POLITECNICO DI TORINO

Master of Science degree in
Mechatronic Engineering

Master Thesis

Person following and visual relocalization for indoor service robotics

Politecnico
di Torino

Supervisor:
prof. Marcello Chiaberge

Candidate:
Paolo Ruggeri
S278528

Academic Year 2020-2021

Abstract

In the last decades the average quality of life improved all over the globe, leading to an increase of elderly people, that often need assistance in their everyday life. Contextually the field of service robotics has been trying to fulfill this social need, providing solutions which can safely coexist in the house. Inside this scenario, PIC4SeR (PoliTO Interdepartmental Centre for Service Robotics) has been putting efforts in a wide project, with the aim of assisting elder people in their daily routines.

In particular, the goal of this thesis was to develop a series of software applications to be run on an Unmanned Ground Vehicle (UGV), designed to roam safely in a domestic environment.

Person following is the task of navigating while maintaining the user framed as he walks. To do so, the robot detects the presence of a person in the camera stream and recognizes the main parts of the human body, exploiting an already trained, light and low resource demanding neural network: PoseNet. Adopting SORT (Simple Online and Real time Tracking), the presence of the same person in multiple successive frames is tracked. Finally, the relative distance between the robot and the person is computed.

The other main subject of this work was relocalization, correcting the odometry error accumulated while roaming. At first it has been performed via markers. Adopting the April tag system, knowing a priori position and orientation of the marker, it is possible to derive the robot global position and orientation very precisely.

Another approach to relocalization was then tried in the last part. Acquiring a database of images when the robot turns on, it can later be compared with a query frame. Finding features matching in two frames, an estimation of the relative pose difference between them can be estimated. This framework, although less precise, has the advantage of working in an unstructured environment, unlike the previous one. It also leaves space for improvements in the way matching couples are selected.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Objective of the thesis

In the last decades the average quality of life improved all over the globe, leading to an increase of elderly people, that often need assistance in their everyday life. Contextually the field of service robotics has been trying to fulfill this social need, providing solutions which can safely coexist in the house. Inside this scenario, PIC4SeR (PoliTO Interdepartmental Centre for Service Robotics) has been putting efforts in a wide project, with the aim of assisting elder people in their daily routines.

In particular, the goal of this thesis was to develop a series of software applications to be run on an Unmanned Ground Vehicle (UGV), designed to roam safely in a domestic environment.
Person following is the task of navigating while maintaining the user framed as he walks. To do so, the robot detects the presence of a person in the camera stream and recognizes the main parts of the human body, exploiting an already trained, light and low resource demanding neural network: PoseNet. Adopting SORT (Simple Online and Real time Tracking), the presence of the same person in multiple successive frames is tracked. Finally, the relative distance between the robot and the person is computed.
The other main subject of this work was relocalization, correcting the odometry error accumulated while roaming. At first it has been performed via markers. Adopting the April tag system, knowing a priori position and orientation of the marker, it is possible to derive the robot global position and orientation very precisely. This is accomplished by firstly acquiring the pose of the tag in the robot reference frame, and then use these measurements to get the robot global pose.
Another, more complex approach to relocalization was then tried in the last part. Acquiring a database of images when the robot turns on, it can later be compared with a series of query frames obtained making the robot spin around when it comes back to what it thinks is the starting position. After extracting significant features

from the environment with SIFT and once some features matching in two frames are found, an estimation of the relative pose difference between where they were taken can be done. This framework, although less precise, has the advantage of working in an unstructured environment, unlike the previous one. It also leaves space for improvements in the way matching couples of images are selected.

1.2 Organization of the thesis

The thesis is composed of seven chapters, as follows:

- In the first chapter a general overview of motivations and achievements of the work done is given, and the following chapters are briefly presented.
- In the second chapter a theoretical introduction about machine learning can be found.
- The third chapter contains an overview on the different types of robotic platforms and their sensors. Then ROS (*Robotic Operative System*) is presented, with all the components of its framework. Finally, the actual hardware used in the project is described.
- The fourth chapter introduces the problem of person recognition and following. Then, the tools adopted in this work are presented, alongside their implementation.
- In the beginning of the fifth chapter an introduction about the problem of relocalization is given. Then it is presented the first approach used to tackle it using markers, in particular April tags.
- In the sixth chapter a more challenging framework is illustrated. By extracting features with SIFT, two frames can be found matching and via the 5 point algorithm the rotation matrix linking the two camera reference frames can be computed. Finally, exploiting information coming from the depth camera, the translation vector can be derived.
- In the seventh and last chapter, the results of the work are recapped, and some ideas for future works are presented.

Chapter 2

Machine learning

Machine Learning is simply speaking the development of computer programs able to improve by themselves, learning from experience. Nowadays it is referred as one of the pillars of the technological advancement, alongside Artificial Intelligence, Internet of Things (IoT), Big Data and Cloud Computing to name few other. In reality Machine Learning has been around since the last century. The term was coined by Arthur Samuel in 1952, who while working at IBM developed an algorithm able to learn to play checkers. After the initial enthusiasm, a period of delusion came in the 70s, caused by the frustration of the poor results obtained. However in the mid 80s a new wave of interest revived the field of research. It was caused by the combined application of neural networks and back propagation, alongside the improvement in computational power. New theoretical contributions led to an exponential growth of the field, which rapidly found usage in everyday applications. One of the very first was the anti spam filter in email inboxes. Today a huge emphasis has been put on Natural Language Processing. It consists in giving computers the ability to understand text and recently even speech in a similar way human beings do. With this, it is possible to develop algorithms that extrapolate information from a written text published on internet, to summarize or categorize it. Automatic translation improved dramatically in the latest years, and the so called chat robots became common in the support sections of companies, while vocal assistants invaded our houses. Inside companies, Machine Learning is used in strategical decision making, included fixing prices according to demand of products. In finance, there is a huge interest in predicting future trends of the market. Other applications can be found in search engines, or in the "recommended for you" section popping up in every web service.

2.1 Characteristics

A formal definition of Machine Learning was given by Tom M. Mitchell: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E" [28].

The task T can be of various types. To name a few of them, in *classification* the algorithm assigns to the input data a label which can be either binary (e.g. is a cat present in the picture?) or identifying multiple classes (e.g. which objects are present?). A Machine Learning algorithm can be trained on data from the past to build a model and predict the future (e.g. guess financial trends), and this is called *regression*. Another task is *anomaly detection*, such as credit card fraud detection based on a model of the user habits.

The performance measure P is in general the accuracy, build around the difference between the label generated and the ground truth value. Usually this is performed with a test set, different and much smaller than the one used to train the model.

The experience E is the core of ML. The training is usually performed providing a large training set to the algorithm, and comparing its predicted label to the ground truth value for each instance, it builds its own model. If the learning process is monitored by a human, it is called *supervised learning*, and unsupervised otherwise. Another way is *reinforcement learning*, in which the algorithm proceeds with a trial and error and receives continuously a feedback on the outcome it has given.

2.1.1 Underfitting and overfitting data

The major challenge in machine learning is *generalization*, that is the ability to perform well on previously unseen inputs. As mentioned before, algorithms are trained on some large training set, and an error can be derived summing in some way the difference between output and ground truth values for each instance. Rationally the smaller the error, the better is the algorithm. However to evaluate how good it performs on unseen data, the algorithm is provided also with a test set, and a test error can be derived in the same way. Naturally, this one will be larger than the training error.

In the end the objective is to tune the model to achieve a small training error while minimizing the difference between it and the test error. In practice it is very difficult to do both altogether. To achieve a small training error, the algorithm will specialize on the training instances, and will result stiff to previously unobserved inputs. Thus generalization will be lost and the difference between the two errors will be large. This phenomenon is called *overfitting*. On the other hand, a model with low capacity, not able to fit a wide variety of functions, will end up with a large training error as the model is not able to adapt to the training instances. This

phenomenon is called *underfitting*. Figure 2.1 presents the three levels of fitting. On the left, the model chosen was a linear function that cannot represent the curvature of the training data (underfitting). On the right, the interpolation with a high degree polinomial function gave rise to overfitting. Comparing them to the center image, it can be noted that the third picture present a completely different behaviour especially on the leftmost part, where it is decreasing and increasing rapidly, unlike the good approximation with a quadratic function.

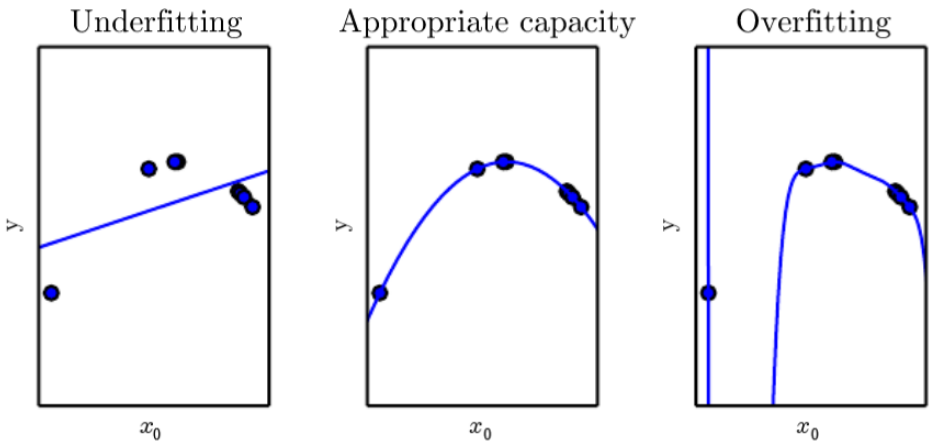


Figure 2.1: The three different levels of fitting [18]

2.2 Classical ML models

They were historically the first developed and can tackle easier problems faster than more complex algorithms.

2.2.1 Linear Regressor

It is the simplest algorithm in Machine Learning. The name is self explanatory: it performs regression, so it provides as output a continuous value, result of a weighted sum of the input. The weights are the parameters tuned in the learning phase. Denoting with $\widehat{y}$ the predicted value, n the number of input features, x_i the i-th feature value and θ the tunable parameters (where θ_0 is called bias), it stands:

$$\widehat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_n x_n = \boldsymbol{\theta} \cdot \mathbf{x} \tag{2.1}$$

The chosen θ will be the one that minimizes the error between predicted ($\widehat{y}$) and ground truth (y) value. Usually the mean squared error is computed:

$$MSE(\boldsymbol{X}, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{\theta}^T x^{(i)} - y^{(i)})^2 \tag{2.2}$$

Since the model is linear, a closed form solution exists to the optimization problem, and it is the so called least squared solution:

$$\widehat{\boldsymbol{\theta}} = arg\min_{\boldsymbol{\theta}} MSE(\boldsymbol{X}, \boldsymbol{\theta}) = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T y \tag{2.3}$$

2.2.2 Support Vector Machine

SVM is a supervised machine learning algorithm able to tackle both classification and regression tasks requiring low computational power. This is granted by the fact that the model considers only the closest points (called support vectors) near the decision boundary, discarding the others (see Figure 2.2). The decision boundary is

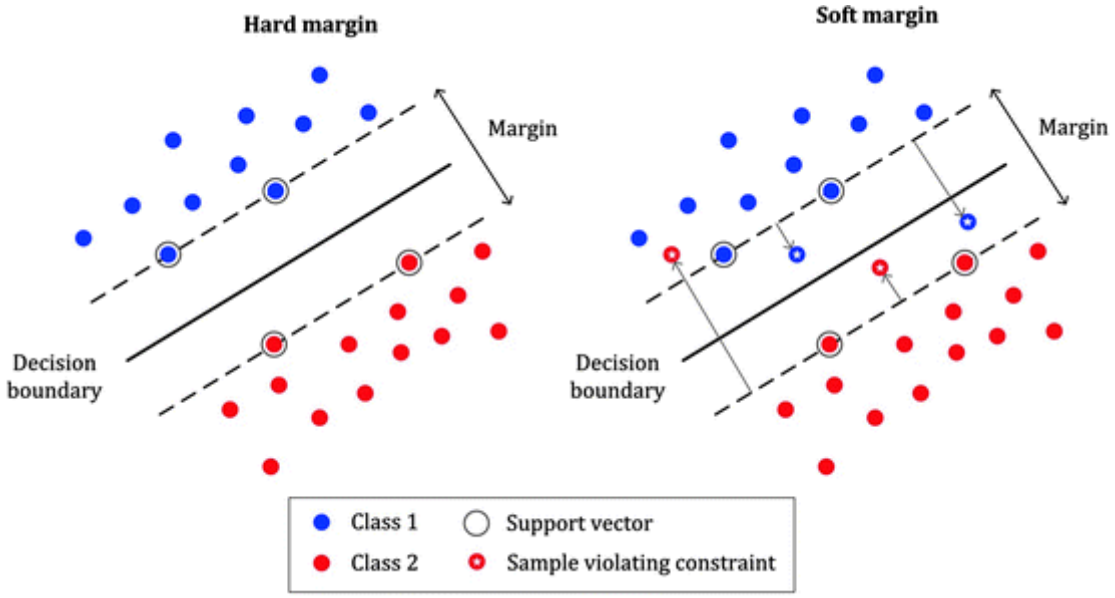


Figure 2.2: Hard and soft margin in Support Vector Machine [26]

an hyperplane of dimension $n-1$, where n is the number of features of the inputs, so that if x is a vector of dimension 2, the decision boundary will be a straight line, while an input vector of dimension 3 will require a plane and so on. The mathematical structure is similar to the linear regressor:

$$f(x) = \mathbf{w}^T\mathbf{x} + b \tag{2.4}$$

where $\mathbf{w}^T\mathbf{x} = 0$ is the equation of the decision boundary. Simply speaking, the algorithms searches for the largest possible region between the support vectors, trying to divide the input data in separate classes. If it is not possible to rigidly isolate groups, a soft margin approach can be applied. In this way some outliers are allowed, and their quantity will become the quantity to be minimized.

2.2.3 K-nearest neighbor

K-nearest neighbors is another algorithm that can be applied for both classification and regression. At its core there are no parameter to tune, as there is no actual training phase indeed. What it does is trying to find the k-nearest neighbors in the training data space for the inference instance. In regression, the average between the y label values of the neighborhood is computed and is assigned to x, while for classification the most common label is taken as output. The distance metric to use should be evaluated carefully, as it is the basis of the model.

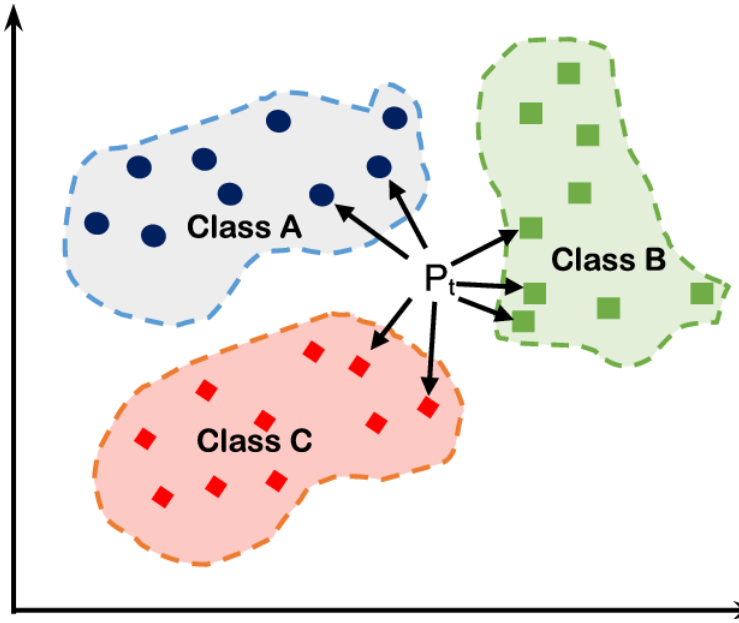


Figure 2.3: Visual representation of the k-nearest neighbor algorithm [21]

2.3 Deep learning models

2.3.1 Artificial neuron

Deep learning is a branch of ML which tackles more complex problems with models based on the artificial neuron. The name comes from the strong similarity with the biological neuron. In fact, as it can be seen in Figure 2.4, a parallel can be traced between the two structures. The biological neuron receives electrical inputs from other neurons via its ramified dendrites, capturing neurotransmitters coming from the axons of other neurons. When the electrical potential in the cell body overcomes a threshold, an impulse is transmitted through the axon to ramified synapses, which constitute the links to other cells.

