POLITECNICO DI TORINO

Department of Control and Computer Engineering
Master of Science in Mechatronic Engineering

Politecnico di Torino

Master's Degree Thesis

VISUAL ODOMETRY (VO) TECHNIQUE FOR CHALLENGING ENVIRONMENTS WITH FOCUS ON LOW-TEXTURE

Supervisor:

Prof. Marcello Chiaberge

Candidate:

Chiara Bonanno

Academic Year 2021/2022

Ai miei genitori e

a mio fratello Alessandro.

Dove siamo nell'universo? Chi dirà la posizione,
dirà tutto.

[FABRIZIO CARAMAGNA]

Abstract

Master thesis project, entitled “Visual Odometry Technique for challenging environment with focus on low-texture”, has been carried out at PIC4SeR (Politecnico di Torino Interdepartmental Centre for Service Robotics).

Service robotics is based on creating autonomously or semi-autonomously systems useful for the human wellbeing. It can have several applications ranging from precision agriculture to space and to services for people with disabilities. The latter is the field of this research project: visual sensor is mounted on a wheelchair that is moved in indoor environment. Different cameras and Visual Odometry algorithms are evaluated in order to obtain a precise real-time position variation, focusing on low-texture challenge. Generally, indoor environments, such as hospital, do not have many features, so placing a front-facing camera may not solve low-texture problems. The idea of placing the visual sensor facing downwards as well as frontally is proposed, in order to focus exclusively on the floor texture and obtain the desired Odometry with higher accuracy.

The purpose of this research is to localize a robot within limited indoor environment through Visual Odometry.

The algorithm discussed in this Master thesis project consists of two parts. In the first one, consecutive images captured by the camera are considered. Thus, landmarks in the surrounding environment are examined and feature detection and matching algorithms are applied, evaluating the change in position of that feature between one frame and the following. In the second part, the Essential Matrix and the Fundamental Matrix are calculated, obtaining an estimate of the robot's position point by point. At the end of the algorithm, the trajectory of the wheelchair is obtained. After having created a simulation environment on Gazebo and once transcribed the above-mentioned algorithms in Python language, ROS 2 Foxy is used to simulate the behaviour of the system. The correctness of the robot's trajectory is verified by comparing the Odometry obtained from Gazebo and the Odometry obtained through the algorithm, by means of a Jackal 3D model. Afterwards, once evaluated the efficiency of the algorithms, system is simulated in real world using a Turtlebot robot.

Through this experimental approach, it can be demonstrated that a low-texture environment compromises the accuracy of the trajectory; however, with the camera facing downwards, when features are greater, the odometry is more accurate.

Keywords

Visual Odometry, Intelligent Wheelchair, Service Robotics, Localization, Texture, Indoor Environment, Visual Sensors, Trajectory, Pose Estimation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, according to Istat data, the average age has increased significantly. The high number of elderly people, combined with the increasing presence of disability cases, such as multiple sclerosis or amyotrophic lateral sclerosis, has led to the need for constant assistance.

For this reason, there is an increasing focus on creating autonomous wheelchairs based on the model of self-driving cars in the automotive field, with the aim of making disabled people more independent.

The idea is to create a system that can move autonomously within spaces such as hospitals, apartments, airports, etc. providing a specific destination, with the help of sensors. For this purpose, there are already several techniques that may involve prior knowledge of the surrounding environment or techniques that do not involve knowledge of the environment.

The idea for this work was born after meeting the start-up Alba Robot [1] an innovative company that aims to use the most advanced technologies to create a platform for electric vehicles equipped with autonomous driving systems.

The purpose of this thesis is to localize a system, the wheelchair in question, in indoor environments through visual odometry techniques using visual sensors.

For proper localization, sensors need features in the space around them. There could be problems in hospitals or airports where walls are generally white and there are not enough landmarks around. Hence, the choice is to consider placing the sensor facing downwards. Analysing floor features is a good approach to overcoming the aforementioned challenge.

The first phase of the work has been to create a simulation environment on Gazebo in ROS 2 that could represent a hospital environment, equipped with rooms, corridor and different obstacles. Then, a floor equipped with tiles with different features has been reproduced with the use of some blocks. The system chosen for the simulations has been Jackal Robot characterized by four wheels, a chassis and a visual sensor Intel RealSense d435i. The idea has been to compare the odometry of the robot with the camera facing the front and those with the camera facing downwards.

In the second phase, an algorithm has been developed in Python language in order to evaluate the Odometry of the robot tracing its trajectory. It consists of two main parts, in the first one the video is captured by the camera mounted on the robot and the features of two consecutive frames are detected and matched; in the second one the position of the camera is found in real time through the calculation of the Fundamental matrix and the Essential matrix.

At the end of this approach, the trajectory of the robot is obtained by summing its various positions. After verifying the correctness of the algorithm and after plotting the trajectory of the robot, we have moved on to

the practical phase by testing it in the real world, precisely in the corridors of PIC4SeR, using Turtlebot3 Burger robot, evaluating whether the behaviour of the algorithm varied with the switch from Gazebo to the real world.

1.1 Thesis Structure

The dissertation begins by introducing the need to create an autonomous wheelchair and the purpose of the work with the challenge to be overcome. The first chapter briefly discusses how to approach the project with the Visual Odometry algorithm developed. Then, simulation environment is described indicating the robots used on Gazebo in ROS 2 and in the real world (Chapter 1).

The Second chapter (Chapter 2) discusses the wheelchair, with its brief history and description of the environments in which it moves in this work.

The Third chapter (Chapter 3) focuses on Visual Odometry (VO), its state of the art and the various problems related to low texture. A solution is proposed to overcome the challenge followed by a detailed description of the algorithm developed

The following chapter (Chapter 4) deals with visual sensors giving an overview of the types of sensors with their various characteristics and focusing on their best performing environments.

The next chapter (Chapter 5) deals with the simulation environment with the description of the robot used on Gazebo and in the real world. In this part is also described the hospital environment and the floor with its features on which the algorithm of Visual Odometry will be tested.

The sixth chapter (Chapter 6) shows the experiments on ROS 2 and in the real world with the aim of evaluating the performance of the algorithm.

Finally, in the last chapter (Chapter 7), results obtained during the experiments are analysed with the conclusion of this work and with potential future work related to Visual Odometry in indoor environments.

Chapter 2

Intelligent Wheelchair

More and more people suffer from temporal or permanent disabilities due to illness, accidents, and motor problems related to old age. In most cases, these conditions lead to the need for constant support from other people. The inability to move independently can alter psychological well-being. In fact, this condition can limit social relationships, creating conditions of depression and anxiety.

For cases of difficult or impossible walking the use of a wheelchair is becoming essential. In recent years, research has been focusing on making smart wheelchairs capable of satisfying patients with various levels of disability, Intelligent Wheelchairs. Their purpose is to meet the different types of situations and disabilities, making the user independent and safe [2].

Intelligent Wheelchairs are in fact equipped with advanced functionalities that allow the user to move independently. It consists of Powered Wheelchair with advanced capabilities such as voice control, fingertip control, head movements control or different inputs that do not involve use of commands [3].

According to [4], System navigation in Service Robotics is characterized by four main blocks: Perception, Localization, Cognition, Motion. The first block consists of understanding the surroundings, evaluating possible static and dynamic obstacles and different environment features; Localization represents the continuous awareness of the robot's position within the environment in which it moves; Cognition block allows the robot to evaluate possible ways to achieve various goals, such as how to reach a specific destination; finally, the last block is the motion control that allows the robot to manage its movement in acceptable way.

Most Intelligent Wheelchairs move within an environment whose map is provided, others are unaware of their surroundings and update a map keeping track of their location in real time. The latter is Simultaneous Localization And Mapping (SLAM) and it is a key problem in the field of Artificial Intelligence because a priori map is not accessible and the system needs to locate itself [5]. To overcome this challenge, researchers are continuously conducting studies evaluating various criteria for Robot localization in both outdoor and indoor environments.

The main element of Autonomous Wheelchair are sensors. They are crucial for avoiding obstacles especially in dynamic environments guaranteeing greater user safety [6]. Sensors can be of different types according to the environment in which they are to operate and according to the performance they are to execute. In service robotics, the most used sensors are visual sensors, navigation ones or those capable of measuring the velocity, orientation, and gravitational forces [7].

All these elements, combined with navigation algorithms, allow wheelchairs to be intelligent by meeting the needs of patients with disabilities. In this way, the latter will no longer require assistance, improving their psychological condition; they will also feel more independent in terms of social contacts.

For example, a Turin-based start-up named ALBA Robot (Advanced Light Body Assistant) has focused on passengers with reduced mobility, equipping hospitals, airports and museums with services based on autonomous driving. It is a smart wheelchair that is guided with voice commands and allows disabled users to leave assistance from airport staff or museum staff [1].

2.1 Related Work

Several smart wheelchair models have been introduced in recent years. Below, according to the [8], are the most significant smart wheelchairs that have contributed to improving the living conditions of people with disabilities.

One of the very first smart wheelchairs dates to the last decade of the 1990s. Its name was *Mr. Ed* and it was implemented by Connell and Viola, it could be controlled through a joystick and was equipped with infrared proximity and sonar sensors that allowed the system to avoid obstacles or follow moving objects.

In 1991, Simon Levin at the University of Michigan developed *NavChair.* It had control system that allowed it to travel safety avoiding obstacles and following walls, adapting to every situation and possible changes in the surrounding environment. All this was possible thanks to twelve ultrasonic sensors and a computer (*Figure 2.1*). Users could set the route to be taken, indicating directions and speed.

Figure 2.1 NavChair Wheelchair [8]

A few years later, Miller and Slack at the KISS Institute in Virginia developed *TinMan I* and *II*. The first prototype could move using a joystick; the second one (*Figure 2.2*), a controller and sensors were added. They allowed the wheelchair to avoid obstacles and follow walls.

Figure 2.2 TinMan II Wheelchair [8]

At the same time, CALL Centre at the University of Edinburgh was developing the CALL Centre smart wheelchair to meet children's disabilities. It was computer-controlled and equipped with switch, joystick, and other elements capable of driving it.

Instead, at Nagasaki University, the first wheelchair capable of locating itself within indoor environment by exploiting landmarks was developed (*Figure 2.3*). For this purpose, it exploited ceiling lights captured through visual sensors. If the lights were missing, azimuth sensors, placed on the wheels, allowed the wheelchair to continue navigating safely. With the help of these sensors, the captured image was processed by a circuit with FPGA that informed the user of the presence or absence of obstacles.

Figure 2.3 Nagasaki University Wheelchair [8]

Obstacle avoidance, passage in narrow spaces, and navigation based on landmarks are the features of the *TAO 1* (*Figure 2.4*) and *TAO 2* (*Figure 2.5*) wheelchairs developed around 1995. With the help of a microcontroller that managed visual, bump and infrared sensors, it was possible to move independently and safely.

Figure 2.4 TAO 1 Wheelchair [8]

Figure 2.5 TAO 2 Wheelchair [8]

In the first decade of the 2000s, the first Lidar SLAM-based wheelchair model with 3D map reconstruction was developed [9]. The goal in this wheelchair was to create a safe, intelligent, and low-cost system. The user could move with either a joystick or GUI (Graphical User Interface) interfaces via computer. Lidar sensors were the stars of this model, there was a horizontal one and a vertical one, and then there was also a stereo camera (*Figure 2.6*).

Figure 2.6 Intelligent Wheelchair platform [9]

2.2 Indoor Environment

Intelligent Wheelchairs are designed to be used in a variety of settings, as long as they assist people with disabilities, but there are some differences involving indoor or outdoor scenario.

The main difference is size. In fact, wheelchairs designed for indoor environments should not be bulky in order to be able to manoeuvre them comfortably. Speed is also a parameter that depends on the space in which it moves; in indoor environments where there are other people, or obstacles and narrow spaces, the wheelchair does not need to reach high speeds [10].

If we consider possible interior spaces within which a wheelchair can move, house and hospitals come to mind. Narrow spaces within rooms, bathrooms, or elevators prevail in these types of environments and for a wheelchair to move comfortably, in addition to its size, it is necessary to use sensors that detect obstacles even very close to it.

