesis, we presented an algorithm for obstacle detection in urban environments during autonomous navigation of a robot, using a LiDAR sensor and a depth camera. This detection is done by comparing height values of the point clouds obtained with the sensors.

Also, this algorithm was integrated in a ROS navigation module for performing the autonomous navigation with the Ana robot.

Results of simulating the obstacle detection and navigation were satisfactory, as the robot could reach a goal without falling into dangerous situations.

Finally, real experiments showed the correct detection of obstacles with the LiDAR, but although the same obstacles were detected with the camera too, it appeared false positive detections due to the camera noise in sunny days.

This makes the tested cameras unfeasible for outdoors navigation with presence of the mentioned reflections. Until the date of writing of the report, no feasible solution was achieved to the problem, but it is being working on it. Some of other ways of development are the use different cameras, like a ToF camera, and also doing the navigation without a camera, inclining downwards the LiDAR sensor to reduce its dead zone or adding another one.

Acknowledgments

I would like to specially thank my tutor Alberto, who gave me the opportunity to take part in one of his projects, the lab technical support guys Fernando and Sergi, who always helped me when I needed it and Jose, who helped me with taking captures of the experiments.

Bibliografy

[1] GHANI,M. and SAHARI, K. *Detecting obstacle using Kinect sensor.* International Journal of Advanced Robotic Systems, 2017 1-10.

[2] NETAJI, M and ARGALL, B. *Automated Incline Detection for Assistive Powered Wheelchairs IEEE RO-MAN, 2017*.

[3] MOBILE ROBOTICS, ACTIV MEDIA ROBOTICS*. Pioneer 3 operations manual* [https://www.inf.ufrgs.br/~prestes/Courses/Robotics/manual_pioneer.pdf, (accessed September 7 2019)].

[4] OKUNSKY, M V and NESTEROVA, N V. *IOP Conference Series: Materials Science and Engineering*, 2019 ,Volume 516, n 1

[5] VELODYNE LIDAR. *PUCK Data Sheet*. [https://velodynelidar.com/vlp-16.html, (accessed September 6 2019)].

[6] *HUIJIE Z. et al. A High Throughput Integrated Hyperspectral Imaging and 3D Measurement System.* School of Instrumentation Science and Opto-electronics Engineering, Beihang University, Beijing 100191, China.

[7] INTEL REALSENSE. *Depth Camera D400 Series Data Sheet.* [https://www.mouser.com/pdfdocs/Intel_D400_Series_Datasheet.pdf, (accessed September 8 2019)]

[8] STEREO LABS. *Zed Stereo Data Sheet. [https://www.stereolabs.com/zed/docs/ZED_Datasheet_2016.pdf*, (accessed September 14 2019)]

[9] ASUS. *Asus Xtion Pro Live Data Sheet. [https://www.asus.com/es/3D-Sensor/Xtion_PRO_LIVE/specifications/* ,(accessed September 14 2019)]

[10] ROBOT OPERATING SYSTEM. *ROS documentation.* [http://www.ros.org/, (accessed September 7 2019)].

[11] *ROS. AMCL documentation* [http://wiki.ros.org/amcl?distro=melodic, (accessed September 13 2019)

[12] POINT CLOUD LIBRARY *PCL documentation.* [http://docs.poinclouds.org/trunk/modules.html, (accessed September 9 2019)]

[13] ZHENG, K. *Navigation tuning guide. 2016*

[14] STROUSTRUP, B. *The C++ Programming Language*. Addison Wesley, 2013.