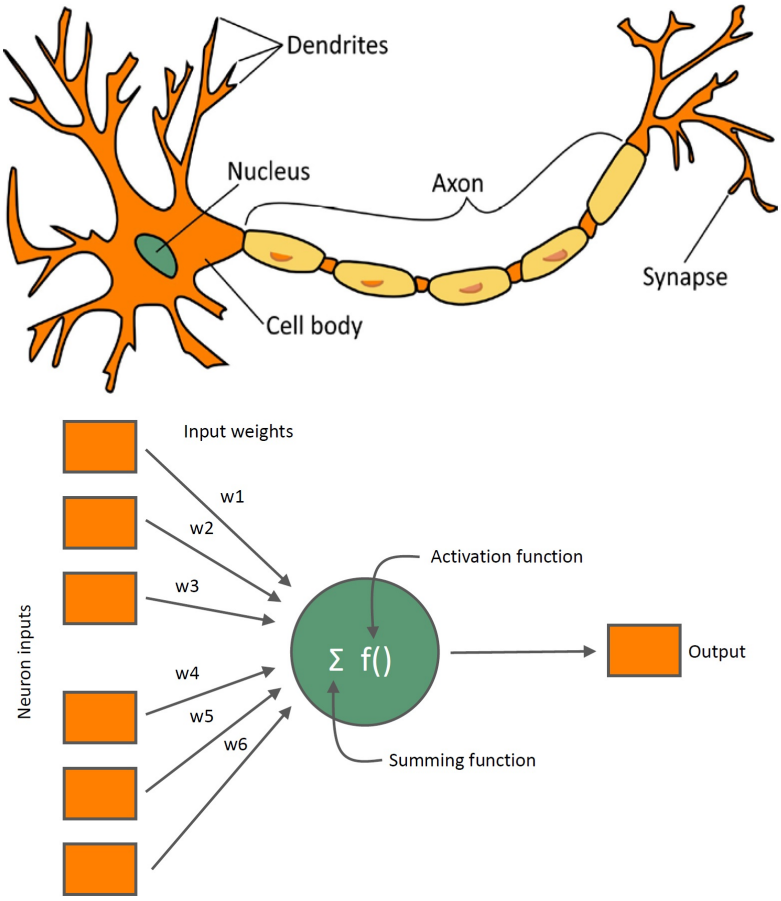


Figure 2.4: Comparison between a biological and an artificial neuron [24]

The first model of an artificial neuron, realized by McCulloch and Pitts in 1943, was later improved in 1958 by Rosenblatt, who renamed it *perceptron*. As it is a binary classifier and given its working principle, it is also called Threshold Logic Unit. In fact it receives binary signals coming from other TLUs, applies a weight w_i in each "dendrite" and then the resulting quantities are sum in the nucleus. The output signal is then forwarded based on an activation function, a basic threshold step function (Figure 2.5).

$$f(\mathbf{x}) = \begin{cases} 1 & if \quad \sum_i w_i x_i > b \\ 0 & if \quad \sum_i w_i x_i \leq b \end{cases} \tag{2.5}$$

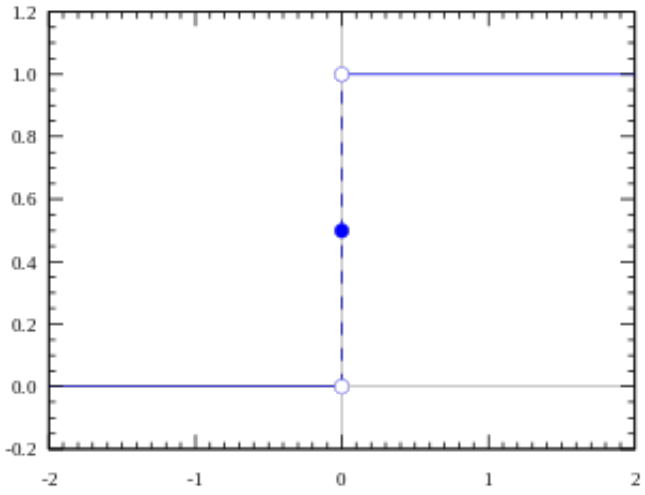


Figure 2.5: Threshold function (here $b = 0$) [40]

2.3.2 Artificial neural networks

The composition of artificial neurons into a well defined model gives rise to a neural network. A general representation is presented in Figure 2.6. Neurons are disposed in columns to form layers. Each neuron of a layer is linked to every element of the previous layer via weighted connections. Three types of layer can be distinguished:

- The *input layer*, formed by the input signals x_i.
- The *hidden layer*, which is usually repeated multiple times in the structure. As their number increases, the more complex the model becomes, and it can achieve better results. An hidden layer is identified by its number of neurons, a weight matrix, a bias vector, and the type of activation function.
- The *output layer* is the last layer of the network, providing the results of the inference as output.

In artificial neural networks the learning process consists in minimizing a cost function, involving the difference between generated output $\widehat{y}$ and ground truth value y. A frequently used cost function is:

$$C(w, b) = \frac{1}{2n} \sum_{x} ||y - \widehat{y}||^2 \tag{2.6}$$

The network is covered backwards, as the information about the cost is used to tune weights w and biases b in an operation called *back propagation*.

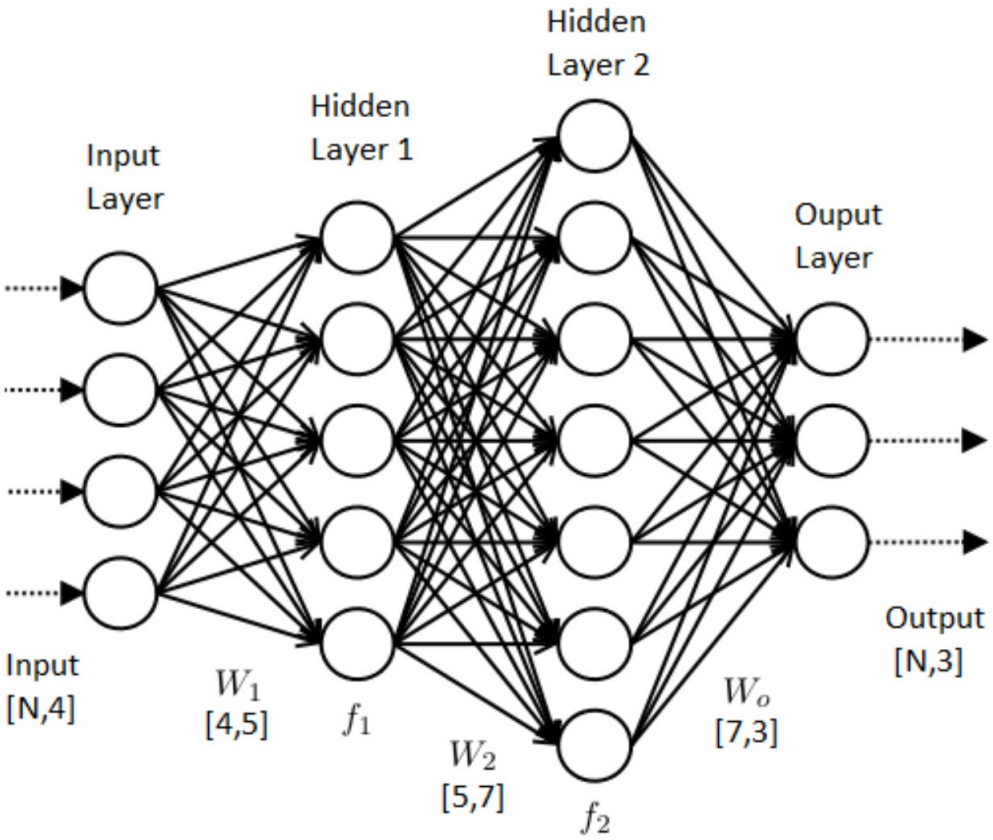


Figure 2.6: Graphical representation of an ANN [2]

Activation functions

The choice of the activation functions determines significantly the model. One of the problem of the perceptron was that having an activation function with binary output, a small change in the values of the weights and of the bias could lead to drastic changes in the overall behavior of the model. Thus, different functions with a non binary output have been used in place of the step function, while preserving the simplicity fundamental to keep the computational cost low.

For the hidden layer neurons, the most adopted activation function is the *sigmoid function*, usually denoted with σ:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.7}$$

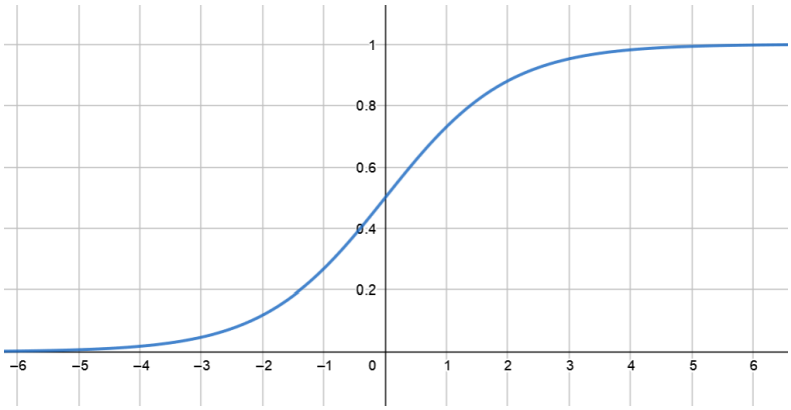


Figure 2.7: Sigmoid function

As the output varies smoothly in the interval, the problems previously described about the perceptron are largely reduced.

With a similar shape to the sigmoid function, the *hyperbolic tangent* is another widely used activation function. The main difference is that the output range is $[-1; 1]$ instead of $[0; 1]$.

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.8}$$

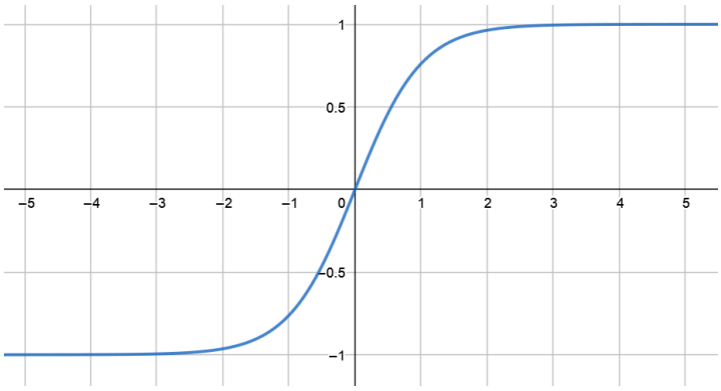


Figure 2.8: Hyperbolic tangent function

The *Rectified Linear Unit function* (ReLU) is another widely chosen activation function. Its main advantage is its low computational cost, while preserving non linearity for good modelling properties.

$$\sigma(z) = max(0, wx + b) \tag{2.9}$$

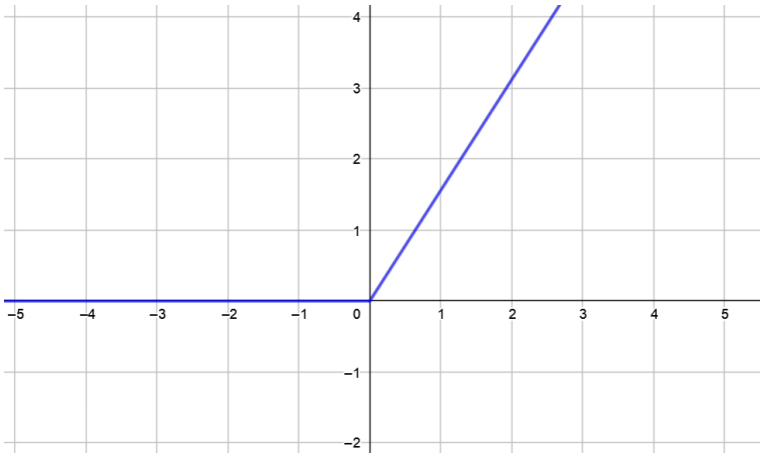


Figure 2.9: Rectified Linear Unit function

Finally, the *softmax* function is an activation function used in particular for the output layer, especially in case of classification. In fact, its output can be interpreted as the probability of an input instance to belong to one class. It is also know as normalized exponential function, and for each j-th element of the input vector z, the expression is:

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{2.10}$$

Notice that $\sum_j \sigma(z_j) = 1$ What the function does in practice is highlighting the biggest element of the vector with respect to the others, identifying, indeed, the most plausible class label.

2.3.3 Convolutional neural networks

Convolutional neural networks are complex ANNs adopted especially to work with images. As the nature of the input is complex, they implement techniques to reduce the complexity of the model, saving computational time. These tools are: *local receptive fields, shared weights* and *pooling*.

First of all, the input image is not decomposed in a column vector of inputs, but its structure is maintained. As a consequence a new type of hidden layer must be introduced, as fully connected ones would not suit the framework. The so called *convolutional layers* consider only a region of pixels at a time, named local receptive field. The convolution operation is illustrated in Figure 2.10. Every receptive field is overlapped with a filter, also called kernel, and the sum of the element wise multiplications is stored to form the input of the next layer. The filter is linked to the feature to recognize, and is in common for the whole layer, unlike fully connected layers where each neuron had its own weights and biases.

Then the next local receptive field is considered, sliding of one pixel column to the right. To reduce the computational cost, a *stride* of 2 can be chosen. The input

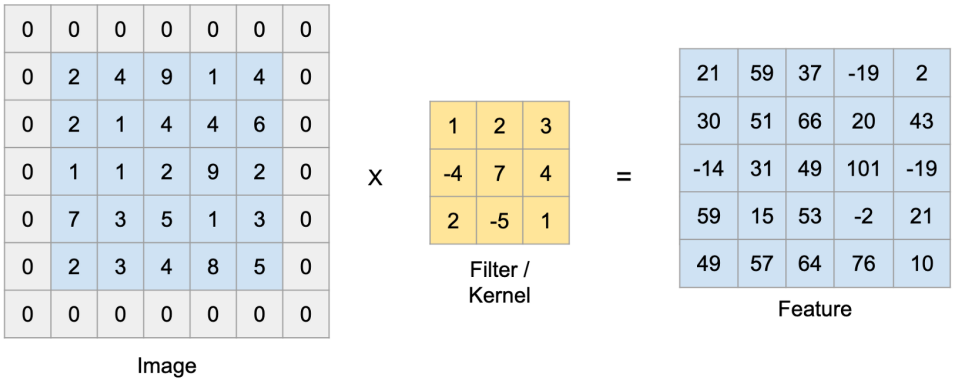


Figure 2.10: Convolution operation in CNN [8]

matrix is often surrounded with a frame of zeros, such that the input values on the edges are equally considered to the ones inside, that would naturally appear in more receptive fields. This operation is called *padding*. To further reduce the complexity of the model, a pooling layer is introduced right after a convolutional layer. The pooling operation considers once again a region of the matrix, and computes one value from it. Usually either the maximum value is extracted, or the average of all the elements is found. In this way the dimensions are further decreased, at the cost of losing information on the precise location of the feature recognized.