Another problem that frequently happens is the low texture of the indoor environment. This means the presence of few features detectable by sensors, due for example to white walls and redundant spaces.

One of the positive aspects of dealing with indoor wheelchairs is related to the wide choice of sensors (covered in Chapter 4); in fact, in these cases there is no problem of possible weather that may interfere, or high distances to be detected.

Chapter 3

Visual Odometry

The term "odometry" originated from the two Greek words *hodos* (meaning "journey" or "travel") and *metron* (meaning "measure") [11]. Visual Odometry (VO) consists of estimating the position of a system by evaluating the variation in motion from images captured by visual sensors. The trajectory is obtained from the sequence of images or the video stream from one or more cameras mounted on moving robot [12].

Figure 3.1 VO Input - Corridor [13]

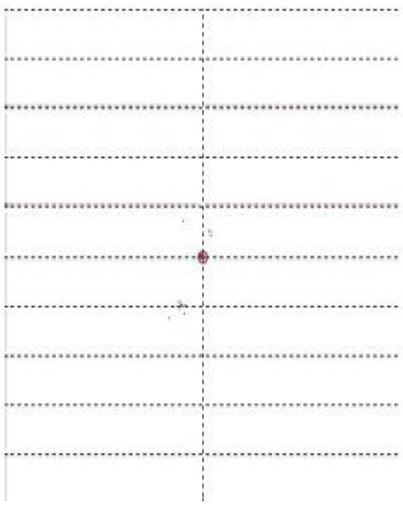

Figure 3.2 VO Output - Camera Trajectory [14]

Visual Odometry is a special case of SFM (Structure From Motion). SFM refers to the problem of reconstructing the three-dimensional structure and position of the camera from a set of images received as input [12]. As early as the 1980s, Moravec was studying on planetary rovers to overcome the challenge related to estimating the ego motion of the system from a single visual input coming from a slider stereo camera [15].

Until about 2000, Visual Odometry was used only for space research purposes by NASA. From 2004 with Nister [16], visual odometry began to be used for other purposes as well, moving into academic environment. He proposed a VO framework for terrestrial application using monocular and stereo camera sensor in order to obtain navigated path [17].

Until the emergence of VO, there was a similar technique considered the simplest for position estimation, called Wheel Odometry (WO). It works from data from the encoders coupled to the rear wheels. Each individual encoder generates 100 sets of pulses per revolution of the wheel. The *Figure 3.3* shows the wheel with the encoder (in green) which sends output signals to processor every 5 milliseconds [18].

Figure 3.3 Wheel Odometry with Encoder [11]

Unfortunately, it suffered from position drift due to wheel slippage mainly due to abrupt accelerations or uneven grounds [11]. With the introduction of VO, it was discovered that it was not affected by wheel slip and that relative positioning errors of VO was much lower than WO errors [19]. VO provides more accurate trajectory estimates, with relative position error ranging from 0.1 to 2% [12].

According to [20], Visual Odometry has various fields of application, in fact it can be applied in gaming, virtual reality, manufacturing industries, aviation and underwater environment, automobiles, agriculture, etc… For this reason, there are several techniques and classifications published in literature in recent years.

In 1988, Aggarwal and Nandhakumar were the first to classy image sequences captured by visual sensors into feature-based and optical flow based [21]. The first approach is to find features, keeping track of them as you move from frame to frame, but this method will be explored in more detail in Section 3.2.1.; Optical flow based approach is based on points within the image from which a speed is calculated by predicting the position of the next point.

Later, Sedouza and Kak classified the two techniques just mentioned into map-based, map-building-based and map-less navigation schemes [22]. In the latter, the map is not provided and therefore the movement of the system takes place considering the surrounding objects. Alternatively, the map of the environment may already be known to the robot from the beginning, or it may be created during its movement.

At the same time, other systems for position estimation already existed. One example is GPS (Global Position System), the first navigation method born in 1970 and mainly used to detect the position of systems in outdoor environments. GPS technique consists of constellation of 24 satellites orbiting the Earth that transmit radio frequency signals accurately calculating the position at any point on the Earth's surface. To increase accuracy, an INS (Inertial Navigation System) equipped with motion and acceleration sensors was often used in addition to the GPS [11]. But these methods just mentioned had their weaknesses, e.g. loss of information or poor accuracy, which is why a more robust and precise method is chosen when locating a system within an environment: Visual Odometry.

After capturing the different frames from the camera, they are analysed two by two taken consecutively by evaluating the features from feature detection technique (*Figure 3.4*).

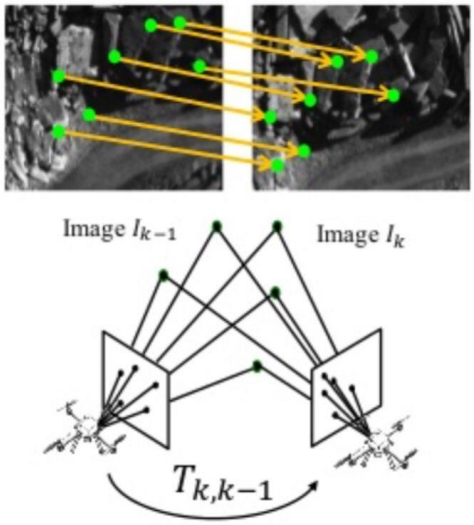

Figure 3.4 Frames at two consecutive time instants [23]

Depending on whether the features correspondences f_k and f_{k-1} (*k* and *k-1* are time instants) are 2D or 3D, there are three different cases [11]:

- *2D – to – 2D*: both f_k and f_{k-1} are expressed in 2D image coordinates.
- *3D – to – 3D*: both f_k and f_{k-1} are specified in 3D image coordinates. In this correspondence, triangulation of points in 3D is required at each instant of time.
- *3D – to – 2D*: f_{k-1} is expressed in 3D and f_k is expressed in 2D.

The benefit of using directly raw 2D points instead of triangulated 3D points lays in a more accurate motion computation [24].

For each method indicated above, there are three different motion estimation algorithms, which are briefly shown in the *Table 3.1* below according to [11].

2D – to – 2D	***3D – to – 3D***	***3D – to – 2D***
Capture new frame I_k	Capture two image pairs $I_{l,k-1}$ and $I_{r,k-1}$ and $I_{l,k}$ and $I_{r,k}$	Capture new frame I_{k-2} and I_{k-1}
Extract and match features of I_{k-1} and I_k	Extract and match features of $I_{l,k-1}$ and $I_{l,k}$	Extract and match features Triangulate matched features
Compute Essential Matrix E_k	Triangulate matched features	*Capture new frame I_k
Compute Transformation Matrix T_k	Compute Transformation Matrix T_k from 3D features	Extract and match features of I_{k-1} and I_k Compute camera pose PnP
Compute $C_k = C_{k-1} T_k$	Compute $C_k = C_{k-1} T_k$	Triangulate all new feature matches
Repeat	Repeat	Repeat from *

Table 3.1 Motion Estimation Algorithms

3.1 Low Texture Challenge

Most very large interior spaces, such as hospitals or airports, have a redundant layout. The different compartments are in fact repeated in the same way within the structure. Even the walls are often monochrome, lacking in features. It can be a problem when dealing with visual sensors and even more so when a self-driving robot is involved, which could lead to dangerous scenarios.

In recent years, several researchers have addressed this issue by trying to overcome the low-texture challenge. According to [25] [26], this condition occurs when there is no change between two consecutive frames and tracking can lost. Localization algorithms are not robust in low-texture environments, they do not perform well especially concerning SLAM algorithms (Simultaneous Localization And Mapping) that should create a map during robot movement [27]. It can produce a large drift error both in position and orientation or even a fail to work, by increasing the runtime efficiency [28].

3.1.1 Solution

In order to overcome low-texture challenge, over the past few years, different solutions have been proposed. The most common ones concern the choice of sensors and feature detection algorithms. Some of them consist of taking as features the lines identified within the structure in which the robot moves (i.e. columns, doorposts, skirting boards, ceiling lines, etc.) [28] or variations in color and objects in the surrounding. They are known as texture-based algorithms and, according to [29], they can be classified into:

- SIFT (Scale Invariant Feature Transform) Algorithm: it calculates a set of feature vectors for a greyscale image. It consists of identifying key points and then, for each of them, vectors describing the local region of the key point, called descriptors, are calculated. Given two images, if at least 3 key points coincide with a Nearest-Neighbour approach, then there is a high probability that the object described by the key points is the same [30].
- SURF (Speed Up Robust Features) Algorithm: it calculates the feature vectors of a greyscale image faster than the SIFT method. The procedure for obtaining descriptors is the same of the previous algorithm [30].
- FAST (Features from Accelerated Segment Test) Algorithm: it deals with searching for angles within greyscale images. The search is carried out by nominating a *pivot* point as a corner and evaluating a 16-pixel surround. The values of the surrounding pixels are checked and if there are at least 12 consecutive pixels that have a value that differs by a threshold t from the value of the pivot pixel, then a corner is detected [30].
- ORB (Oriented FAST and Rotated BRIEF) Algorithm: it builds on the FAST key point detector and the BRIEF descriptor. FAST applies Harris corner measure and uses pyramid to produce multiscale features. A negative aspect of FAST concerns orientation, which is not calculated by this algorithm, so ORB can steer BRIEF according to the orientation of key points [31].

Although these VO techniques show promising results for several applications, they are sensitive to environmental variations, such as lighting conditions and surrounding texture. Some of the other conditions that lead to poor tracking data are blurring, the presence of shadows, and visual similarity. In addition, some

artificial errors find their way into the data during the image acquisition and processing phases, such as distortion and lens calibration, feature matching or triangulation. Therefore, VO schemes must be robust and able to manage these problems efficiently [20]. However, it may also be useful to focus on the position of the sensor and consider modifying the shot.

In other words, turning the sensor downwards could solve few features problem. Floor tiles could hide relevant landmarks for the sensor. At the very heart of the principle of the computer mouse.

In particular, the use of mouse sensors offers robust solutions for applications requiring accurate relative positioning measurements. Firstly, they provide displacement measurements along ground surfaces with high frequency and accuracy. Secondly, their working distance is commonly limited to small measurements. In fact, their operating principle requires that a very small portion of the target surface be observed in order to report information over several frames. Generally, VO techniques allow the robot pose to be estimated in terms of six degrees of freedom. When sensors are oriented towards the ground plane, the problem of pose estimation can be simplified to a planar rotation and translation of three degrees of freedom (3-DoF) when considering ground robots [19].

Based on these considerations, the camera turns downwards, measuring how the accuracy changes.

3.2 Approaches

For each new image acquired by the sensor, feature detection and matching algorithms are applied between the first frame and the following one. Thus, image correspondences are considered: two-dimensional features that are the re-presentation of the same three-dimensional features in different frames [14].

The following step consists of calculating the relative motion between the instants of time between one image acquisition and the next one. Then, the camera pose is calculated by concatenating the relative motion with the previous pose.

These steps are shown in the diagram below (*Figure 3.5*):

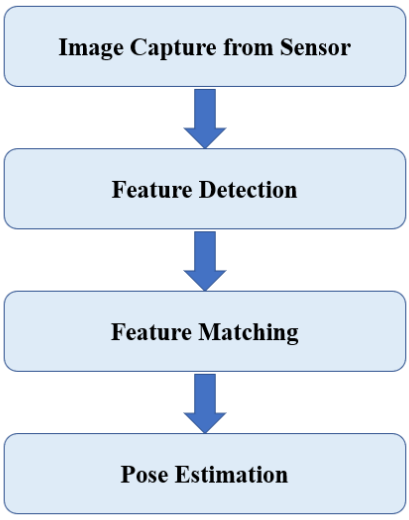

Figure 3.5 Block diagram VO Approach

3.2.1 Feature Detection Approach

According to [14], the relative motion can be obtained by two methods: appearance-based approaches and feature-based approaches. The first ones get information from the input image pixels; instead, the second ones use repeatable features from captured images. Most of the methods used for Visual Odometry are feature-based because it is considered a faster and more robust approach.

Image registration consists of detecting areas of local features, describing them, matching them, estimating the transformation pattern and aligning two images. Local features normally include points, lines, curves, edges and contours. However, common points between images, also known as key points, are used for image registration due to their simplicity in being detected and described. In addition, these detectors keep finding or identifying local regions, which have been subject to transformations in terms of illumination, scale, blurring, etc. [32].

The method chosen in the Visual Odometry algorithm is a feature-based method of the ORB (Oriented FAST and Rotated BRIEF) type. It is the combination of FAST detector and BRIEF descriptor. ORB uses the centroid of intensity to measure the orientation of angles [32]. ORB feature descriptors encode interesting information in a series of numbers and operate as a kind of numerical fingerprint. They can be categorized into two classes: local descriptor and global descriptors. The first one corresponds to a compact representation of the neighborhood, generally around an image point. The second one represents the whole image. Based on the key points identified in the image, feature vectors are created as in the image below [33].

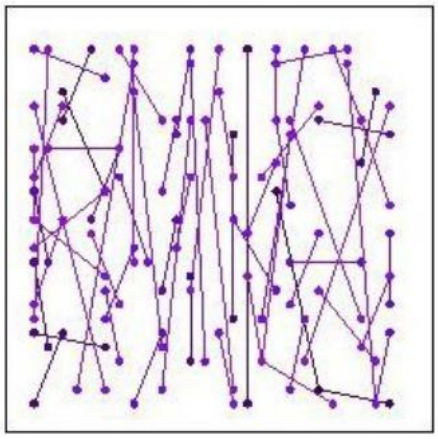

Figure 3.6 ORB Feature Detection Algorithm [33]

3.2.2 *Feature Matching Approach*

In computer vision, after finding descriptors and detecting interest points, they are matched. Feature matching consists in establishing feature correspondences between two consecutive images.

Matching features in this work is performed by Brute Force Matcher. The choice is not casual, even though the computation time is longer than in other methods, it is very precise. It takes the descriptor of a feature in the first image and compares it with the descriptors of all features in the second image using the distance calculation. Finally, the closest descriptor in a resulting pair is returned [34].

At the end of the matching algorithm, two consecutive frames with their matched descriptors are considered (*Figure 3.7*).

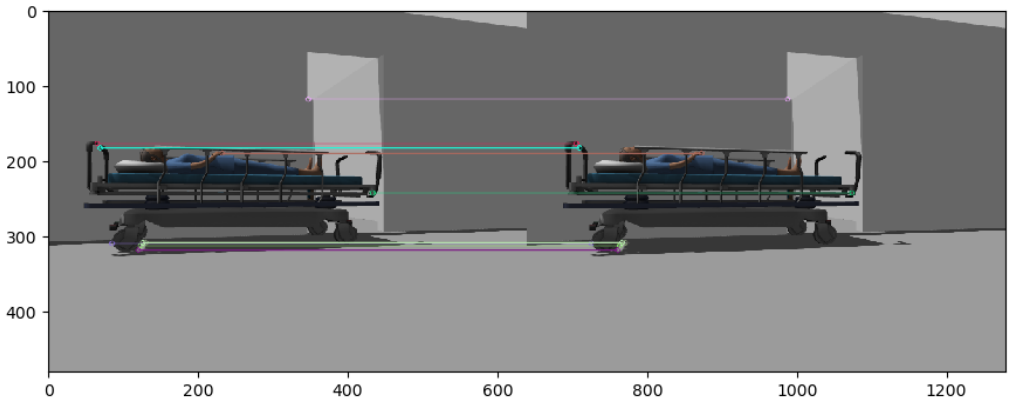

Figure 3.7 Feature Matching Algorithm result

The image above shows the result of the Brute Force Matching algorithm in Gazebo world. After setting the number of matches to be represented, the two consecutive frames representing a portion of the hospital environment are shown side by side.

3.2.3 *Pose Estimation*

In order to obtain the estimated camera position, there are a series of calculation steps to be performed. Both the Essential and Fundamental matrices completely describe the geometric relationship between corresponding points of pair of images coming from camera. There are several ways to derive these Matrices [35]:

1. the Fundamental matrix F is calculated using the eight-point algorithm with RANSAC (RANdom Sample Consensus). The accuracy of the F matrix is guaranteed by dividing the image into an 8x8 grid in which 8 random points are selected and once the Fundamental matrix is found, the error is

calculated, and it should be below a certain threshold value. Thus, only the F matrix with the lowest possible error is taken into account.

Given x and x' matches in two distinct frames and F Fundamental Matrix:

$$x'^T F\, x = 0$$

$$F = \begin{bmatrix} f11 & f12 & f13 \\ f21 & f22 & f23 \\ f31 & f32 & f33 \end{bmatrix}$$

Let $x = (u, v, 1)^T$ and $x' = (u', v', 1')^T$, each match gives the linear equation below:

$$uu'f11 + vu'f12 + u'f13 + uv'f21 + vv'f22 + v'f23 + u\,f31 + v\,f32 + f33 = 0$$

After selecting 8 points, Normalization of Fundamental Matrix is performed:

$$\tilde{x} = T\,x \qquad \tilde{x}' = T'x'$$

RANSAC (RANdom Sample Consensus) with 8 points consists in randomly sampling 8 points, computing F via least squares and enforcing $\det(\tilde{F}) = 0$ by SVD (Singular Value Decomposition). Then, we repeat the procedure and take into account only the F matrix with the most inliers.

F normalized is de-normalized, obtaining:

$$F = T'^T \tilde{F}\, T$$

2. Essential Matrix E is used to obtain camera poses between two consecutive frames. Before computing E matrix, Calibration Matrix K is calculated:

$$K = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 0 \end{bmatrix}$$

with fx, fy that corresponds to camera focal length, instead cx, cy are principal points coordinate of the skew parameter.

$$E = K'^T \text{ F K}$$

E Matrix is then decomposed again using SVD (Singular Value Decomposition) matrices.

$$E = U\ diag(1,1,0)\ V^T$$

Rotation Matrices R1 and R2 are obtained:

$$R1 = U\ W^T\ V^T \qquad R2 = U\ W\ V^T$$

where:

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

At this point, the correct position is verified using the Linear Triangulation method: the two feature points in the first and in the second frames are taken as reference and a new point in 3D space is determined by their projection.

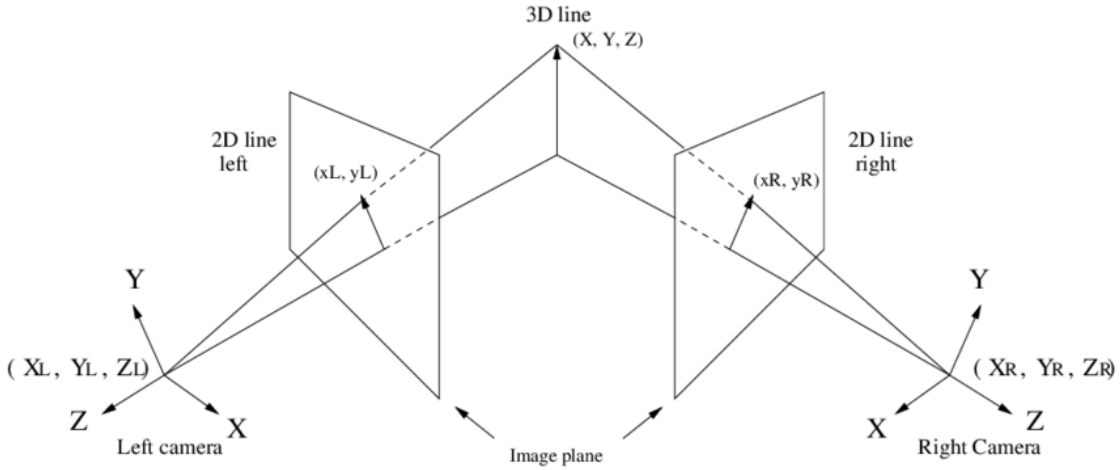

Figure 3.8 Linear Triangulation [36]

$$\bar{X}_L = R\ \bar{X}_R\ + T$$

Rotation R and Translation T Matrices are obtained from camera calibration in order to estimate the position of the system. Components that are calculated are *x* and *y* ones, *z* component is neglected because it is assumed that the distance with respect to the ground is kept constant during the displacement of the robot.

Chapter 4

Sensors

In recent years, service robotics is increasingly focusing on creating systems that can move autonomously. Sensors are used for this purpose; they can collect external stimuli and send them to the microprocessor recognizing the surrounding environment or detecting obstacles and variation in light intensity.

The robot's sensors can be mechanical, chemical, or electrical, and each of the sensor operations is based on the principle of transduction, which converts energy from one type to another.

The performance of a sensor, according to [37], is defined based on parameter such as resolution, repeatability, precision, and accuracy. The resolution is usually specified by the constructor in the technical specifications of the sensor, it conditions repeatability and accuracy; repeatability defines and quantifies the ability of a sensor to measure the same value with measurements made in different times; then, a measure is said to be precise if, made a series of measurements under repeatable conditions, they are concentrated around the average; and accuracy, on the other hand, expresses the absence of systematic errors in the measurement.

Many sensors are designed for outdoor or/and indoor environments, but it is very common to classify them according to the type of measured quantity. For example, there are position, speed, acceleration, distance, temperature, force, and visual sensors.

Using sensors means collecting a large amount of data that can be processed to optimise processes, resources, services. Using a sensor means increasing safety, monitoring environments and processes in real time and increasing production [38].

4.1 Visual Sensors

The term *visual sensor* refers to all sensors which, like the human eye, detect at least a two-dimensional image of the object to be measured [39].

Visual sensors are essential for finding visual odometry. Information from the surroundings is obtained from them and the most used visual sensors are cameras.

In [40], the types of cameras used for these purposes are classified:

- *Monocular Camera*: monocular camera-based visual odometry can have problems related to the perception of actual size, known as "Scale Ambiguity". Monocular cameras are used when size and weight are crucial; they are light in weight and compact [41] (*Figure 4.1*).

Figure 4.1 Monocular Camera [41]

- *Stereo Camera*: The stereo camera is equipped with two monocular cameras (two views) spaced by a known distance called the “Baseline” (*Figure 4.2*).

Figure 4.2 Stereo Camera [41]

- *RGB-D Camera*: it is also called a Depth camera because it is possible to obtain depth in pixels. This camera consists of emitting infrared beams useful for calculating distance based on TOF time-of-flight. Their limited range makes them more efficient in indoor environments [41] (*Figure 4.3*).

Figure 4.3 RGB-D Camera [41]

- *Event Camera:* it measures changes in brightness per pixel responding to changes. The event camera has high temporal resolution, low power consumption and does not suffer from motion blur. Therefore, event cameras are able to provide better performance than traditional cameras (*Figure 4.4*).

Figure 4.4 Event Camera

Visual sensors are cheap and passive, and a great deal of information can be obtained from them. They can, however, be difficult in poorly textured environments.

4.1.1 Intel RealSense D435i

Intel is a manufacturer of visual sensor, and the Intel RealSense D435i depth camera is a stereo solution used for multiple applications, which is part of the RGB-D visual sensors group.

The wide field of view is perfect in robotics or virtual and augmented reality, where it is essential to see as much of the scene as possible. This small camera can be easily integrated, having a range of 10 m. It is equipped with an Intel module, a vision processor and personalised software that allows it to interact with the environment. It is compact, lightweight, powerful and performs well in low-light scenarios [42].

Intel RealSense D435i specifications are enclosed in the *Table 4.1* below.

Parameter	**Value**
Resolution	1920 x 1080 px
Frame Rate	30 fps
Sensor FOV (H x V x D)	69.4° x 42.5° x 77° (± 3°)
Dimensions	90 x 25 x 25 mm
Connection	USB-C 3.1 Gen 1

Table 4.1 Intel RealSense D435i specifications

The high performance given by the above specifications make this sensor the most suitable for the purpose of this project (*Figure 4.5*).