Right before the output layer, a fully connected layer is present, with a softmax activation function. In fact, as it can be seen in Figure 2.11 , one characteristic of convolutional neural networks is that the data is progressively flattened thanks to the convolution and pooling operation previously described.

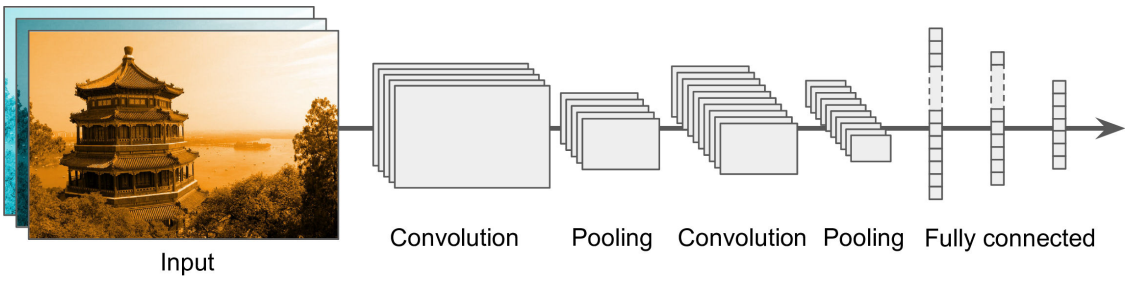


Figure 2.11: Typical structure of a CNN [17]

Chapter 3

Robot platform

3.1 General introduction

A robot is a programmable machine able to autonomously perform repetitive tasks or change its behaviour according to the environment, learning from experience. It is composed by an hardware and a software part. Mechanical links and joints, batteries, electrical motors, electronic boards and sensors all belong to the hardware part, while the software is composed by the microcontroller firmware and the utilities and routines developed to manage the mechanical parts in the execution of the tasks.
As the number of tasks robots can perform is constantly increasing, robotics is an exponentially growing field and different architectures can be identified, each one adapting best to its own application:

- Mobile robots: designed to move in a well-defined environment, these usually wheeled robots have a large application field, from the military to space exploration, from logistics to domestic use.

- Industrial robots: inspired to a human arm, they are formed by a set of mechanical joints and links that allow many degrees of freedom. A gripper is mounted at the end of the arm, which can be interchanged with other tools to properly perform the large variety of tasks requested in an industrial environment.

- UAVs: short for *Unmanned Aerial Vehicles*, commonly called drones, they are classified by the number of propellers. The most known is the quadcopter. Given their ability to fly autonomously and carry relatively small objects, cameras primarily, drones are used in a vast area of applications, ranging from photogrammetry to delivery, from military to recreational and civil use.

- Humanoid robots: manufactured with a head, a torso, two legs and two arms,

they aim to simulate the human execution of tasks, like a standing bipedal movement. They are mostly used in service robotics.

- Bio-inspired robots: given a specific application, they are designed to resemble an animal functional to the goal.

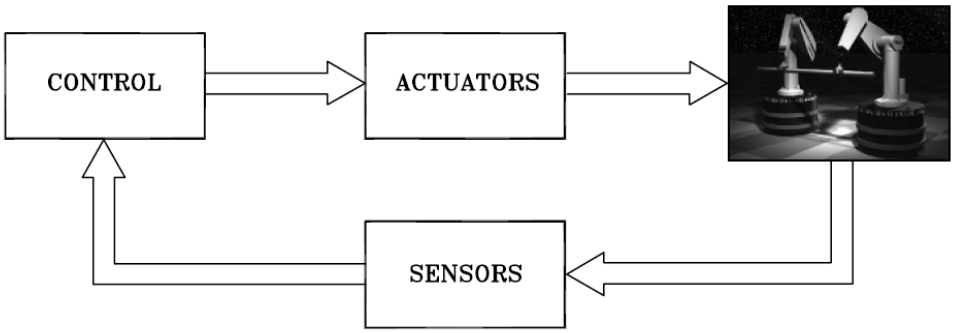


Figure 3.1: Components of a robotic system [38]

All these categories share the common structure of a robotic system, shown in Figure 3.1. In particular the microcontroller sends actuation signals to electrical motors, usually brushless, to change the behaviour of the robot, based on information coming from sensors. They can be distinguished in two classes: proprioceptive and exteroceptive sensors.

3.1.1 Proprioceptive sensors

Proprioceptive sensors acquire information relative to the state of the robotic system. Some examples are battery level indicators, IMU, encoders.

IMU

The *Inertial Measurement Unit* is an electronic device incorporating a gyroscope, an accelerometer and a magnetometer for each of the three main axes. As such, the sensor is able to measure the acceleration and the orientation of the robot, while providing in second instance velocity and position. The problem of working with an IMU is that, as it integrates the acceleration to compute velocity and position, it accumulates error over time. This leads to an increasing difference between the estimated and real robot state.

Encoders

A rotary encoder is an electro-mechanical sensor that translates the angular motion of a shaft into a digital signal. Once attached to the shaft of a wheel, it can be

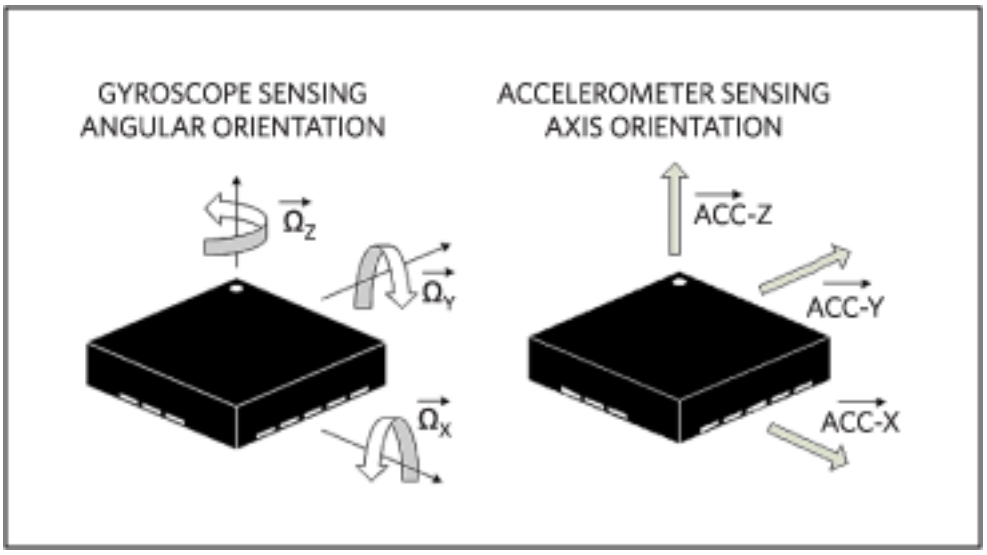


Figure 3.2: Gyroscopes and accelerometers [10]

used to measure the distance covered by the robot, and thus its position. The most common is the optical encoder, consisting in a disc of opaque and transparent areas put between a light source and a photodetector. The transparent areas let the light pass, corresponding to a 1 value bit, while opaque areas which block the light correspond to a 0 value bit. Gray code is used in the alternation of transparent and opaque areas, to minimize the probability of errors occurring in the reception of the information. Optical encoders are of two types:

- Absolute encoders, able to measure angular shift up to 0.06° and hold it even when powered off. The resolution depends on the number of tracks: for example a 12 track encoder will offer $2^{12} = 4096$ segments per revolution, thus providing $\frac{360°}{4096} \approx 0.088°$ of resolution.
- Incremental encoders, on the other hand, give up on the ability to hold the measurement when powered off to provide a faster translation of change in the angular position. The relative difference in output of the two tracks is used to understand the sense of rotation, clockwise or counterclockwise.

3.1.2 Exteroceptive sensors

Exteroceptive sensors acquire information external to the robot, from the environment. Two among the most used ones in robotics are lasers and cameras. Lasers are generally cheaper and can provide 2D or 3D measurement of the environment. The most famous one is 2D LiDAR, *Light Detection And Ranging*, consisting of a

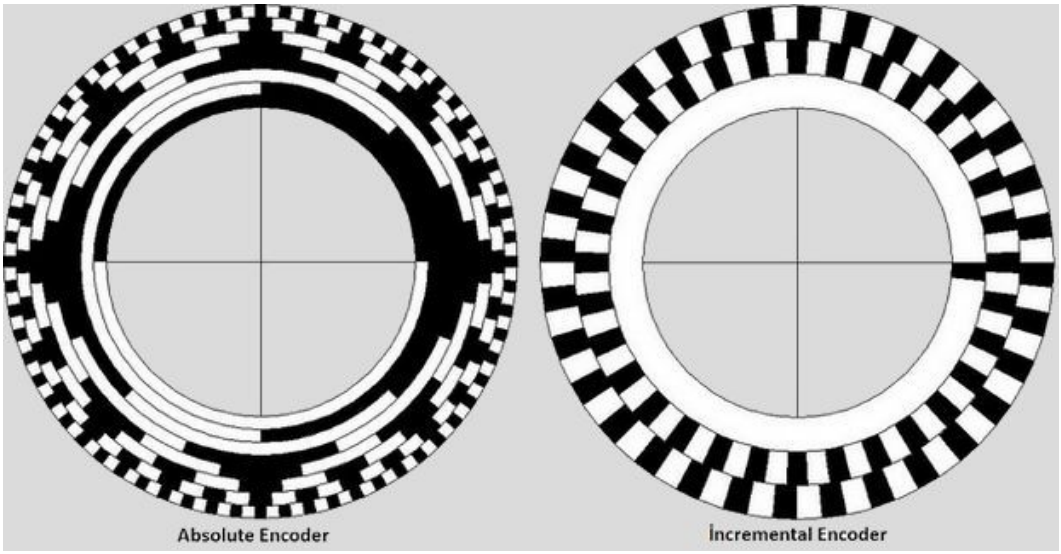


Figure 3.3: Difference between absolute and incremental encoders [13]

laser and a support platform performing a full 360° rotation, obtaining the distance of features at the same horizontal level.

Figure 3.4: An example of a LiDAR sensor [22]

On the other hand, given their price, cameras provide rich information, are light, compact, with low energetic consumption, although the field of view is generally reduced to 90°. They can be used in different configurations:

- Single camera, not providing 3D depth and images need to be compared to the previous ones to be processed.
- Stereo cameras, simulating human vision, provide 3D depth.

- Catadioptric sensors are a mix of mirrors and lenses, provide higher field of view but with lower resolution.
- Multiple camera rigs provide high field of view and uniform spatial resolution.

Alternatively, to solve the problem of 3D depth, an active device can be used, like a depth camera as the Intel RealSense D435i.

3.2 ROS

ROS, *Robot Operating System*, is an open source set of libraries and tools that help in the otherwise difficult realization of a robotic platform. It has become over the years the de facto standard in robotics development, thanks to its continuously growing community where projects, ideas and suggestions are shared.

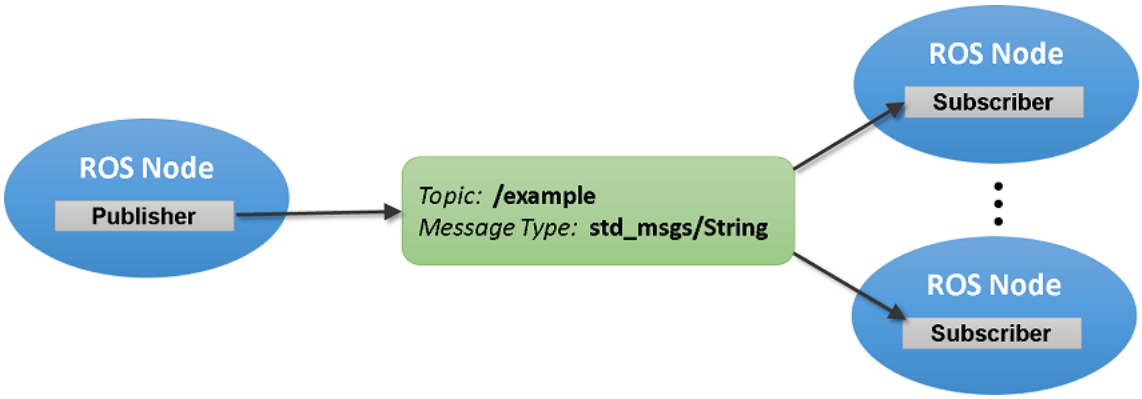


Figure 3.5: Example of data exchange between ROS nodes [14]

The essential components of the framework are:

- ROS node: the software module developed to perform a certain task. It can be coded either in C++ or Python.
- ROS message: it is the data type nodes can send or receive. A part from the large variety of messages that ROS already offers, the user can create new custom messages, according to his necessity.
- ROS topic: it is the bus nodes use to communicate with each other. Nodes can *publish* a message on it, or *subscribe* to the topic to read a message as soon as it is published by another node.
- ROS service: it is another way nodes can use to communicate. The *client* node produces a synchronous call, sending a request to the *server* node, which will reply with some kind of response. They are most suited for calculations to be done quickly and occasionally.

- ROS action: instead of services, actions can be used for long running requests, as they also provide feedback while executing.

A visual representation all the components active in a ROS environment can be obtained at any moment in a graph like the one in Figure 3.5, by using the command *rqt graph*. With the goal of clarity and functional distinction, in ROS the filesystem is divided in packages, which must be compiled before launching the file within them. In the spirit of collaborative development promoted by ROS, hardware and software producers share publicly packages to be used along their products. In particular in this work two packages were used: **turtlebot3**[35] and **ros2_intel_realsense**[34].