Figure 4.5 Intel RealSense d435i

Inertial Measurement Unit (IMU) is also integrated in the sensor. It consists of a device that "reports acceleration, orientation, angular velocities and other gravitational forces. It consists of 3 accelerometers, 3 gyroscopes and 3 magnetometers" [43] (*Figure 4.6*).

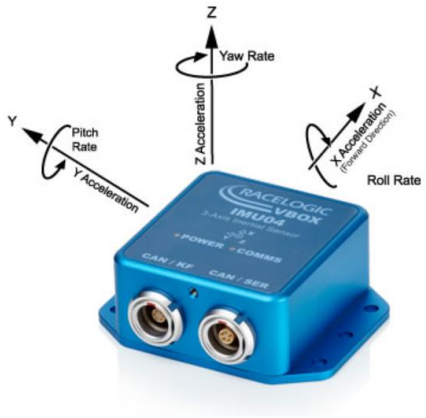

Figure 4.6 Inertial Measurement Unit (IMU) [44]

Chapter 5

Simulation Environment

The simulation environment used for this thesis aims to represent an indoor environment where a wheelchair could move. It is generally in hospitals that low texture problems might occur, which is why the simulated environment is a portion of a hospital. The idea is to reproduce a space with obstacles and featureless walls.

The operating system used is Linux Ubuntu, using ROS 2 Foxy [1] and Gazebo 11[2] simulator. ROS is a Robot Operating System used in service robotics that use several programming languages including Python.

ROS is a Robot Operating System used in service robotics that uses several programming languages including Python. It is equipped with libraries, datasets and various processes that are located within Packages. Through the use of libraries, Nodes are created to process data. Several nodes communicate with each other by exchanging messages via buses called Topics [45].

Gazebo, on the other hand, is a virtual simulator that allows robot models to be simulated within an environment (World) created through the tools provided by it. With the simulator it is possible find robot and sensor models to perform a system and its behaviour.

5.1 Simulation World

The world created on Gazebo aims to represent a hospital with rooms and a corridor within which a robot can move. In order to make the performance as realistic as possible, all elements that can be found in a hospital are included. From beds, to stretchers, wardrobes, chairs, nurses, people, cupboards. These are the elements found in Gazebo's libraries. Finally, walls, doors and other furnishings were added to add value to the simulated world. After creating the simulation environment, it is saved as a *.world* file.

In the test phase the robot will travel within the hospital environment, passing through corridors and entering rooms, evaluating its behaviour in a space similar to the real one (this part will be addressed in the next chapter).

Figure 5.1 shows the hospital environment on Gazebo before the simulations are carried out.

[1] ROS 2 Foxy: https://docs.ros.org/en/foxy/Installation.html
[2] Gazebo 11: https://classic.gazebosim.org/download

Figure 5.1 Simulation environment on Gazebo

5.2 System Modelling

Regarding the system used for simulations, Jackal Robot is considered. Jackal is a platform equipped with a chassis, four wheels, an on-board computer and an IMU, capable of simulating a robot [46].

Robot model is inside *Jackal_Simulation* package and it is represented by Unified Robot Description Format file (URDF), an XML format in which all robot specifications are indicated [47].

To understand the space occupied by the robot within an environment, it is necessary to be aware of its dimensions. According to [48], the Jackal design with its dimensions is shown (*Figure 5.2*).

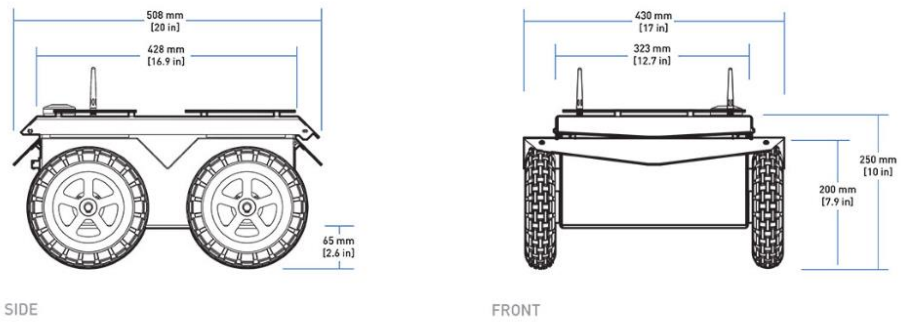

Figure 5.2 Jackal design with dimensions [48]

Jackal robot is equipped with a sensor Intel RealSense d435i, which can be used to take information from the surrounding space. The following photo shows the Jackal inside the World created on Gazebo with the sensor at the front (*Figure 5.3*).

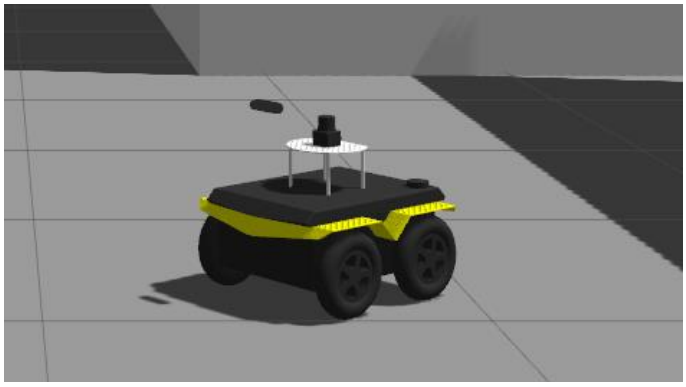

Figure 5.3 Jackal Robot with sensor

The aim of the research is to analyse odometry results by comparing the images captured by the front-facing camera and the images from the downward-facing camera. For this reason, the system will be modified during the simulation regarding the sensor mounted above.

In the second case, the system, in addition to having the camera facing the floor, is placed on blocks (*Figure 5.4*) that reproduce the characteristics of the floor and it is moved over them.

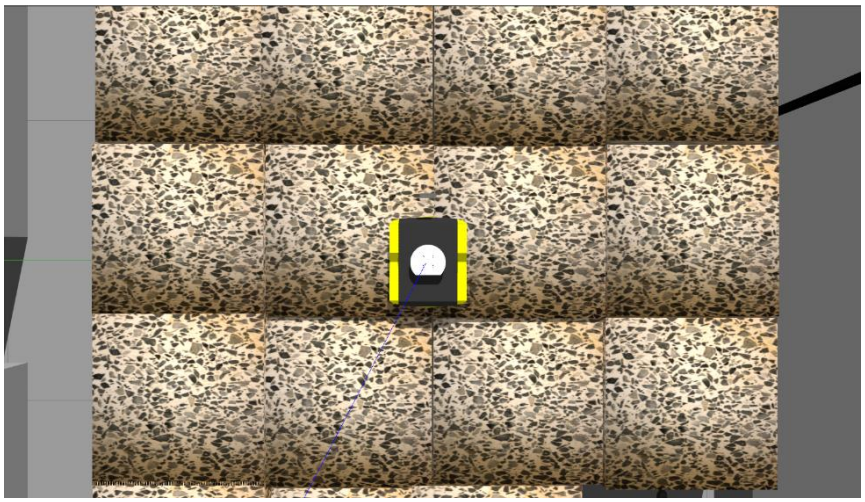

Figure 5.4 Jackal Robot above the blocks

Chapter 6

Experiments

This chapter focuses on the experimentation of the algorithm developed on Python in ROS 2. The aim of this phase is to evaluate first on Gazebo and then in the real world the behaviour of the Visual Odometry algorithm focusing on the low texture challenge. This is done by comparing the results obtained with the camera facing frontwards and then turning it downwards; this makes it possible to evaluate which is the best situation to overcome the challenge of low texture indoors.

Once the sensor position has been set, the system is moved within the environment and the developed algorithm is launched. Then, the odometry of the moving robot is evaluated using the computer keyboard and, finally, a trajectory is obtained. Through the latter it is possible to establish which sensor configuration provides better results.

In the following sections, we will first deal with the simulation on Gazebo and then the simulation of the algorithm in the real world, with the results obtained.

6.1 Experiments on Gazebo and ROS2

Before simulating the Jackal Robot within the hospital environment designed on Gazebo, for simplicity it is placedin the origin of the coordinate system (xyz), as in *Figure 6.1*.

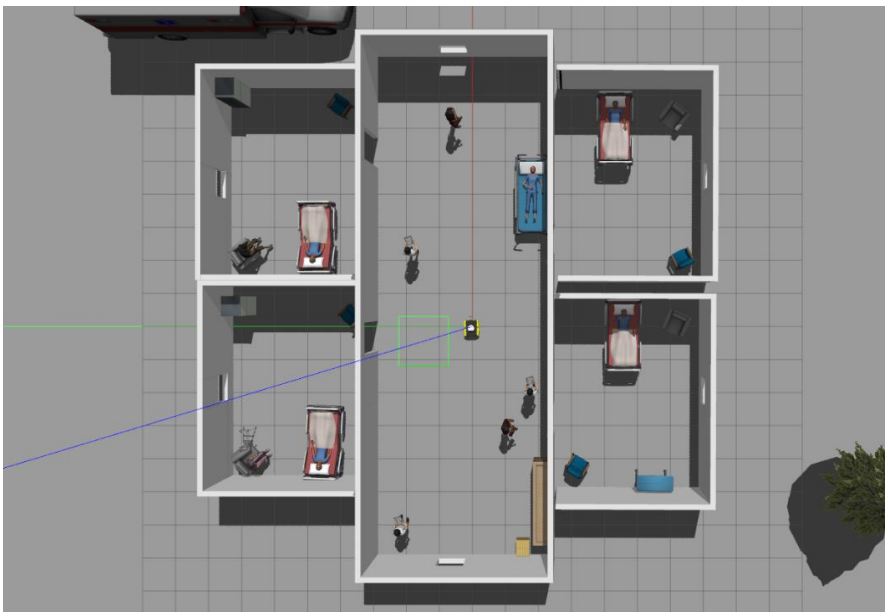

Figure 6.1 Jackal in the origin of the coordinate system on Gazebo

As described in *Chapter 3* where the approach used is explained, the algorithm begins with the capture of the various frames from the camera mounted on the Jackal, followed by the features detection and matching of two consecutive images.

This last step results in images like the one shown below (*Figure 6.2*). It is the result of the first step of the algorithm.

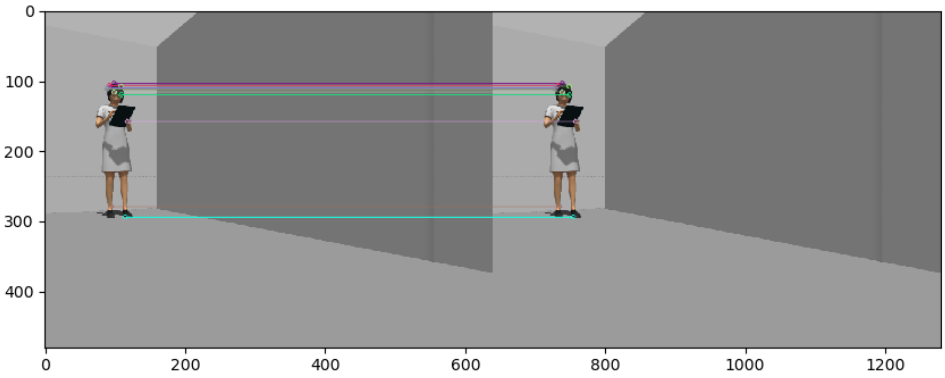

Figure 6.2 Features Detection and Matching output

The second step of the algorithm is to obtain the estimated position of the camera. The result of this step is a two-dimensional graph representing where the robot is located (as shown in *Figure 6.3*). From this step, it is also possible to obtain a text file with a list of the coordinates of the points in question.