3.3 Hardware

3.3.1 Turtlebot3

Born thanks to a collaboration between Open Robotics and ROBOTIS with educational and research purposes, the Turtlebot project helps beginners in their early steps in robotics, providing a customizable starter kit. Arrived at its third gener-

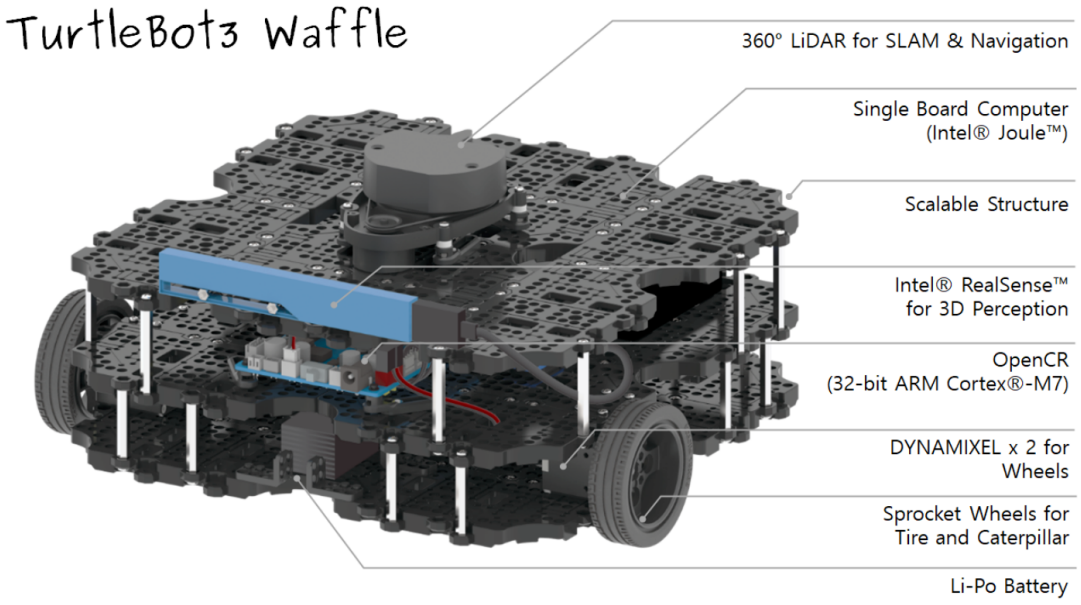


Figure 3.6: Turtlebot3 Waffle [39]

ation, the Turtlebot family counts three members: Burger, Waffle and Waffle Pi. All of them are small unmanned ground vehicles differentially driven, with the former being smaller and less powerful overall, and the Waffle Pi differentiating from

the standard Waffle due to the presence of a Raspberry Pi and the absence of the Intel RealSense D200. As the basis platform for this work, Turtlebot3 Waffle was chosen. The default hardware included is presented in Figure 3.6. Dynamixel is a high performance and versatile motor actuator designed by ROBOTIS. Open CR (Figure 3.7) is an embedded board equipped with a lot of ports as its microcontroller manages the actuators and sensors of the robot. The board itself has a 3 axes accelerometer, gyroscope and magnetometer sensor.

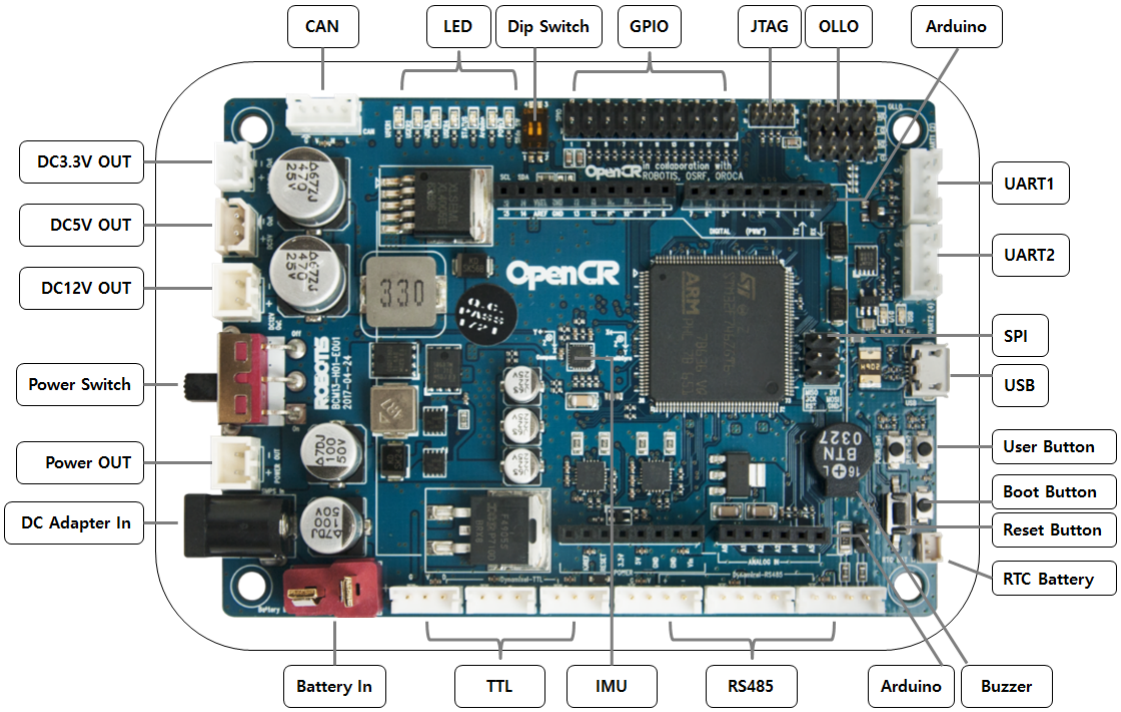


Figure 3.7: OpenCR components [30]

The Intel Joule Single Board Computer has been removed, and its place was taken by an Acer Swift 3 with 8 GB of RAM and a Ryzen 5 4500U processor, running Ubuntu 20 and ROS2 Foxy. Although more bulky, it has the benefit of providing, alongside computational power, a graphical interface. Furthermore, the Coral USB Accelerator[9] was added to the system, an Edge TPU coprocessor specifically designed to perform high speed machine learning inference.

3.3.2 Intel RealSense D435i

The Intel RealSense coming by default is a R200, but it has been substituted with a more advanced D435i. It is composed by a 30 fps RGB camera with 69°x42° field of view and 2 MP resolution, and a stereoscopic depth camera 87°x58° and global

Figure 3.8: Coral USB accelerator [9]

shutter that limits blurring in the images acquired. Its ideal range goes from 30 cm to 3 meters, with a measurement uncertainty below 2% at 2 meters.

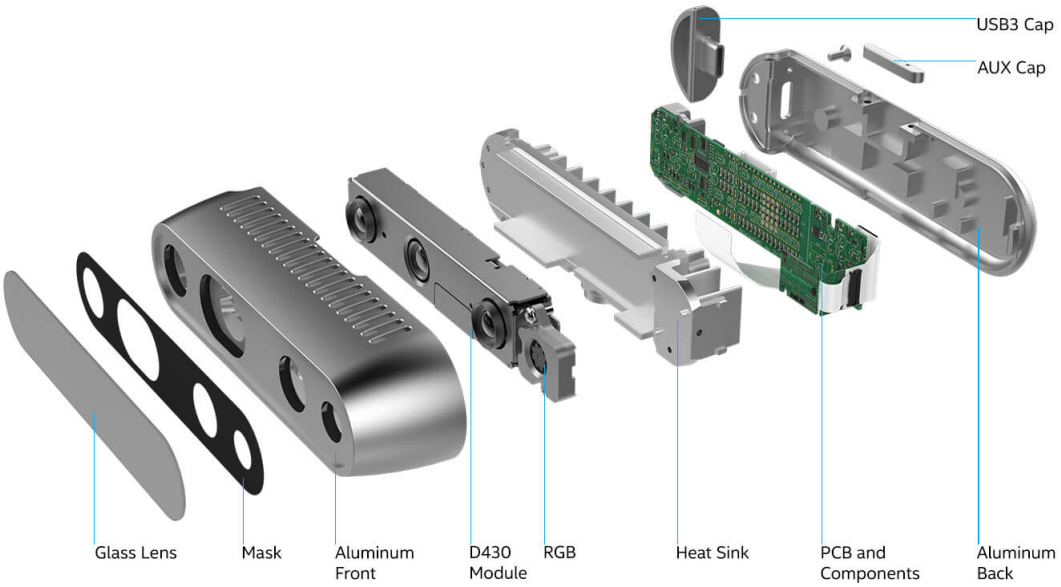


Figure 3.9: Intel RealSense D435i components [20]

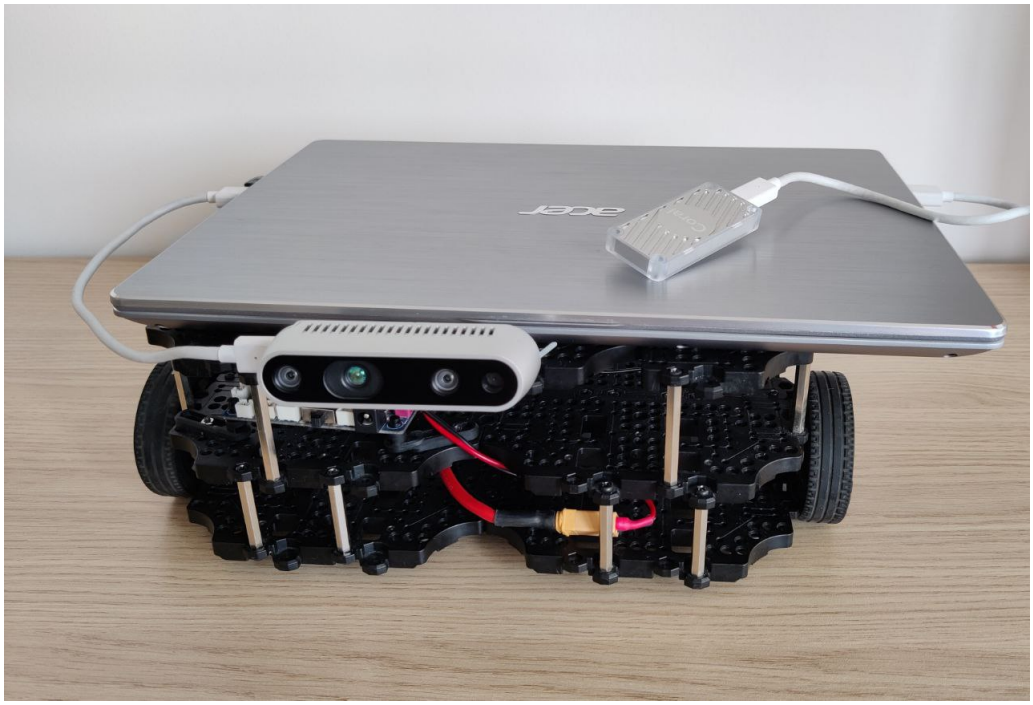


Figure 3.10: Final version of the robot

Chapter 4

Person following

The first objective of the thesis was the development of a ROS node able to recognize if inside the field of view of the robot a person is present, and then get the relative distance. In this way the robot can either follow the person or check that they are following it, if the distance is not increasing. Therefore, the first step was to make the robot detect the figure of a person in the camera stream. To do so, it was used an already well tested convolutional neural network, PoseNet[32].

4.1 PoseNet

PoseNet is a light but reliable framework for pose estimation and instance segmentation. In particular, in this work, the first result has been exploited. Given an input image, at first the CNN detects individual keypoints of the human body, then tries to group them starting from the more confident. This is called bottom-up approach, while on the other hand in the top-down approach at first there is an object detection phase, where a box is drawn containing the figure identified as a person, then during a second step individual keypoints are searched inside the box. PoseNet, by default, is trained on a set of 17 keypoints: nose, eyes, ears, shoulders, elbows, wrists, hips, knees and ankles.

As it is a Tensorflow Lite model, PoseNet can easily be compiled and run on the Coral USB Accelerator.

4.2 SORT

Simple Online Realtime Tracking[6] is an algorithm developed by Alex Bewley to track the movements of a detected object in a camera stream. The necessity to integrate it in this work comes from the fact that PoseNet alone is not able to recognize the presence of the same person in different frames, and thus if more than one person are identified, different IDs will be assigned to the resulting poses

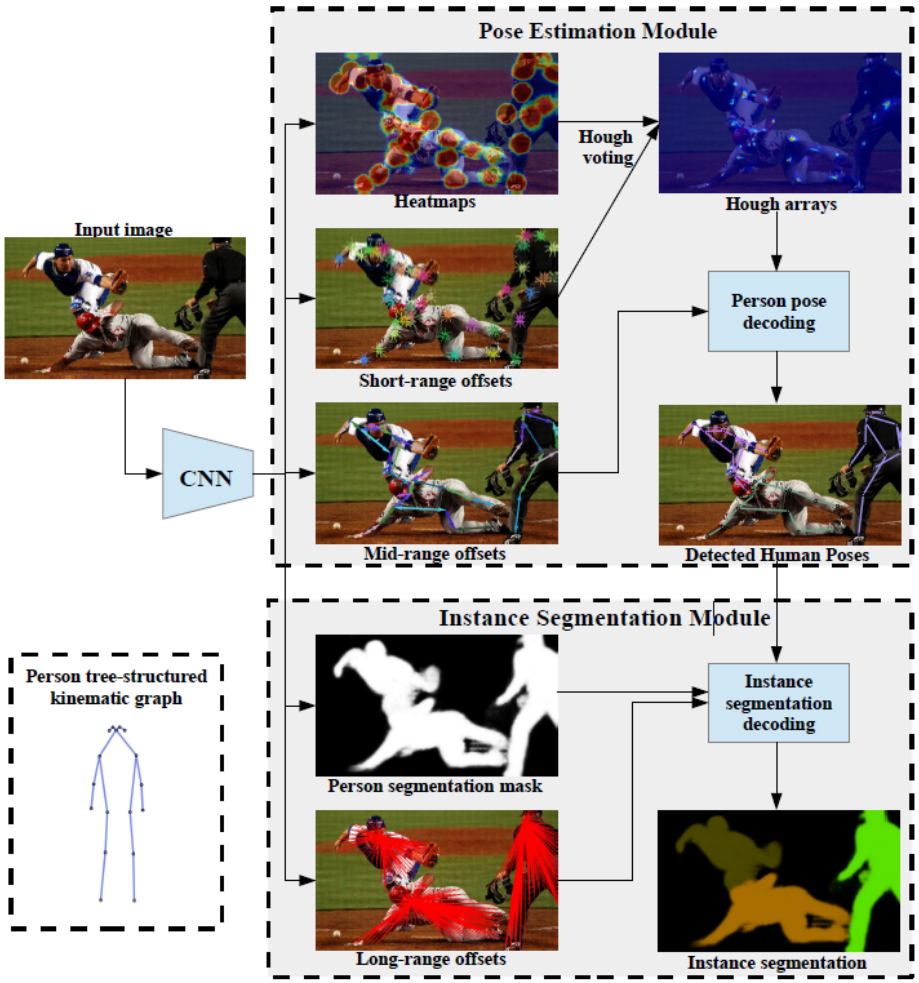


Figure 4.1: PoseNet framework [32]

estimated, frequently interchanging among consecutive frames. To run properly, SORT needs to be fed with the bounding box of the object. In the case of study, it was decided to take a box containing the two keypoints relative to the shoulders of the person. Indeed, these are two of the most recognizable pose features, and they are present nearly every time a person is identified, as the upper body gives much more confidence to the algorithm than the lower body. Therefore, as SORT requests an input vector in the form [left, bottom, right, top], the vector passed was chosen as

$$[x_{min} - 20, \quad y_{min} - 20, \quad x_{max} + 20, \quad y_{max} + 20] \tag{4.1}$$

where each element is taken computing coherently the maximum and minimum between the two shoulder keypoint.

4.3 Acquiring the distance

To get the relative distance between the person and the robot, the camera pinhole model has been exploited. As a calibrated camera is being used, the calibration matrix K, which is fundamental in this work, is known:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

where $f_x = 617.42$ and $f_y = 617.79$ are the focal lengths of the Intel RealSense D435i depth camera along respectively the x and y axis, while $c_x = 316.72$ and $c_y = 244.22$ are the coordinates in pixels of the camera center in the frame. Firstly, the middle point between the shoulders is found as:

$$\begin{cases} x_{mp} = \dfrac{x_{right-sh} + x_{left-sh}}{2} \\ \\ y_{mp} = \dfrac{y_{right-sh} + y_{left-sh}}{2} \end{cases} \tag{4.3}$$

Now that these coordinates in the RGB camera frame are known, it is possible to obtain the distance between the camera plane and the plane parallel to the camera containing the middle point. This information, measured by the depth camera, is conveniently stored inside a matrix corresponding to the depth camera frame aligned with the RGB camera, so that it is sufficient to access the element in the same row and column as the pixel in the RGB camera.