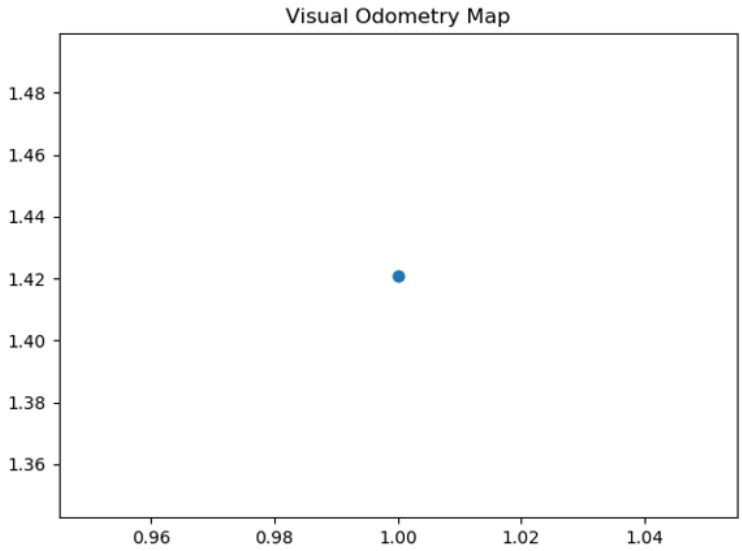

Figure 6.3 Jackal Pose Estimation (point)

Once the algorithm has been launched, moving the Jackal Robot in the Gazebo world allows you to obtain a series of coordinates of points that will be shown on a graph at the end of the path, thus obtaining a trajectory.

The first path of the Jackal Robot is shown in yellow in the image below. The system with the front-looking sensor is moved along part of the corridor and then into the hospital rooms of Gazebo (*Figure 6.4*).

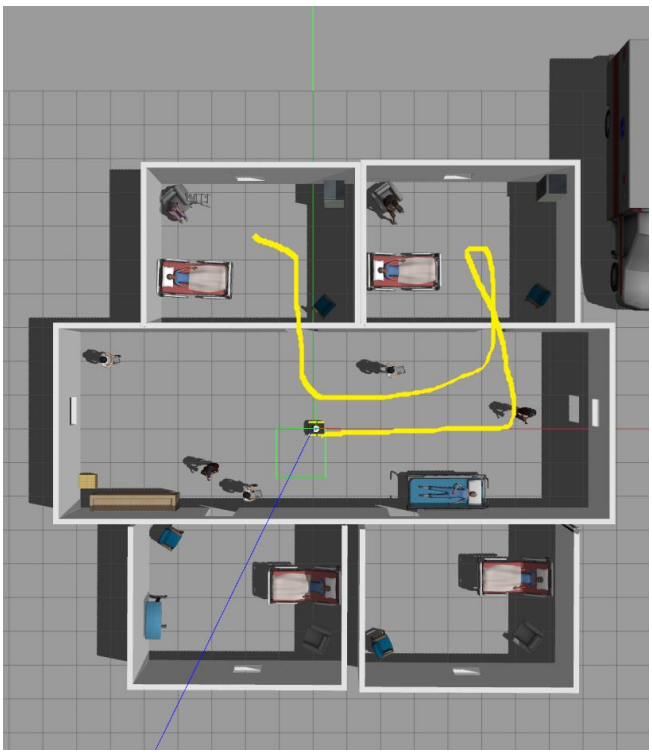

Figure 6.4 Jackal Robot first path on Gazebo

From this path, a series of points indicated in the table (*Table 6.1*) are obtained. The points obtained by the Algorithm (blue coloured – *Figure 6.5*) and those obtained from the */odom* topic of ROS 2 (red coloured – *Figure 6.6*), which accurately calculates the position of the robot in Gazebo, are taken into account and they are represented on a plot obtaining a linear trajectory.

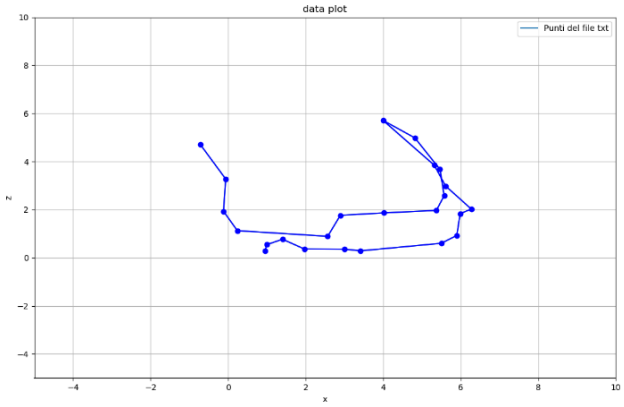

Figure 6.5 Linear Trajectory by the Algorithm

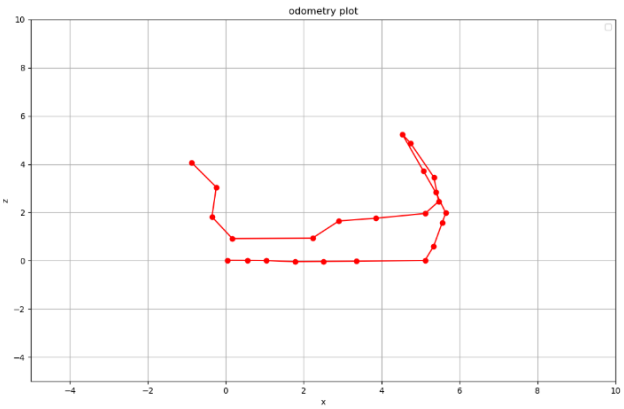

Figure 6.6 Linear Trajectory by the /odom topic on ROS2

FRONT ODOMETRY DATA FROM /ODOM TOPIC		FRONT ODOMETRY DATA FROM ALGORITHM	
x	**y**	**x**	**y**
0.03801664	0.01671115	0.95542770	0.29279739
1.03973587	0.00828904	0.99999999	0.54716256
2.50600921	-0.02942213	1.40211101	0.77049041
5.11435583	0.01017592	1.96713958	0.36511288
5.55456867	1.56689595	2.99935415	0.35875982
5.38821801	2.83377264	3.41711658	0.28860963
4.53422826	5.23658345	5.49661403	0.60208521
5.34227619	3.45906340	5.89906987	0.92802218
5.11705299	1.95598344	5.98431972	1.83152171
2.90579795	1.64752953	6.27134112	2.02628223
0.17009634	0.91461640	5.60822483	2.98649994
-0.24212538	3.04684091	5.31805137	3.84716610
0.55419263	0.01399195	3.99637281	5.71144656
1.78249209	-0.04054680	4.81743071	4.98296010
3.34943328	-0.01662266	5.46493490	3.67342199
5.33823024	0.60618695	5.57299987	2.58177152
5.64830591	1.98062956	5.37177152	1.96862162
5.07062691	3.72813670	4.02391132	1.86632565
4.74093228	4.87598356	2.89449020	1.76482826
5.46549208	2.45575941	2.56862162	0.89181554
3,85053497	1.75848588	0.23812424	1.12715393
2.22807532	0.93564048	-0.11903691	1.93178806
-0.35334629	1.82021107	-0.06285610	3.27515792
-0.87448221	4.06314262	-0.72181554	4.71771521

Table 6.1 Coordinates points from the algorithm and from /odom topic with front-facing camera

To better understand the difference between the two graphs obtained, they are represented in the same plot resulting in the figure below (*Figure 6.7*).

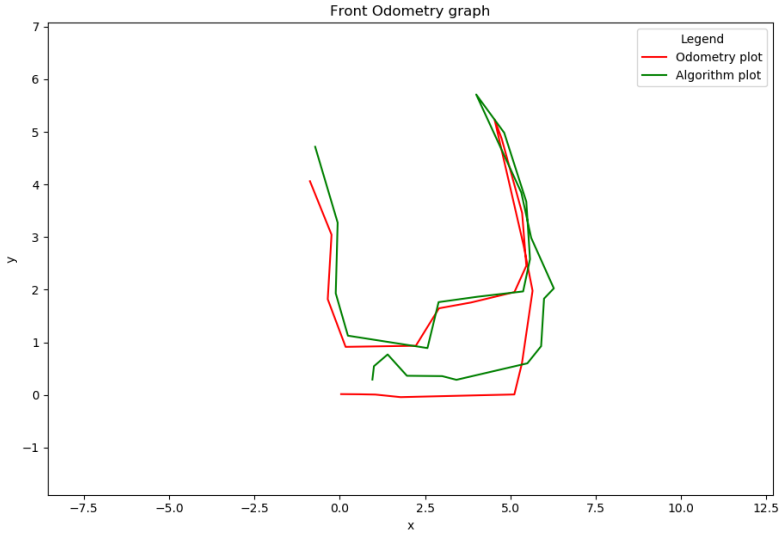

Figure 6.7 Linear Trajectories graph with front-facing camera

This procedure is, then, repeated by taking another route and placing the camera facing downward, specifically facing the blocks that reproduce the floor. The route on the Gazebo blocks is shown in the *Figure 6.8* in yellow.

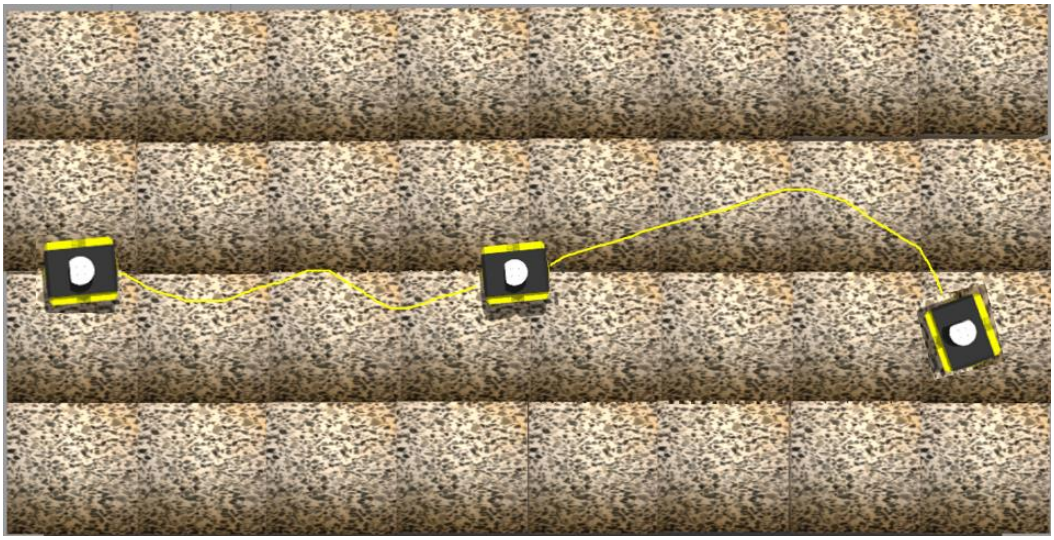

Figure 6.8 Jackal Robot path on Gazebo blocks

The downward-facing sensor captures frames by analysing the features of the tiles. In the first step of the developed algorithm, matching between the two consecutive images will be performed in the following way (*Figure 6.9*).

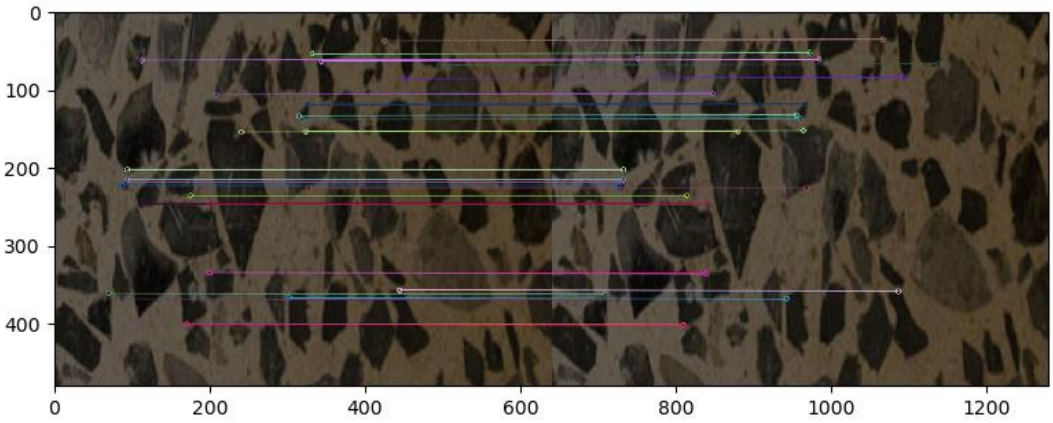

Figure 6.9 Features Detection and Matching output (Floor)

The points obtained by the Algorithm (blue coloured – *Figure 6.10*) and those obtained from the */odom* topic of ROS 2 (red coloured – *Figure 6.11*) with camera facing downward are represented in the following figures, obtaining a linear trajectory.