With reference to Figure 4.2 this is the distance along the z axis. To get the ones along x and y, at first they are easily calculated in pixels,

$$\begin{cases} x_{px} = x_{mp} - c_x \\ y_{px} = y_{mp} - c_y \end{cases} \tag{4.4}$$

Then we have basically two pairs of similar triangles, as shown in Figure 4.3, so that

$$\begin{cases} X_m = \dfrac{x_{px} \cdot Z_m}{f_x} \\ Y_m = \dfrac{y_{px} \cdot Z_m}{f_y} \end{cases} \tag{4.5}$$

4.4 ROS node implementation

All the concepts mentioned above were merged inside a ROS node which can run in the background while the robot is performing some other tasks. It subscribes to two

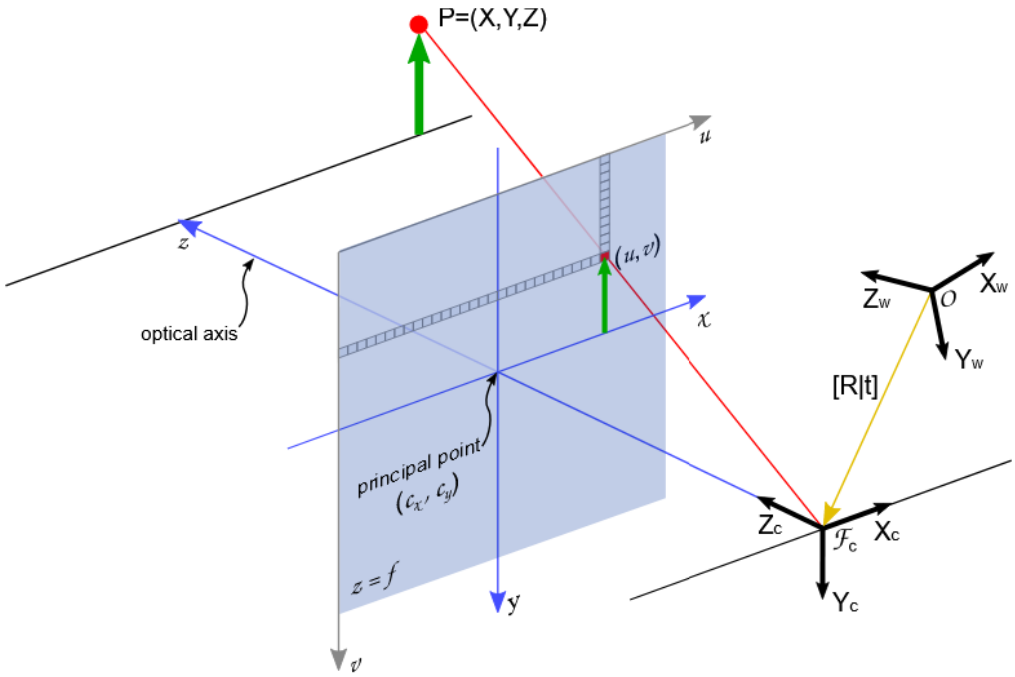


Figure 4.2: Pinhole model for a camera [33]

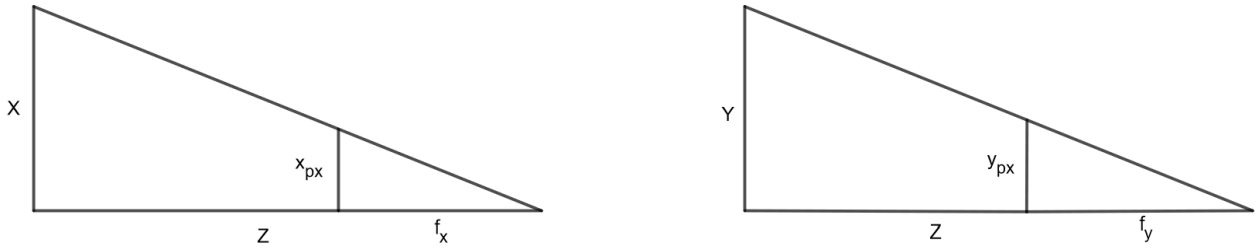


Figure 4.3: Similar triangles putting in relation focal length, distances in pixels and real world distances.

topics: */camera/color/image_raw* and */camera/aligned_depth_to_color/image_raw*. Every time a new RGB frame is published by the Intel RealSense, it runs the inference of PoseNet. If a person is detected, the keypoints are drawn on the frame and linked to build the stylized skeleton of the figure. Then the SORT algorithm is called, as explained in Section 4.2. When computing the person distance, the displacement along the y axis can be ignored: as the mobile platform is constrained to the ground, that displacement cannot be compensated. Instead, the distance along the x and z axes are stored, along with the person ID.

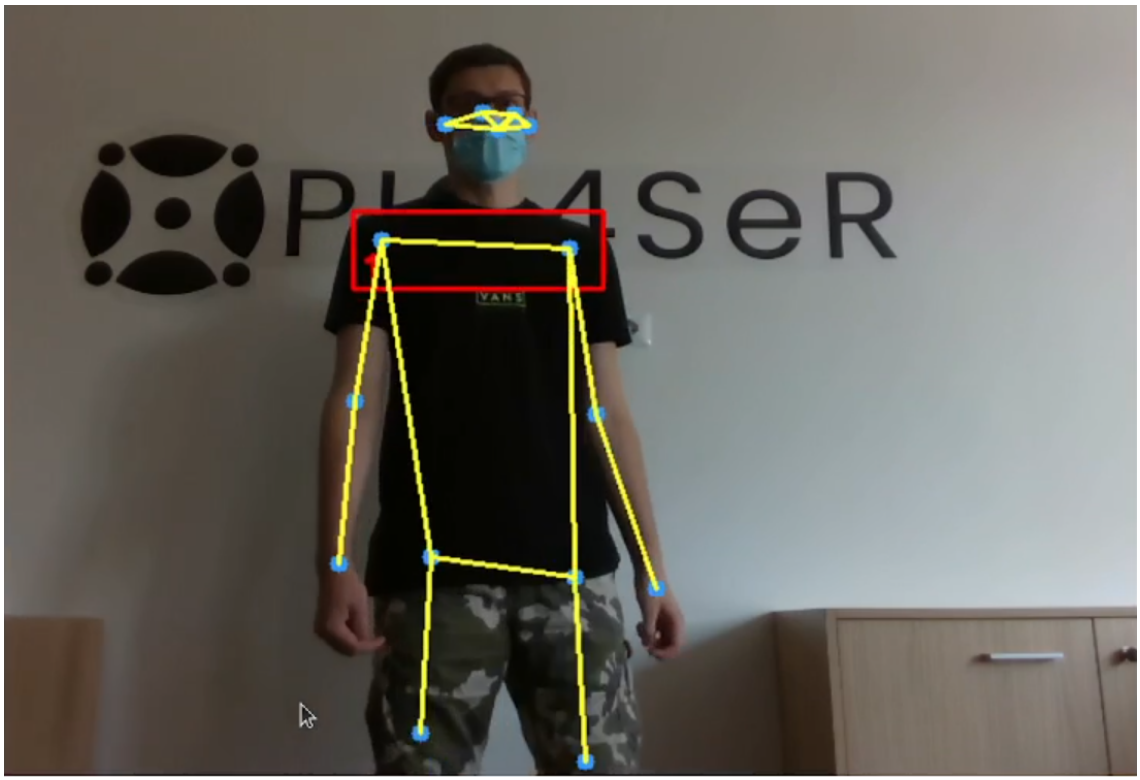


Figure 4.4: Pose recognition

4.4.1 ROS service integration

Alongside the ROS node previously presented, a ROS service was realized. It reads the robot position in the */odom* topic, and acquires the goal location from the PoseNet node, combining them in a unique message consisting of a new data type. In this way the SLAM algorithm can easily access the starting and the goal point of the robot.

Chapter 5

Relocalization with tags

Relocalization is a fundamental task for a mobile platform. As it navigates in the environment, the pose estimated by the robot itself increasingly differs from the actual one, if it is left alone with no feedback. This happens due to errors in the IMU as explained in Section 3.1.1, but also with encoders in presence of not smooth ground, as the wheels spin making no contact. It has a significant outcome even if it happens for a fraction of a second, as in the case of floor grout lines.

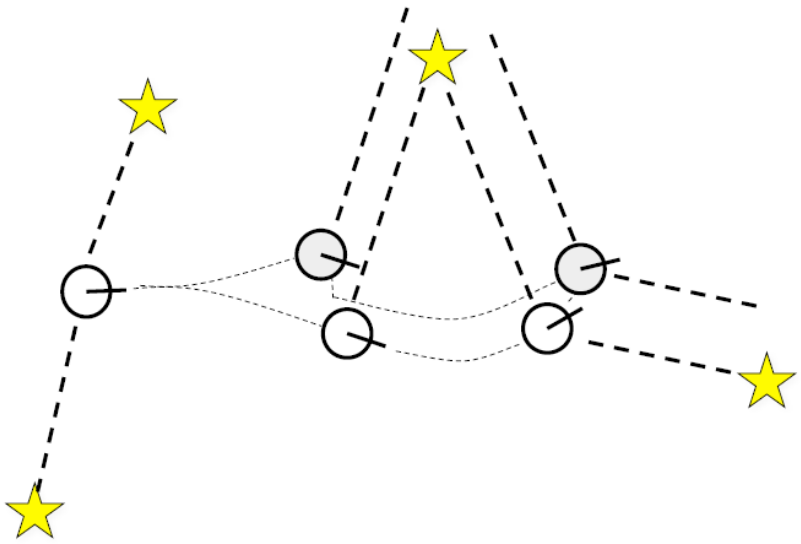


Figure 5.1: Correction of the odometry error via landmarks

A classical way to perform relocalization is via fiducial markers, so that knowing the a priori position and orientation of the tag in the global map, computing the relative pose of the robot from it, the robot global pose can be calculated.

Differently from other barcode systems like QR codes, visual fiducials are designed to be artificial landmarks easily recognizable. To accomplish that, a compromise must be accepted. They contain fewer but bigger data cells, such that a tag can be detected and decrypted even if tucked away in the corner of the image, badly lit, or seen laterally. That's not the case of QR codes for example, which require an high resolution image and must be well framed to be recognized. In turn, they can contain much more data.

Here, with respect to the person recognition problem previously presented, a further step is needed. Even if the tag is detected and its center is found, like it was for the middle point between shoulders in the PoseNet node, there is not enough information to estimate the robot pose. Indeed, knowing the orientation of the tag is fundamental. With reference to Figure 5.2, Z and X can be computed as already explained thanks to the measurements of a depth camera, but without knowing the angular displacement ϕ between the tag plane and the camera plane, X_W and Y_W cannot be calculated. Luckily, the classes of tags developed by researchers include

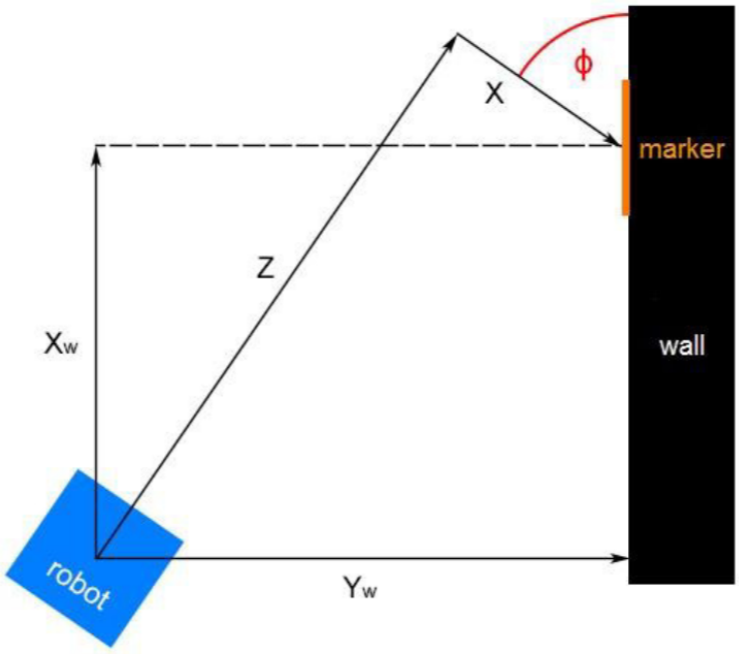


Figure 5.2: Relocalization via tag [4]

in most cases built-in functions that provide the relative orientation of the marker. Among the list, April tags[29] were picked as they are easy to integrate inside any project and all the material related is open source.

5.1 April Tags

April tags are sets of markers specifically designed for computer vision purposes. They come in different families, each one more suited for a specific task. For this work, a marker among the ones present in the standard Tag36h11 family was randomly picked.

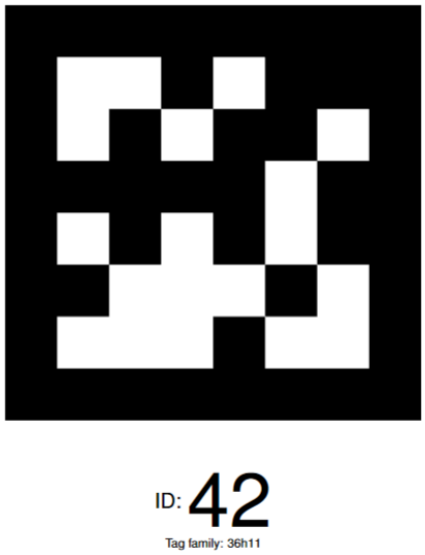


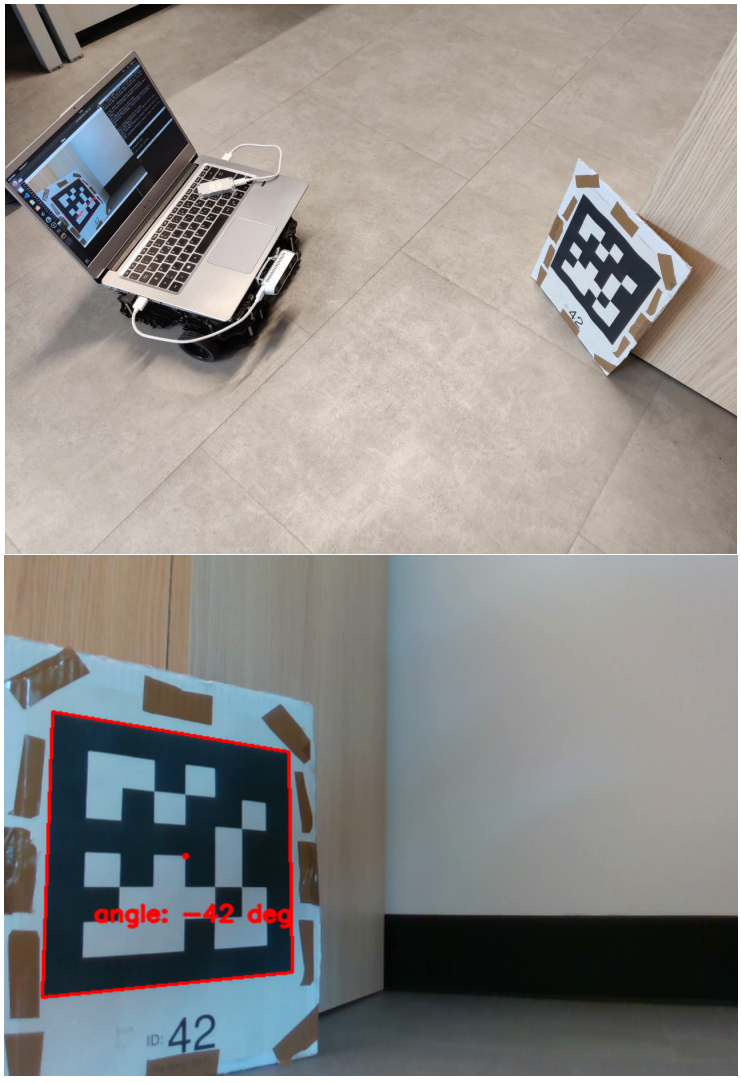
Figure 5.3: The specific April tag used

April tags framework comes with a series of tools, including a Python library containing the detector. What it does is basically to look in the gray-scaled converted image for a four sided area that has a black inside and a white outside, applying image segmentation. Once the tag dimensions in the image are identified, they are compared to its real ones, to be provided a priori, so that the algorithm is able to compute the tag pose in the camera reference frame.

Finally, April tags creators state that below 3 meters the distance is estimated almost perfectly, and then starts to slowly get worse, while the error on the rotation angles is lower than 0.2° till 70°. After that threshold it reaches even 2°, but values over 70° represent extreme cases, in two ways. First, it is difficult for the detector to recognized the tag in these situation, as it is almost parallel to the camera. Second, imagining a real world application, if the tag is put on a flat surface like a wall or a door in the perimeter of a room, it is unlikely that the robot will be in a situation where it sees the marker from these angles.

5.2 Implementation

The first step in the realization of the ROS node was the integration of the detector inside a Python script. This was made easy by the library directly provided by the authors of April tags. Given in input the RGB camera frame, the detector returns the position of the four vertices of the polygon, and the transformation matrix between the camera reference frame and the one of the tag. As a visual little addition, the polygon of the marker is drawn on the image. Two adjustment were



added to the raw detector. Firstly, if the tag is estimated being further than 2.5 meters, it is ignored, as the estimation would be prone to errors. On the contrary if the marker is too close, its figure could be not completely inside the camera frame, but enough to be recognized. This is especially critical when a part of the black

external edge is cut, as the tag would result "squeezed" from the point of view of the algorithm, thus leading to an overestimation of the angle with respect to what it is in reality (Figure 5.4). To prevent this a check is made after the four vertices

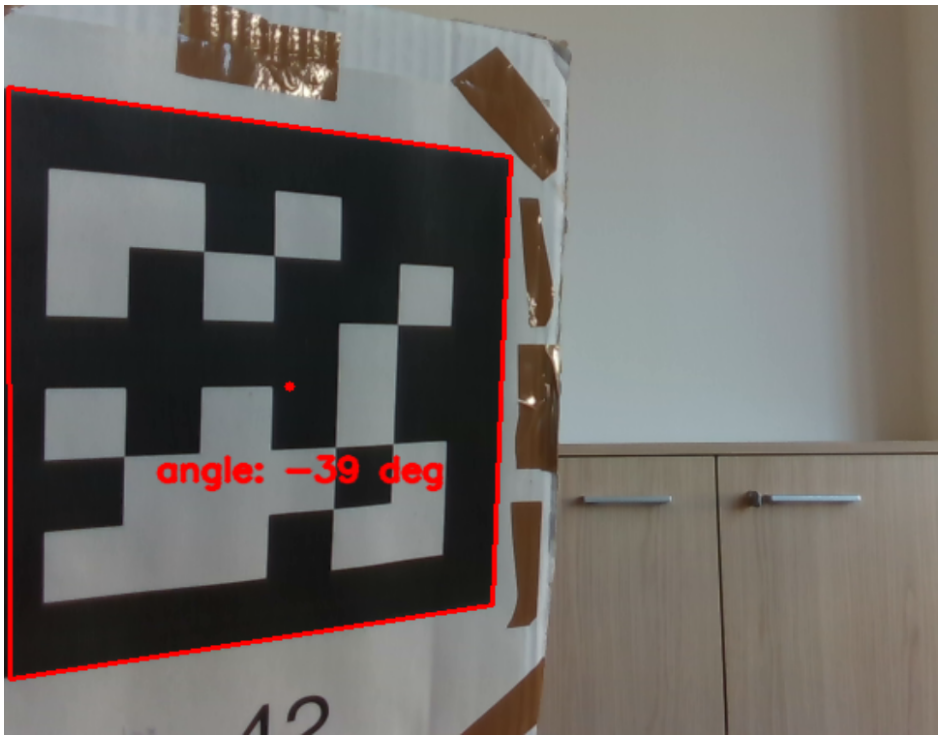


Figure 5.4: Detection of a tag cut at the border

of the polygon are detected: if they are too close to the camera frame edge, the tag is ignored.

5.2.1 Frontal tag

Position estimation

Now, as explained in Appendix A, we can derive from the rotation matrix a set of three angles that describe the orientation of the tag. In this particular application, we are interested in the pitch angle, relative to a rotation around the y axis of the tag. Since it is antiparallel to the robot z axis, that angle is the only one changing while the robot is navigating. Indeed it is holonomically constrained to move only along its x axis and rotate around its z axis. As a result, there cannot be any

vertical displacement and the other two angles of rotation are always null.

$$\begin{aligned} z_{robot} &= fixed \\ \theta_x &= 0° \\ \theta_y &= 0° \end{aligned}$$

An exemplifying situation is presented in Figure 5.5.

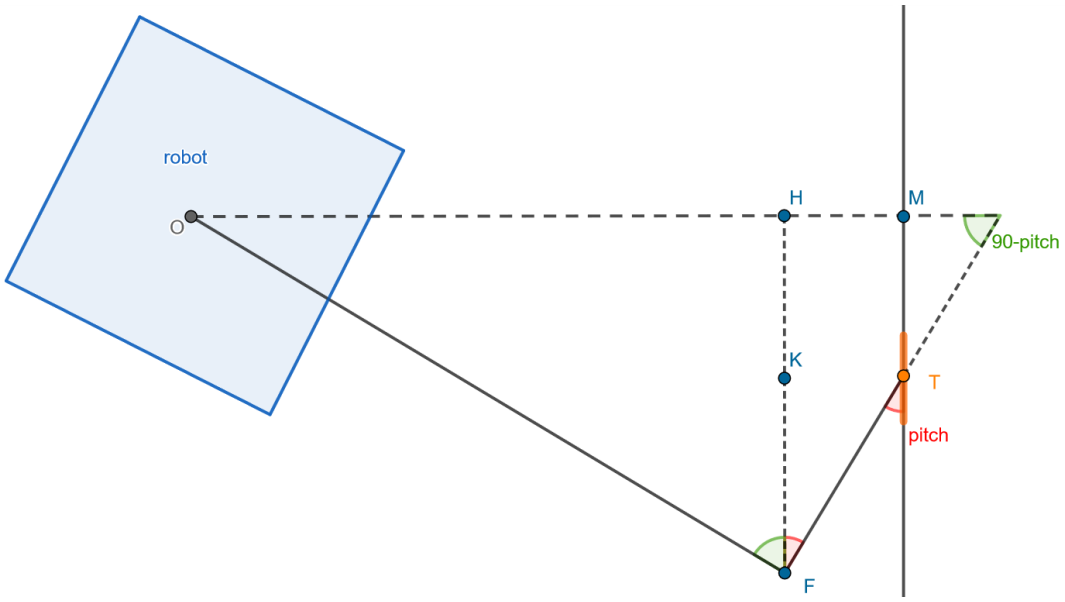


Figure 5.5: Geometrical description of a situation where the tag plane is parallel to one of the axes of the global map

Up to this point $\overline{OF}$ and $\overline{FT}$ distances and the pitch angle are known. The goal is to compute $\overline{OM}$ and $\overline{MT} = \overline{KH}$, that are the distances projected along the x and y axes of the global map. To do so, a bit of trigonometry is involved:

$$\begin{aligned} OH &= OF \cdot \cos p \\ HM &= FT \cdot \sin p \\ FK &= FT \cdot \cos p \\ FH &= OF \cdot \sin p \\ OM &= OH + HM \\ KH &= FH - FK \end{aligned}$$

One important thing to notice is that while $\overline{OM}$ will always be a positive quantity, as the tag cannot be detected from behind, $\overline{KH}$ can be both positive and negative,

depending if the robot is, respectively, to the right or to the left of the center of the marker. It is important to specify that this configuration considers the tag as parallel to one of the axes of the global map. This is a simplification that makes the immediate next step easier, but it is not a stretch, as it represents the situation in which the marker is put on a wall in room, where usually walls are orthogonal each other. At this point the a priori information on the location of the tag comes into play. Since the vector of the marker position and the relative distance between it and the camera are parallel, a simple algebraic sum is needed to estimate the robot global position. However, since the tag can be detected only when in front on the camera, two cases must be distinguished. The first is when the z axis, always exiting from the center of the marker perpendicular to its plane, is parallel to the x axis of the global map (situation 2 in Figure 5.6). In this case, the quantities can be just sum together:

$$x_{robot} = x_{tag} + \overline{OM}$$
$$y_{robot} = y_{tag} + \overline{KH}$$

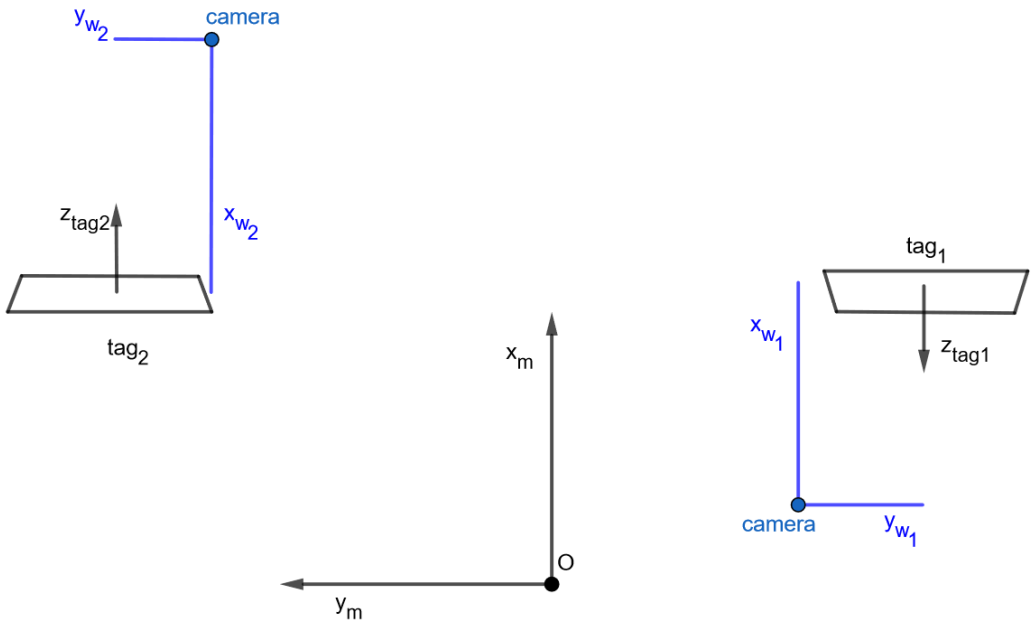


Figure 5.6: The two different situations, where the z axis of the tag can be parallel (2) or antiparallel(1) to the x axis of the global map

In the other configuration (1 in the figure) the z axis of the tag is antiparallel to the x axis of the global map, the quantities must be subtracted:

$$x_{robot} = x_{tag} - \overline{OM}$$
$$y_{robot} = y_{tag} - \overline{KH}$$

In general, using a variable *z_parallel* with value 1 if we are in situation 2 and value -1 if we are in situation 1, it all can be put together as:

$$x_{robot} = x_{tag} + z\_parallel \cdot \overline{OM}$$
$$y_{robot} = y_{tag} + z\_parallel \cdot \overline{KH}$$

Orientation estimation

For what regards orientation, things are a little easier. Knowing the angular displacement between the tag and the robot in the form of the pitch angle, we can just add it with sign to the tag orientation with respect to the global map reference, respectively 0° and 180° in situation 1 and 2 represented in Figure 5.6. Exploiting the *z_parallel* previously defined, the robot orientation can be computed as:

$$\theta_{robot} = pitch + 180 \cdot (1 + \frac{z\_parallel - 1}{2})$$

In this way, in case 2 (z_parallel= 1), 180° will be added to the pitch angle, as the robot must have completed half a turn to have it in front. On the other hand, in case 1 (z_parallel= -1), no angle will be added.

5.2.2 Rotated tag

Previously a significant hypothesis was assumed, that the tag would always have been parallel to one of the axes of the global map. To generalize the work, let the marker plane form any angle with the x global axis. A general situation is pictured in Figure 5.7, where this angle is called α.

Position estimation

In the previous step what here are called x_A and y_A were retrieved. Now, exploiting the a priori knowledge of the angle α, x_W and y_W must be computed. First of all, it is useful to define angle $\beta = 90° - \alpha$. Then with a bit of trigonometry it can be derived that:

$$x_W = x_A \cdot \sin\beta + y_A \cdot \cos\beta$$
$$y_W = -x_A \cdot \cos\beta + y_A \cdot \sin\beta$$

These are the distances between the robot and the tag along the x and y axes of the global map. Now, they can be put together with the a priori information on the position of the tag. As this is the most general framework, in which the case of the parallel tag can be obtained by imposing $\alpha = 0°$, the following equations are the final ones to estimate the robot position:

$$x_{robot} = x_{tag} + z\_parallel \cdot x_W$$
$$y_{robot} = y_{tag} + z\_parallel \cdot y_W$$

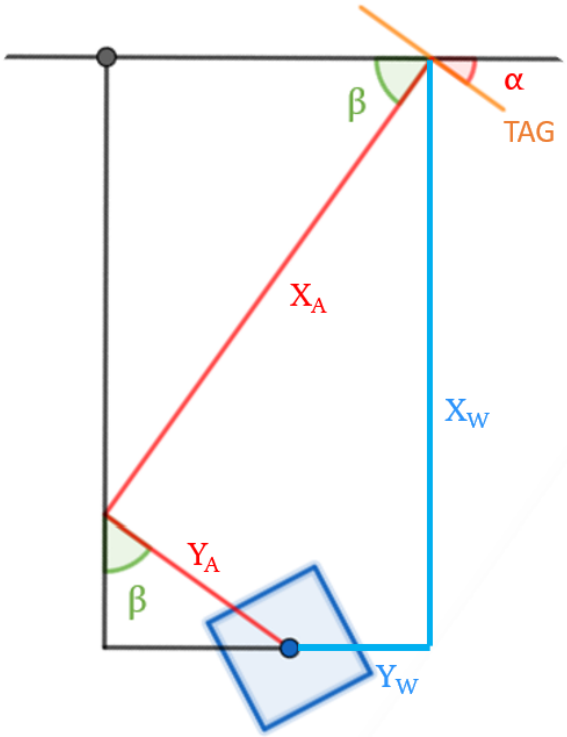


Figure 5.7: Geometrical description of a situation where the tag plane forms an angle α with the x axis of the global map

Orientation estimation

Finally, the estimation of the robot angle of rotation is performed with the same equation, but in this case the angle α must be subtracted from the sum:

$$\theta_{robot} = pitch - \alpha + 180 \cdot (1 + \frac{z\_parallel - 1}{2})$$

To explain why, with reference to Figure 5.7, let's imagine the case in which the camera, and thus the robot, is face to face with the tag. In this case the pitch angle detected would be 0°, and there would be no other contribution outside α, as z_parallel= -1 in this case. Since here $\alpha \approx 30°$, the estimated rotation around the z axis of the robot would be $\approx -30°$, coherently.