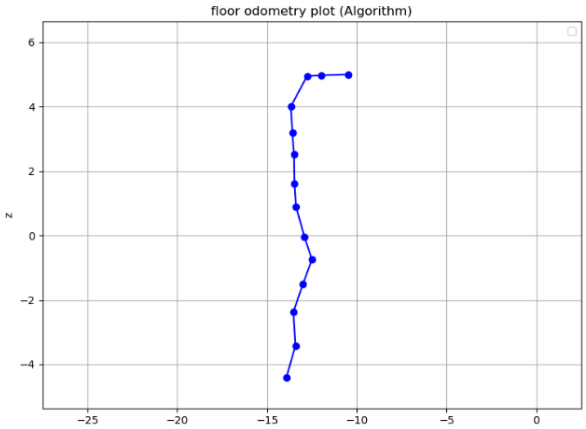

Figure 6.10 Linear Trajectory by the Algorithm (Floor)

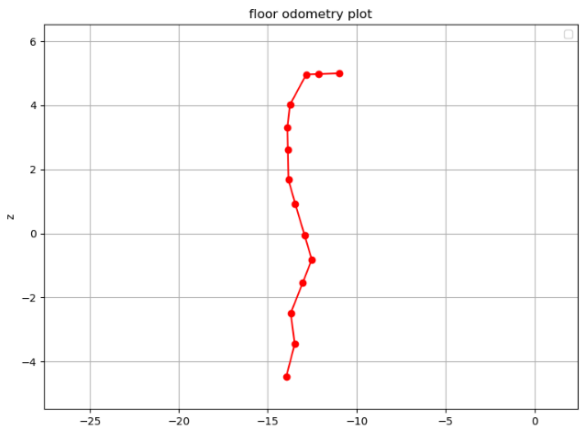

Figure 6.11 Linear Trajectory by the /odom topic (Floor)

Below is the data for the points obtained after running the Visual Odometry algorithm and after selecting */odom* topic in ROS 2 (*Table 6.2*).

FLOOR ODOMETRY DATA FROM ALGORITHM		FLOOR ODOMETRY DATA FROM /ODOM TOPIC	
x	**y**	**x**	**y**
-10.48925167	4.99827402	-10.97788150	5.00007822
-11.99364638	4.97218907	-12.15748783	4.97743919
-12.77245275	4.94762538	-12.83160075	4.96449315
-13.67482659	3.99836248	-13.75057520	4.01730617
-13.58972574	3.18374682	-13.89492489	3.29697784
-13.49826490	2.52375937	-13.86921101	2.61683855
-13.47264984	1.61664844	-13.83359165	1.67217949
-13.38497154	0.89965275	-13.45740472	0.93036312
-12.91243586	-0.04458258	-12.92141545	-0.07031138
-12.49743468	-0.74553886	-12.52467753	-0.81470175
-12.99873257	-1.50435861	-13.03830747	-1.54930358
-13.53237599	-2.37435769	-13.70777229	-2.48643931
-13.41746583	-3.41836484	-13.48402258	-3.45369840
-13.91746474	-4.39946146	-13.96126606	-4.47782195

Table 6.2 Coordinates points from the algorithm and from /odom topic with facing downward camera

As in the previous case, the two graphs are represented in a single plot in order to note the difference between the two trajectories (*Figure 6.12*).

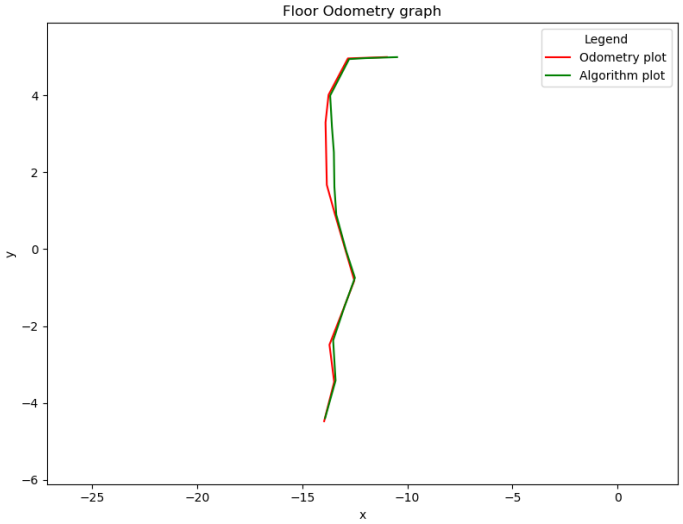

Figure 6.12 Linear Trajectory path with camera facing downward

Once it has been demonstrated that the linear path is consistent with that of the Jackal, the Jackal's trajectory is represented graphically by adding the rotation components.

The above procedure is repeated, taking into account the rotation obtained by the algorithm in the form of a matrix. Both sensor position conditions are considered again.

The robot is moved by making it curve within the Gazebo environment, with the aim of evaluating the curved trajectory.

The *Figure 6.13* below shows the path executed indicated in yellow.

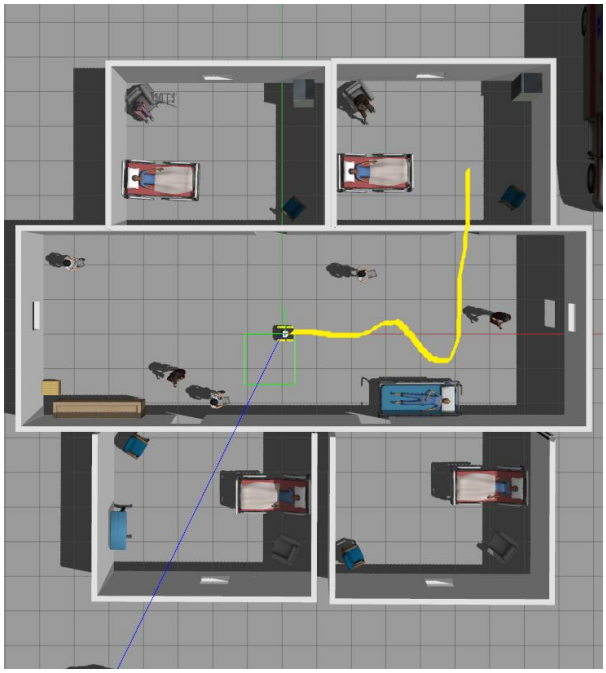

Figure 6.13 Jackal Robot second path on Gazebo

TRAJECTORY CAMERA FRONT (ALGORITHM)		TRAJECTORY CAMERA FRONT (/ODOM TOPIC)	
x	**y**	**x**	**y**
0.09554277	0.02927974	0.03801664	0.01671115
0.60000000	0.05471626	0.65419263	0.01399195
1.10211101	0.00770490	1.03973588	0.00828904
1.81713958	-0.03351129	1.78249209	-0.04054680
2.59935415	-0.03887598	2.50600921	-0.02942213
3.41711658	-0.03186096	3.31943328	-0.01662266
5.19661403	0.02020852	5.09435583	0.01017592
5.41906987	0.68802218	5.33823024	0.60618695
5.58431972	1.63152171	5.49456867	1.56689595

Table 6.3 Coordinates points from the algorithm and from /odom topic for trajectory

The points shown above combined with the rotation component give rise to the graph shown in *Figure 6.14*.

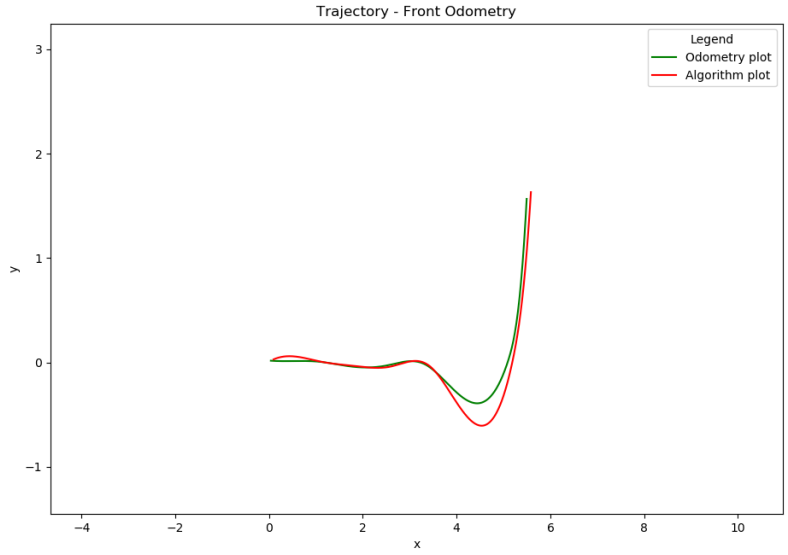

Figure 6.14 Jackal Robot trajectory with rotational component with front-facing camera

After placing the blocks that reproduce the floor with the features inside the hospital and turning the camera downwards towards the tiles, the Jackal is moved along the path drawn in yellow (*Figure 6.15*).

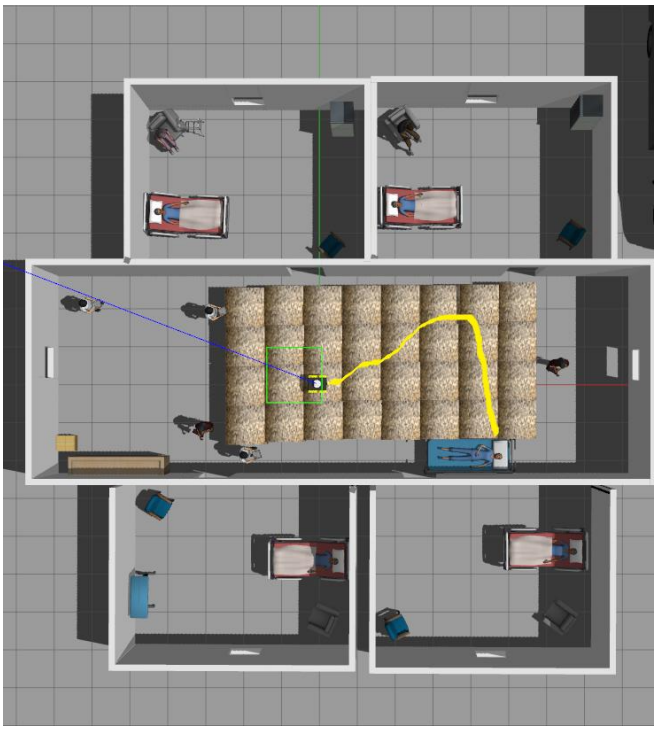

Figure 6.15 Jackal Robot path on Gazebo blocks with rotation component

The points in the Table 6.4, which are the result of both the algorithm and the ROS 2 topic, are given below.

TRAJECTORY CAMERA DOWNWARD (ALGORITHM)		TRAJECTORY CAMERA DOWNWARD (/ODOM TOPIC)	
x	**y**	**x**	**y**
0.13957827	-0.34127593	0.14439722	-0.28769902
1.07342659	-0.02345376	1.06760986	-0.03011315
2.10365839	0.17947256	2.07902104	0.18965106
3.11358037	0.44264963	3.08209456	0.40581956
4.17038563	0.31975910	4.15937858	0.33147782

Table 6.4 Coordinates points from the algorithm and from /odom topic for trajectory (Floor)

The result is the graph shown in the *Figure 6.16* that compares the two trajectories obtained with the sensor facing a feature-rich floor.