5.3 Overwriting Turtlebot3 self estimation

As the manufacturers of Turtblebot3 did not expect that someone would want to overwrite the pose self estimated, it has been necessary to dive in its core C++ files to making it possible. In particular a callback was added that runs when a

pose is published on a new personal topic, modifying the x and y coordinates of the robot, and its orientation. After that, the control is given back to the usual odometry functions of Turtlebot.

5.4 Tests and results

To test the correctness of the algorithm, the tag was put in a room in four different configurations, as pictured in Figure 5.8. The marker was put in places with both positive and negative x and y coordinates, and with positive and negative angles of rotation. As the robot was navigating, it had the marker both on its right and on its left, thus simulating all possible scenarios.

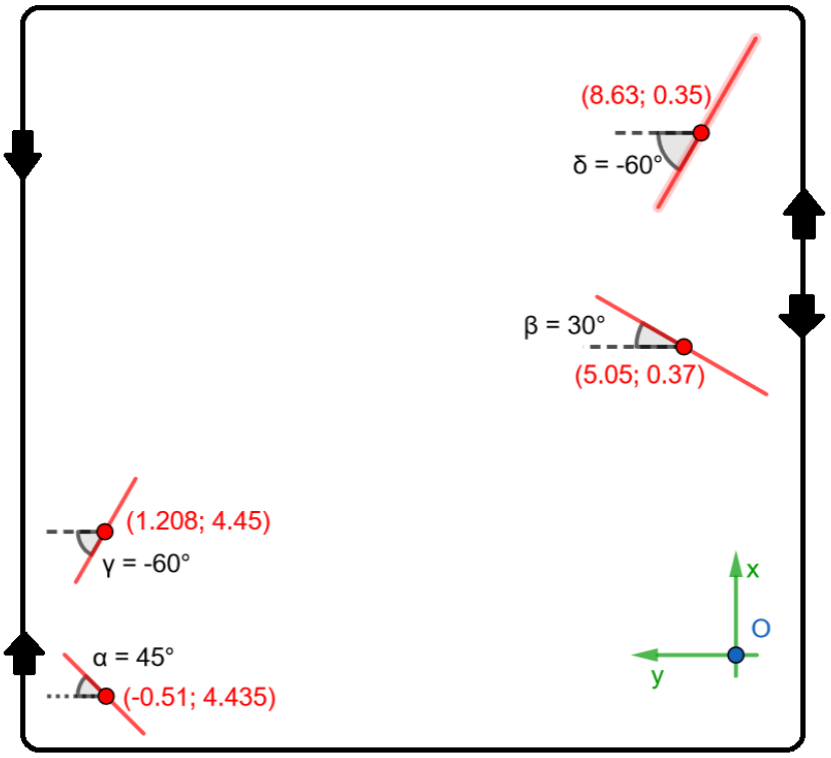


Figure 5.8: Pose of the tag in the four different positions in the test

Comparing the estimated with the real robot pose, it can be concluded that the algorithm output has an error of around 0.8° for the angle of rotation and 1 cm for the x and y coordinates. The resulting estimation is of course very dependant on the preciseness of the a priori info on the tag. A lot of effort should be put in having a value of the angle as accurate as possible, as this is the factor with the biggest impact on the correctness of the estimations.

Chapter 6

Relocalization with feature matching

Once the application with the tag was working very well, it was decided to go on with a more challenging framework. This time relocalization was performed removing the marker, and relying only on images taken by the camera, with nothing particular in them. Indeed, the idea is to extract relevant features from a pair of similar pictures. Comparing the different position in pixel coordinates of couples of features present in both images, the different camera pose can be derived. Three are the most popular feature extractors: SIFT(Scale Invariant Feature Transform)[23], SURF (Speeded-Up Robust Features)[5] and ORB(Oriented FAST and Rotated BRIEF)[36]. While ORB is the fastest, and as such more suitable for application like visual odometry, SIFT is the most reliable one, but requires more computational time. For this work SIFT was chosen, as quickness is not the priority.

6.1 SIFT and RANSAC

SIFT is a method presented by David Lowe for feature extraction and description. Given a gray-scaled image, the algorithm returns a list of vectors containing, for each feature, its location on the image and its description, consisting of the module of the gradient in the eight major directions, computed in a neighbor of the feature. These feature are invariant to affine transformations such as rotation, translation or scaling, and partially to changes in the illumination. As SIFT looks for high contrast regions inside the frame, usually keypoints lie on the edges of objects, as seen in Figure 6.1.

The algorithm has received particular interest for its possible use in the fields of image stitching, robotic navigation and object recognition. Indeed, having two similar images, features can be extracted from each one of them and then the vectors of features can be compared to find matching couples. The way this is performed

Figure 6.1: Feature extraction with SIFT

is by evaluating the ratio between two euclidean distances: the first is the one between the descriptor vectors of one feature and its closest neighbor; the second is the distance with its second closest neighbor. Naturally, the second is bigger than the first one, and thus the ratio is always < 1. The smaller the ratio, the closer the nearest neighbor is to the starting feature than the second nearest neighbor. Lowe suggests a threshold of 0.8 in order to reject 90% of the false matches together with a 5% of good matches. At the end of the process a list of feature couples is derived.

As a final refinement, RANSAC (Random sample consensus)[16] is applied to filter out residual false matches. The algorithm estimates, according to some parameters to be given in input, a mathematical model to describe the input data. It performs a large number of iterations to confirm the output model, and the randomness comes from the fact that it first starts with a randomly chosen subset of data, and later tries to expand it for all data. At the end the input data is divided in two sets: inliers, whose distribution fits a certain model, and outliers, those who does not. In Figure 6.2 the result of SIFT feature extraction and RANSAC outlier detection is presented. The two pictures were taken with an angle shift of 24°, and since many objects were present on the scene, 144 feature couples were detected.

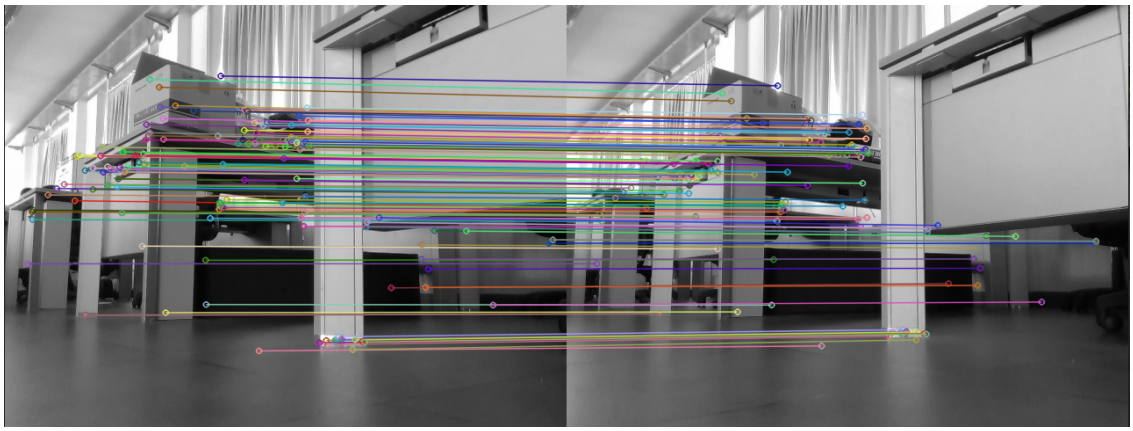


Figure 6.2: Result of feature matching after RANSAC is applied

6.2 Essential matrix

When a calibrated camera is being used, a relation exists between two 2D points in two images, representing the same point in real life. Such relation is called coplanarity constraint:

$$x^T E x' = 0 \tag{6.1}$$

where x and x' are the homogeneous point vectors, and E is a 3x3 matrix called essential matrix. It is a useful tool in estimating the relative pose from which the two images were taken. Indeed, the translation vector t and the rotation matrix R from the second reference frame to the first one can be derived, as the essential matrix can be decomposed as

$$E = S_t R^T \tag{6.2}$$

where S_t is the skew-symmetric matrix built from vector t:

$$t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \rightarrow S_t = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix} \tag{6.3}$$

It is important to notice that the translation vector can be derived up to its direction, and it will be always of unitary magnitude. This is due to the fact that working with 2D images, the information about depth is lost, thus decreasing the degrees of freedom from six to five. Furthermore, there is a mathematical ambiguity generated while performing the singular value decomposition (SVD). It returns four possible configuration, represented in Figure 6.3, respectively characterized by $(R_1, t), (R_1, -t), (R_2, t), (R_2, -t)$. Out of them, only one is physically realizable, placing the point in front of both cameras. To choose which one of them is the correct one, the so called chirality test should be performed, but in the case of

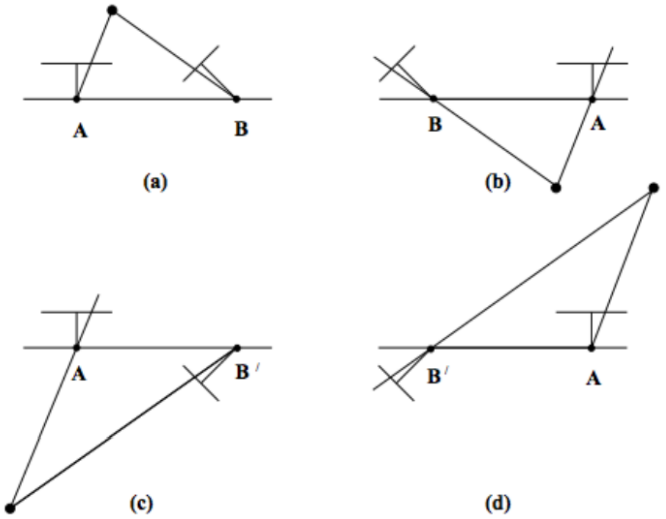


Figure 6.3: The four possibilities coming from the decomposition of the essential matrix[11]

study some assumptions have simplified the process. First, the translation vector was discarded, as it did not provide a complete information. Later it will be discussed how this information was retrieved, using the depth camera. Second, we can exploit once again the fact that the robot can only rotate around its z axis. This means that only one angle of rotation should have a significant value, while the other two should be close to zero. For these reasons, among the two matrices returned as output by the algorithm, the correct one will be close to matrix R_y in Equation A.4, as the z axis of the robot is parallel to the y axis of the image, while the other one will represent a rotation xyz of $[180°, \theta_y, 180°]$. As a result, the correct matrix can be easily identified on the basis of the value of θ_x and θ_z.

6.3 Experiment

After some positive tests with previously acquired images, the feature extraction algorithm was run on the robot. To acquire the database of images, as soon as the robot is turned on, it spins around at $\pi/20$ rad/s, so that in 4 seconds it rotates of $\pi/5$ radiants. In this way it can performs a full rotation (2π radiants) in 10 steps. Every time it stops, the robot saves both the RGB and the depth images in two separate lists. It also performs feature extraction via SIFT at this moment to optimize time. The descriptors are stored in a list as well. After the initial orientation at 0° is regained, the robot is free to roam and perform other tasks. When it comes back to what it thinks is the starting position, it performs again a full rotation to acquire another set of ten images, exactly in the same as for the

database. After that each query image is compared with each database image via their descriptors. The three pairs with more feature matching are saved and for each pair the couple of features with the lowest sum of depth distance is found. This is done because the data coming from the depth camera is more accurate for closer ranges. As the depth distance is extracted, the orthogonal distance is derived too, as shown in Equation 4.5.

To recap, at this point three candidate pairs of images are being considered, and from each one of them the pair of feature closest to the camera are identified. For each pair the distances, both longitudinal and orthogonal, and the pixel coordinates in the respective images are known. The pair of features to continue to work with is chosen by taking the one with the smallest difference between the pixel coordinates of the feature in the database image and in the query image. Once a feature is

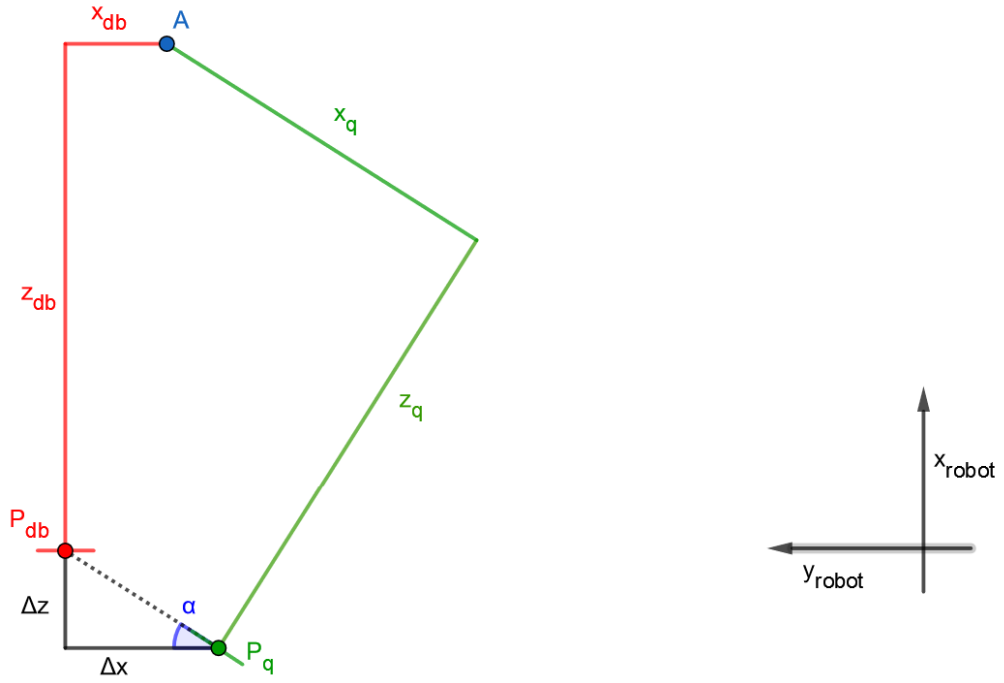


Figure 6.4: Deriving relative displacement between the two camera poses

chosen among the three, the situation is represented in Figure 6.4. Data relative to the database picture is in red, while the one coming from the query image is in green. The angle α is the one derived from the decomposition of the essential matrix. With once again the assistance of trigonometry, the relative displacements Δz and Δx can be computed as

$$\Delta z = z_{db} - (x_q \cdot \sin\alpha + z_q \cdot \cos\alpha)$$
$$\Delta x = x_{db} - (x_q \cdot \cos\alpha - z_q \cdot \sin\alpha)$$

To bring these measurements in the robot reference system of when the database image was shot, it is sufficient to change sign to Δx, as out of the equation the quantity is positive to the right.