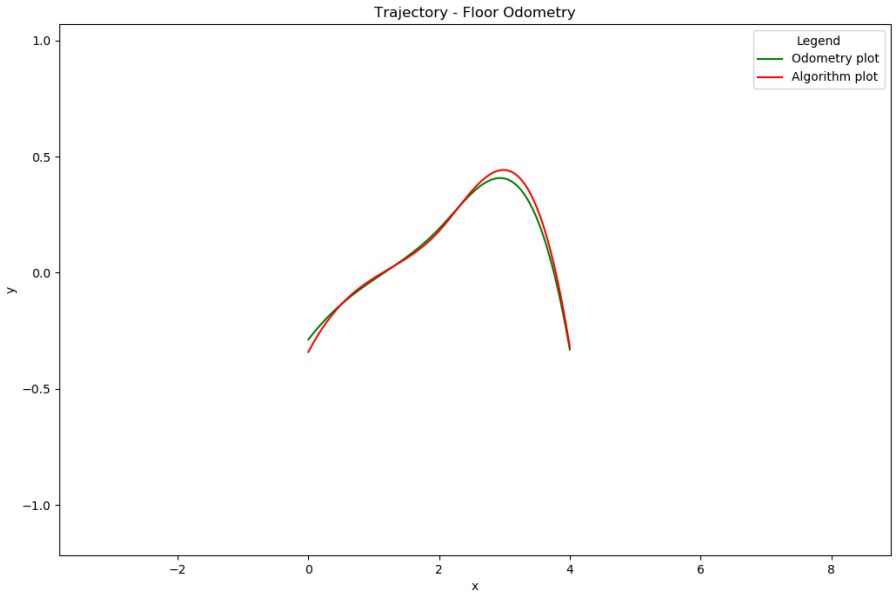

Figure 6.16 Jackal Robot trajectory with rotational component with camera facing downward

6.2 Experiments in Real World

After evaluating the correctness of the algorithm developed on Python, I test it in the real world by making the Intel RealSense d435 camera move through the PIC4SeR environments

In order to achieve better camera performance, an Intel Next Unit Computing (NUC) is used, and the algorithm is launched from there.

Firstly, I connect the algorithm with the sensor topic, then I evaluate the consistency of the points obtained from the algorithm. It is then simulated in a real indoor environment.

Experiments are conducted at the PIC4SeR spaces (*Figure 6.17*) evaluating front-facing sensor and facing downward one.

Figure 6.17 PIC4SeR environment

It consists in moving sensors around the room comparing results from the Visual Odometry algorithm and from odometry from the topic */d435/color*.

Results are consistent, especially those regarding front-facing camera d435. It proved difficult to demonstrate this, as it was necessary to use the Turtlebot robot. In doing so, the odometry was compared and then the trajectory of the system could be obtained.

X coordinate from Algorithm testing in PIC4SeR	Y coordinate from Algorithm testing in PIC4SeR
0.000343	-0.00248
0.924537	-0.00765
1.083741	-0.13013
1.361033	-0.11375
1.536414	-0.202841
1.937569	0.131284
1.645216	-0.183341
1.606682	-0.188375
1.236431	-2.148362
1.194768	-2.165418

Table 6.5 Odometry coordinates in real world simulation with front-facing camera

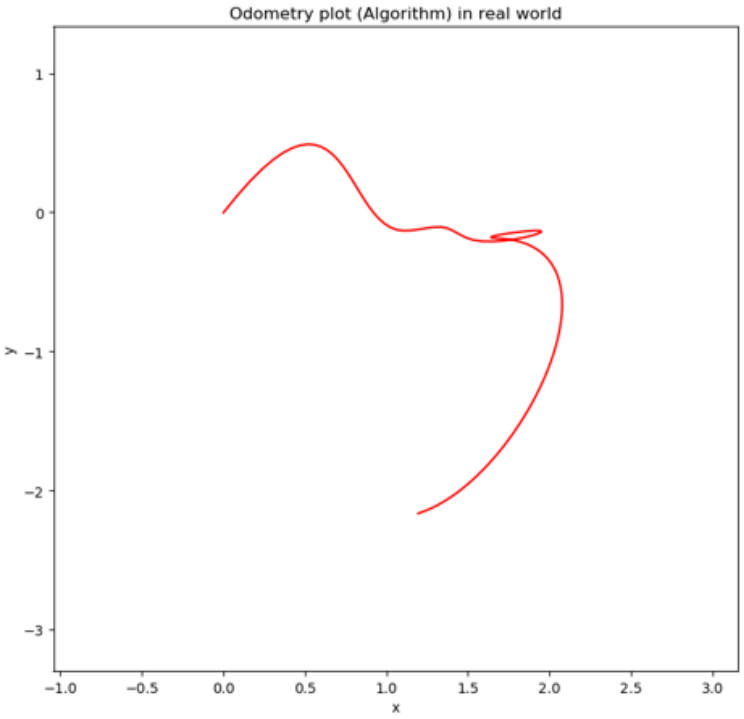

Figure 6.18 Trajectory with rotation component - Real World with front-facing sensor

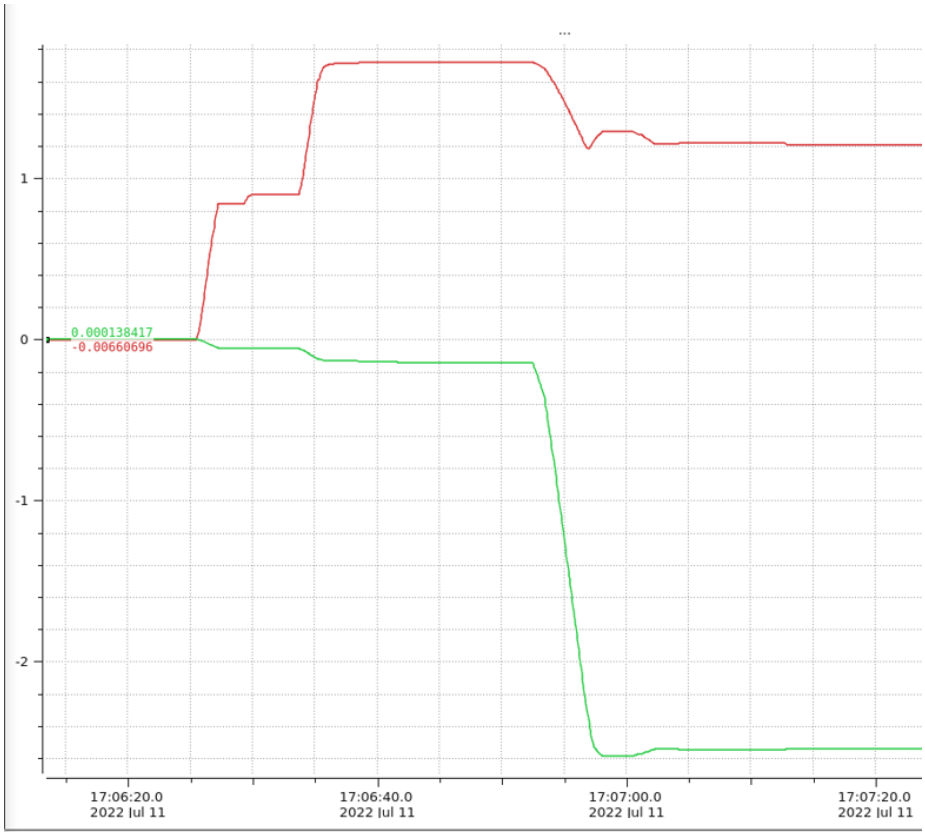

Figure 6.19 Space-Time graph with Turtlebot trajectory

Moving sensor facing downward, due to the presence of floors with few features, there is a risk of not achieving the minimum number of matches to obtain an odometry point from the algorithm.

Chapter 7

Conclusion

From the graphs shown in the previous chapter, it is possible to observe how the presence of features in the surrounding space affects the performance of the algorithm.

In fact, from the Gazebo simulation, odometry is more accurate when the sensor is facing downwards than when the sensor is facing frontwards.

It demonstrates how a low texture makes it difficult and less accurate to locate a system.

The idea of positioning the camera towards the floor certainly improves performance and it consists in a progressive step with the aim of overcoming this challenge.

The following section shows the graphs with the error determined by the deviation between the point obtained by the Visual Odometry algorithm and the Odometry point of the /odom topic in the Gazebo simulation, considering the two cases of the camera facing frontwards and then towards the floor; as regards the real-world test, the consistency of the points obtained and the trajectory with respect to the displacement within the indoor environment is demonstrated.

7.1 Results Analysis

Analysing the graphs in the previous chapter, it is evident how the higher number of features affects the accuracy of odometry.

Another property of the algorithm that emerges when looking at the graphs concerns the low precision of the odometry points of the linear path rather than a trajectory with a rotation component.

During simulation in the real world, the trajectory travelled by the camera is consistent with the trajectory represented in the algorithm graph. Some problems may occur in the absence of features on the floor, in fact when the camera is facing the floor, the algorithm often crashes. The reason for this is that the sensor used does not detect contrast variations or stains on the floor tiles because it is too high.

The above conclusions are also demonstrated through the study of absolute error. In fact, by analysing the absolute errors from the tables of odometry values, the following emerges: difference between visual odometry value obtained by the /odom topic on Gazebo and visual odometry value obtained by the algorithm result is shown below.

X ABSOLUTE ERROR (cm)	Y ABSOLUTE ERROR (cm)
-0,917411	-0,27609
0,039736	-0,53887
1,103898	-0,79991
3,147216	-0,35494
2,555215	1,208136
1,971101	2,545163
-0,962386	4,634498
-0,556794	2,531041
-0,867267	0,124462
-3,365543	-0,37875
-5,438128	-2,07188
-5,560177	-0,80033
-3,44218	-5,69745
-3,034939	-5,02351
-1,6144	-3,69004
-0,23477	-1,97558
0,276534	0,012008
1,046716	1,861811
1,846442	3,111155
2,89687	1,563944
3,612411	0,631332
2,347112	-0,99615
-0,29049	-1,45495
-0,152667	-0,65457

Table 7.1 Absolute Errors (first case)

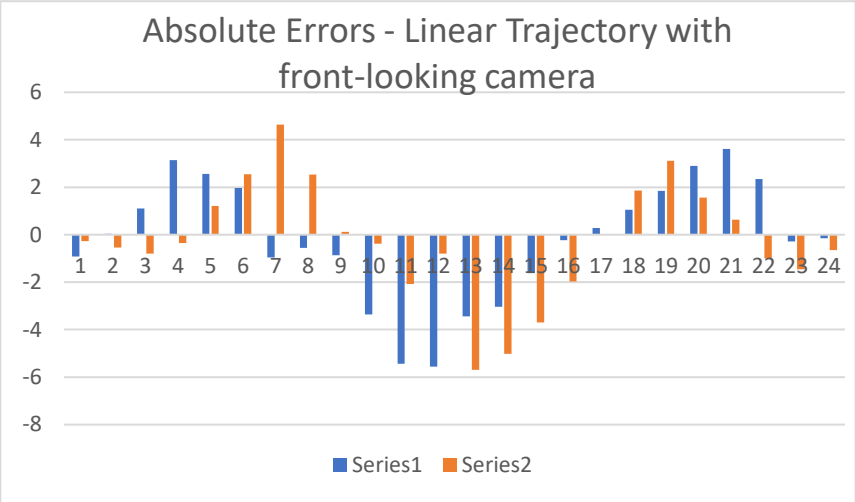

Figure 7.1 Absolute Errors plot in first case

X ABSOLUTE ERROR (cm)	Y ABSOLUTE ERROR (cm)
-0,48862983	0,0018042
-0,16384145	0,00525012
-0,059148	0,01686777
-0,07574861	0,01894369
-0,30519915	0,11323102
-0,37094611	0,09307918
-0,36094181	0,05553105
-0,07243318	0,03071037
-0,00897959	-0,0257288
-0,02724285	-0,06916289
-0,0395749	-0,04494497
-0,1753963	-0,11208162
-0,06655675	-0,03533356
-0,04380132	-0,07836049

Table 7.2 Absolute Error (second case)

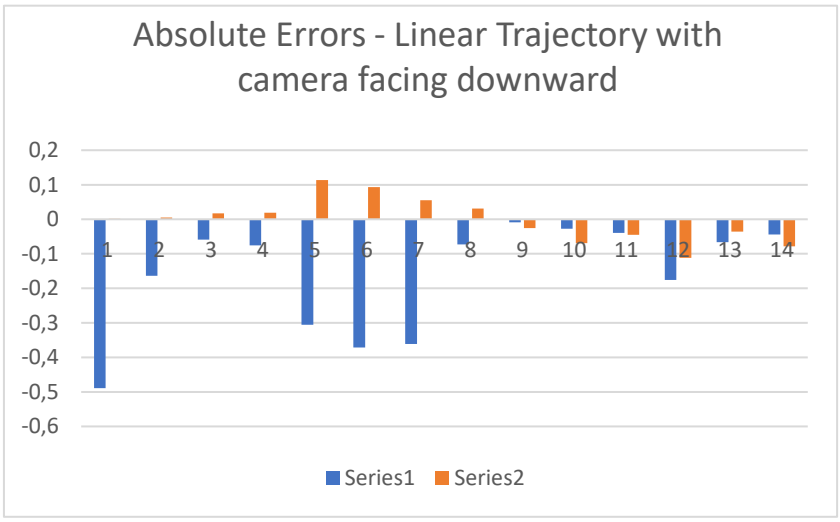

Figure 7.2 Absolute Errors graph in second case

X ABSOLUTE ERROR (cm)	Y ABSOLUTE ERROR (cm)
-0,05752613	-0,01257
0,05419263	-0,04072
-0,06237513	0,000584
-0,03464749	-0,00704
-0,09334494	0,009454
-0,0976833	0,023238
-0,1022582	-0,01003
-0,08083963	-0,08184
-0,08975105	-0,06463

Table 7.3 Absolute Errors (third case)

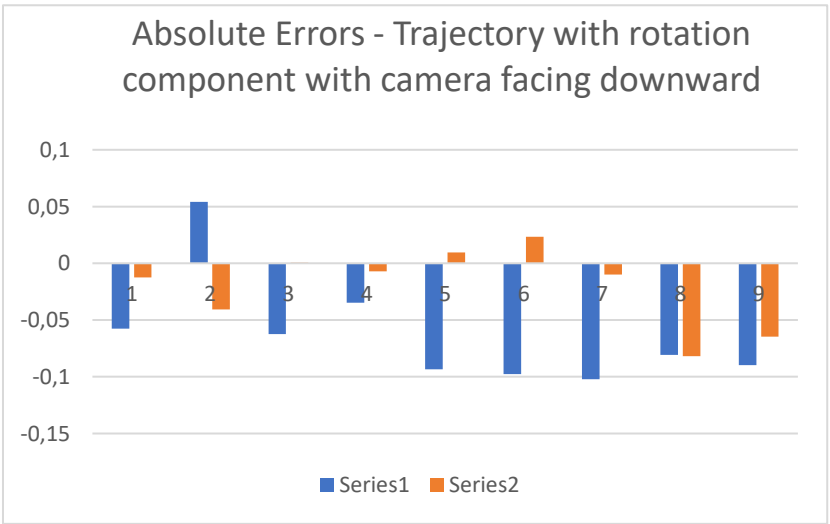

Figure 7.3 Absolute Errors graph in third case

X ABSOLUTE ERROR (cm)	Y ABSOLUTE ERROR (cm)
0,00481895	0,05357691
-0,00581673	-0,00665939
-0,02463735	0,0101785
-0,03148581	-0,03683007
-0,01100705	0,01171872

Table 7.4 Absolute Errors (Fourth case)

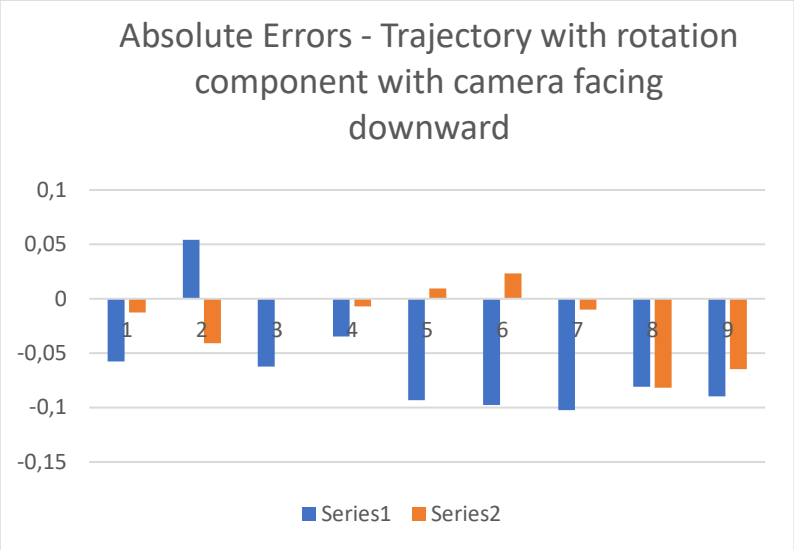

Figure 7.4 Absolute Errors graph in fourth case

7.2 Future Works

In real-world simulation, from the results obtained, the Intel RealSense d435 camera has difficulty detecting floor features, particularly those that are very small or have little contrast.

The ideal would be to use a sensor placed as close as possible to the floor in order to be sure that the features are detected, and the accuracy and robustness of the algorithm is increased.

In future work, the same algorithm will be run using a sensor suitable for working a few centimetres from the floor and facing towards it. The same experimental procedures described in the previous chapters will then be reproduced and the Visual Odometry will be verified from the inputs captured by this sensor.

By doing so, the system in question with the algorithm and appropriate sensor will be able to localise a wheelchair, providing greater safety for the user with disabilities.

Bibliography

[1] ALBA Robot, *https://www.alba-robot.com/.*

[2] Mohammed Hayyan Al Sibai, Sulastri Abdul Manap, *A Study of Smart Wheelchair Systems.*

[3] Jesse Leaman and Hung Manh La, *A Comprehensive Review of Smart Wheelchairs: Past, Present, and Future.*

[4] Siegwart, Roland & Nourbakhsh, Illah R., *Introduction to Autonomous Mobile Robots*.

[5] Hamid Taheri, Zhao Chun Xia, *SLAM; definition and evolution.*

[6] Dr. Emna Baklouti, Prof. Nader Ben Amor & Prof. Mohammed Jallouli, *Autonomous wheelchair navigation with real time obstacle detection using 3D sensor.*

[7] Martin Rowe, *Sensor basics: Types, functions and applications.*

[8] Takashi Gomi and Ann Griffith, *Developing Intelligent Wheelchairs for the Handicapped*.

[9] Aniket Murarka, Shilpa Gulati, Patrick Beeson and Benjamin Kuipers, *Towards a Safe, Low-Cost, Intelligent Wheelchair*.

[10] The difference between indoor electric wheelchairs and outdoor electric wheelchair - *https://www.uk-wheelchairs.co.uk/blog/the-difference-between-indoor-electric-wheelchairs-and-an-outdoor-electric-wheelchairs.*

[11] Mohammad O. A. Aqel, Mohammad H. Marhaban, M. Iqbal Saripan and Napsiah Bt. Ismail, *Review of visual odometry: types, approaches, challenges, and applications.*

[12] Davide Scaramuzza and Friedrich Fraundorfer, *Visual Odometry: Part I - The First 30 Years and Fundamentals.*

[13] Steven Connor, *Corridors - http://www.stevenconnor.com/corridors/*

[14] Davide Scaramuzza, *Tutorial on Visual Odometry - https://rpg.ifi.uzh.ch/docs/Visual_Odometry_Tutorial.pdf*

[15] Hans P. Moravec, *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover*.

[16] D. Nister, O. Naroditsky and J. Bergen, *Visual Odometry*.

[17] F. Bellavia, M. Fanfani, C. Colombo, *Selective Visual Odometry for Accurate AUV Localization*.

[18] Ali Barzegar, Oualid Doukhi and Deok-Jin Lee, *Design and Implementation of an Autonomous Electric Vehicle for Self-Driving Control under GNSS-Denied Environments*

[19] Cosimo Patruno, Roberto Colella, Massimiliano Nitti, Vito Renò, Nicola Mosca and Ettore Stella, *A Vision-Based Odometer for Localization of Omnidirectional Indoor Robots.*

[20] Shashi Poddar, Rahul Kottath, Vinod Karar, *Evolution of Visual Odometry Techniques.*

[21] J. K. Aggarwal and N. Nandhakumar, *On the computation of motion from sequences of images-a review.*

[22] N. W. Oumer and G. Panin, *3D point tracking and pose estimation of a space object using stereo images.*

[23] Image by Tomasz Malisiewicz - *https://twitter.com/quantombone*

[24] A. Comport, E. Malis, and P. Rives, *Accurate quadrifocal tracking for robust 3d visual odometry*.

[25] Zhongli Wang, Yan Chen, Yue Mei, Kuo Yang and Baigen Cai, *IMU-Assisted 2D SLAM Method for Low-Texture and Dynamic Environments*

[26] Nan Yang, Rui Wang, Xing Gao and Daniel Cremers, *Challenges in Monocular Visual Odometry: Photometric Calibration, Motion Bias and Rolling Shutter Effect*

[27] Shichao Yang, Yu Song, Michael Kaess, and Sebastian Scherer, *Pop-up SLAM: Semantic Monocular Plane SLAM for Low-texture Environments*

[28] Huizhong Zhou, Danping Zou, Ling Pei, Rendong Ying, Peilin Liu, Member, IEEE, and Wenxian Yu, *StructSLAM: Visual SLAM With Building Structure Lines*

[29] M.M. El-gayar, H. Soliman, N. Meky, *A comparative study of image low level feature extraction algorithm*

[30] Daniele Licitra, *CVideoPlayer: feature FAST, SIFT e SURF*

[31] ORB (Oriented FAST and Rotated BRIEF) - *https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html*

[32] Cuiyin Liu, Jishang Xu, Feng Wang, *A Review of Keypoints' Detection and Feature Description in Image Registration*

[33] Oriented FAST and Rotated BRIEF using OpenCV - *https://github.com/deepanshut041/feature-detection*

[34] Amila Jakubovic, Jasmin Velagìc, *Image Feature Matching and Object Detection using Brute-Force Matchers*

[35] Essential and Fundamental Matrices, *http://robotics.stanford.edu/~birch/projective/node20.html*

[36] Stefan Winkler, Hang Yu, ZhiYing Zhou, *Tangible Mixed Reality Desktop for Digital Media Management*

[37] TechTip: Accuracy, Precision, Resolution and Sensitivity, *https://www.mccdaq.com/TechTips/TechTip-1.aspx*

[38] Sensori: cosa sono e loro applicazioni attuali, *https://www.internet4things.it/iot-library/sensori-cosa-sono-e-loro-applicazioni-attuali/*

[39] Visual Sensors, *https://werthinc.com/visual-sensors*/

[40] Baichuan Huang, Jun Zhao, Jingbin Liu, *A Survey of Simultaneous Localization and Mapping with an Envision in 6G Wireless Networks*

[41] Yuncheng Lu, Zhucun Xue, Gui-Song Xia & Liangpei Zhang, *A survey on vision-based UAV navigation*

[42] Intel RealSense Depth Camera D435i, *https://www.intelrealsense.com/depth-camera-d435/*

[43] IMU – Inertial Measurement Unit, *https://www.sbg-systems.com/inertial-measurement-unit-imu-sensor/*

[44] Inertial Measurement Unit, *https://en.racelogic.support/VBOX_Automotive/Product_Info/Modules_and_Accessories/Inertial_Measurement_Units*

[45] ROS Nodes, Topic, Messages, *https://subscription.packtpub.com/book/hardware-&-creative/9781788479592/1/ch01lvl1sec13/ros-nodes-topics-and-*

messages#:~:text=ROS%20topics&text=The%20information%20in%20ROS%20is,is%20published%20by%20the%20node.

[46] Jackal, *http://wiki.ros.org/Robots/Jackal*

[47] Package Summary, *http://wiki.ros.org/urdf*