$$\Delta x_{rot} = \Delta z$$
$$\Delta y_{rot} = -\Delta x$$

The situation at this point is represented in Figure 6.5. The two quantities are

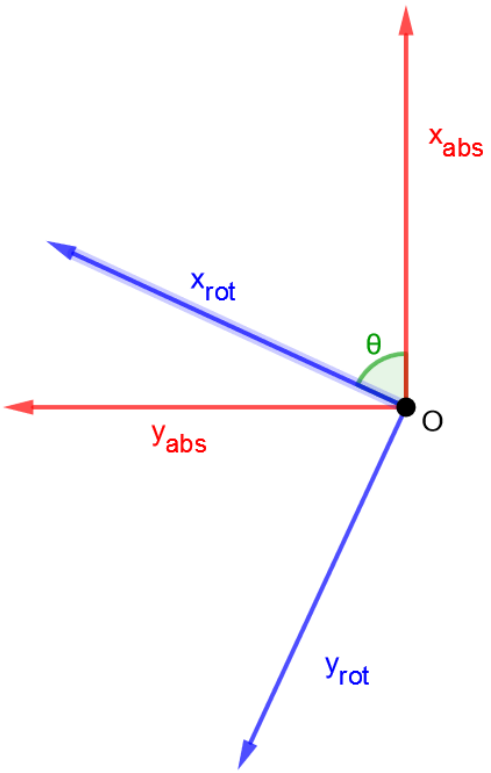


Figure 6.5: The two reference frames

in a reference frame that shares the origin with the global map reference system, but the axes are rotated. This time the angle is solely dependant on which image was chosen among the one in the database, so it can be retrieved easily from the image index. As the first shot, with index 0, is done at 0°, the second (index 1) at 36°, it will be $\theta = 36 \cdot index_{db}$. Thus, the global displacements are related to the previously computed distance by the 2D version of rotation matrix R_z presented in Appendix A.

$$\begin{bmatrix} \Delta x_{final} \\ \Delta y_{final} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \Delta x_{rot} \\ \Delta y_{rot} \end{bmatrix} \tag{6.4}$$

Finally, the estimation of the orientation of the robot is done by adding θ, the angle related to the database image, and α, the estimated angle shift between the two camera poses. This would be the absolute orientation of the robot relative to the query image. To compute the one relative to the moment the robot stopped roaming, it is necessary to subtract the angle related to the query image. To clarify this point, imagine that the query image used was the one shot as third. This means that at the moment it stopped roaming, and after completing the full turn to acquire query images, the robot orientation was $36 \cdot 2$ degrees more than the one it had when it acquired the third query image (the first shot is done at 0°). Thus the final equation is

$$\theta_{final} = \theta + \alpha - 36 \cdot index_{query} \tag{6.5}$$

6.4 Results

A series of tests was performed, in different locations, comparing the robot real position in the map and the one estimated by the algorithm. The results os some of them are shown in Table 6.1. As it can be seen, in general the estimation works quite well. A deterioration in precision with respect to the tag was expected, but it remains sufficient enough. However, in some tests the results were really bad. In these cases the images used in the process contained features at long distances ($>$ 4 meters), and the estimation of the essential matrix in the first place and the final displacements suffered from this inaccuracy. Even though the vicinity of features is a criteria present in the algorithm, those images were selected because they contained elements distant from the robot, that remained more or less all framed together even with a slight change in the angle at which the robot was seeing the scene. On the other hand, elements closer to the robot disappear more easily from the frame as the robot rotates, and this makes the application discard the relative image. In the end, more than a confidence range from the initial position in which the results are satisfying, it is more significant to talk about the environment surrounding the robot while it spins and acquires pictures. Furniture and objects should be closer than four meters, to make the estimation error not explode. This is a likely situation in a domestic environment, that usually does not have so much open space.

Test		Ground truth [cm]	Estimated [cm]
n. 1	x	5	5.6
	y	−10	−17.05
n. 2	x	−7	−2
	y	−9	−7
n. 3	x	−17.5	−17.8
	y	27	23.3
n. 4	x	−10	−10.88
	y	−6.5	−7.2
n. 5	x	−2	45
	y	−11	4
n. 6	x	−36	−43
	y	16	14.18
n. 7	x	−9.5	−24
	y	5	−11
n. 8	x	−16	−18
	y	8	10.7
n. 9	x	−55	−60
	y	20	24
n. 10	x	−133	−145
	y	44	28
n. 11	x	−81	−79
	y	23	34
n. 12	x	−16	−13
	y	260	467
n. 13	x	75	150
	y	150	232

Table 6.1: Tests for relocalization with feature matching

Chapter 7

Conclusions

In the end the development of all the three applications brought good results, to different degrees:

- In the first one multiple people can be identified inside the same image and they are tracked if they remained constantly framed. The errors in the relative distances to the robot, which do not compromise the goodness of the results, depend mainly on the accuracy of the depth estimation. To facilitate the recognition of a person, a support to the camera could be implemented, such that it is located at a higher distance from the ground. From a software point of view, the overall code could be optimized to shorten its runtime, since there is a bit of delay between the moment the image is acquired and when the output is computed. Furthermore, a possible feature to add could be the classification of the person, such that if they exit the frame, they are recognized as the same person when they are seen again.

- Relocalization using the tag is almost flawless. It can be expanded to accept multiple markers to be positioned in different locations in the environment, so that the odometry error is frequently corrected. This framework has the potentiality to work also in 3D, using the other two angles derived from the transformation matrix.

- The relocalization via feature matching is for sure the one that can improve the most. Being more challenging and complex than the one via marker, it was expected to be less accurate, but possible improvements can be found in the way image pairs are selected. At first the couples were classified based on the number of features matching, but it turned out that more couples did not correspond to a more precise estimation. One factor has been identified in the closeness of the feature pair used in the trigonometry calculations, as the distances are more precise in this cases. In the final version, the pair of images is selected by comparing the pixel coordinates of the features extracted. The

one presenting the smaller shift is chosen. Thus in the end there is a trade off to work on: images with close objects should be preferred because depth estimation is more accurate, but as the robot rotates, the pixel coordinates of one feature change greatly. On the other hand, objects far from the camera remain framed for greater variations in the angle of view, and they stay in almost the same spot in the image for small variation. A possible solution could be to consider weights during the choice of image pairs, putting emphasis on the medium distance of the features filtered by RANSAC. However this would increase the computational time, as the RANSAC algorithm would be applied to the whole set of 100 couples, instead of 3.

Appendix A

Transformation matrix

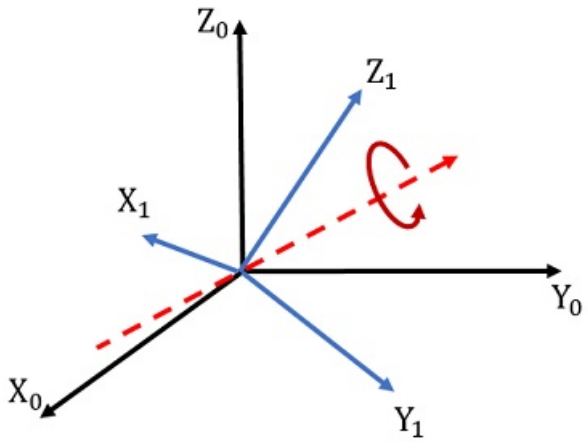


Figure A.1: Two reference systems sharing the origin [1]

Given two reference systems $\Re_0$ and $\Re_1$ as in Figure A.1, their reciprocal relation can be described by two rotation matrices, depending from which reference frame is considered as the starting one. For example, the three unit vectors taken along the main axes of $\Re_1$ can be described using the basis of $\Re_0$ composed by its unit vectors, building the matrix conventionally denoted as R_1^0:

$$R_1^0 = \left(\left[x_1 \right]_0 \left[y_1 \right]_0 \left[z_1 \right]_0 \right) \tag{A.1}$$

The inverse operation is represented by the rotational matrix $R_0^1 = (R_1^0)^{-1} = (R_1^0)^T$ as they are orthonormal matrices. As Figure A.1 suggests, an axis can be found around which a single rotation is needed to make the two reference systems assume the same orientation. But every rotation can also be described as a composition of

three elementary rotations along axes x,y,z:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{A.2}$$

$$R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{A.3}$$

$$R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.4}$$

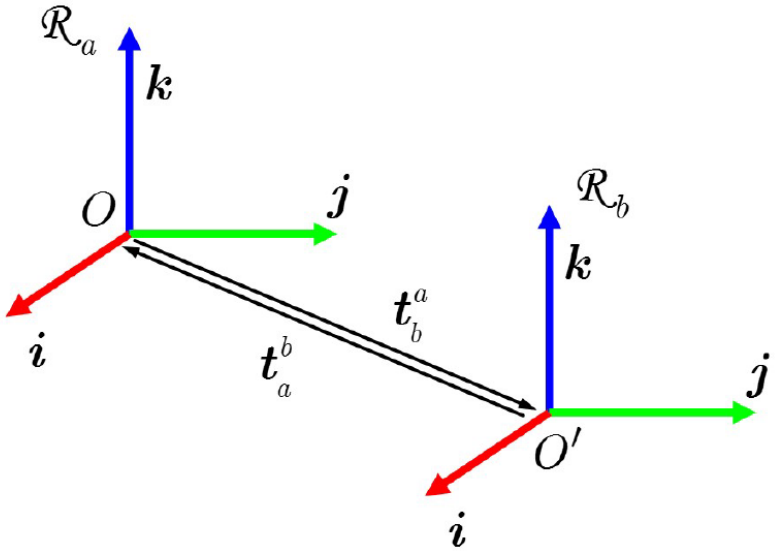


Figure A.2: Two reference systems rigidly displaced

If two reference systems have parallel axes and the origin rigidly displaced, like in Figure A.2, their relation can be expressed by translation vectors. $\overrightarrow{OO'} = t_b^a$ is the translation vector represented in $\Re_a$. Vector $t_a^b = -t_b^a$ is its inverse.

Both rotation matrices and translation vectors are useful to convert the representation of a vector between two reference systems. Let v_a, v_b two vectors representing the same point in two reference frames $\Re_a$ and $\Re_b$, given the rotation matrix R_b^a and the translation vector t_b^a it holds:

$$\widetilde{v_a} = \begin{bmatrix} R_b^a & t_b^a \\ 0^T & 1 \end{bmatrix} \widetilde{v_b} = T_b^a \widetilde{v_b} \tag{A.5}$$

where 0^T is a 1x3 row vector of zeros, and $\widetilde{v_a}$ is the homogeneous vector obtained

adding 1 at the end in an additional row.

$$\widetilde{v_a} = \begin{bmatrix} v_a \\ 1 \end{bmatrix}$$

The transformation matrix T_b^a is a 4x4 matrix that describes a roto-translation relationship at once.

Bibliography

[1] *3D Rotations and Euler angles*. URL: https://www.meccanismocomplesso.org/en/3d-rotations-and-euler-angles-in-python/.

[2] Jayesh Bapu Ahire. *The Artificial Neural Networks Handbook: Part 1*. URL: https://dzone.com/articles/the-artificial-neural-networks-handbook-part-1-1.

[3] Simone Angarano. *Deep Learning Methodologies for UWB Ranging Error Compensation*. 2020.

[4] Andrej Babinec et al. «Visual localization of mobile robot using artificial markers». In: *Procedia Engineering* 96 (2014), pp. 1–9.

[5] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. «Surf: Speeded up robust features». In: *European conference on computer vision*. Springer. 2006, pp. 404–417.

[6] Alex Bewley. *SORT:Simple Online and Realtime Tracking*. URL: https://github.com/abewley/sort.

[7] Anna Boschi. *Person tracking methodologies and algorithms in service robotic applications*. 2019.

[8] *Convolutional Neural Networks — A Beginner's Guide*. URL: https://towardsdatascience.com/convolution-neural-networks-a-beginners-guide-implementing-a-mnist-hand-written-digit-8aa60330d022.

[9] *Coral USB Accelerator*. URL: https://coral.ai/products/accelerator/.

[10] Majid Dadafshar. *Accelerometer and Gyroscopes Sensors: Operation, Sensing, and Applications*. URL: https://www.maximintegrated.com/en/design/technical-documents/app-notes/5/5830.html.

[11] *Decomposition of the essential matrix*. URL: https://answers.opencv.org/answers/38326/revisions/.

[12] Andrea Eirale. *Indoor SLAM and Room Classification with Deep Learning at the edge*. 2020.

[13] *Encoder, angle measurement CCS C Pic16f628 sample application.* URL: https://320volt.com/en/encoder-kullanimi-aci-olcumu-ve-ccs-c-pic16f628-ornek-uygulama/.

[14] *Exchange Data with ROS Publishers and Subscribers.* URL: https://it.mathworks.com/help/ros/ug/exchange-data-with-ros-publishers-and-subscribers.html.

[15] Kaveh Fathian et al. «Quaternion based camera pose estimation from matched feature points». In: *arXiv preprint arXiv:1704.02672* (2017).

[16] Martin A Fischler and Robert C Bolles. «Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography». In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[17] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems.* Sebastopol, CA: O'Reilly Media, 2017. ISBN: 978-1491962299.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* http://www.deeplearningbook.org. MIT Press, 2016.

[19] Michiel Holtkamp and Sjoerd de Jong. «Robot Localisation Using SIFT and Active Monocular Vision». In: (Oct. 2006).

[20] *Intel RealSense D435i camera.* URL: https://www.intelrealsense.com/depth-camera-d435/.

[21] *K nearest Neighbors.* URL: https://laptrinhx.com/k-nearest-neighbors-unlocked-454254569/.

[22] *LIDAR Sensor Scanner to Prevent Obstacles and Navigate Robots.* URL: https://www.amazon.de/-/en/Slamtec-RPLIDAR-Scanning-Obstacles-Navigate/dp/B07VLFGT27.

[23] David G Lowe. «Distinctive image features from scale-invariant keypoints». In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[24] Francois Marais and John Thompson. *Machine learning algorithms in boiler plant root cause analysis.* URL: https://www.ee.co.za/article/application-of-machine-learning-algorithms-in-boiler-plant-root-cause-analysis.html.

[25] Mauro Martini. *Visual based local motion planner with Deep Reinforcement Learning.* 2020.

[26] *Math behind SVM(Support Vector Machine).* URL: https://ankitnitjsr13.medium.com/math-behind-svm-support-vector-machine-864e58977fdb.

[27] Eckart Michaelsen et al. «Estimating the essential matrix: Goodsac versus ransac». In: *Photogrammetric Computer Vision* (2006), pp. 1–6.

[28] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.

[29] Edwin Olson. «AprilTag: A robust and flexible visual fiducial system». In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 3400–3407.

[30] *OpenCR by ROBOTIS*. URL: https://emanual.robotis.com/docs/en/parts/controller/opencr10/.

[31] *OpenCV, a Python library for computer vision*. URL: https://docs.opencv.org/3.4.15/index.html.

[32] George Papandreou et al. «PersonLab: Person Pose Estimation and Instance Segmentation with a Bottom-Up, Part-Based, Geometric Embedding Model». In: *CoRR* abs/1803.08225 (2018). arXiv: 1803.08225. URL: http://arxiv.org/abs/1803.08225.

[33] *Pinhole Camera Model*. URL: https://docs.nvidia.com/vpi/appendix_pinhole_camera.html.

[34] *ROS2 Foxy Intel RealSense package on Github*. URL: https://github.com/IntelRealSense/realsense-ros/tree/foxy.

[35] *ROS2 Foxy Turtlebot3 package on Github*. URL: https://github.com/ROBOTIS-GIT/turtlebot3/tree/foxy-devel.

[36] Ethan Rublee et al. «ORB: An efficient alternative to SIFT or SURF». In: *2011 International conference on computer vision*. Ieee. 2011, pp. 2564–2571.

[37] *Scikit-image, a Python library for image processing*. URL: https://scikit-image.org/.

[38] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. Springer, 2009. ISBN: 978-1-84628-642-1.

[39] *Turtlebot3 overview*. URL: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/.

[40] *Understanding Activation Functions in Neural Networks*. URL: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

[41] John Wang and Edwin Olson. «AprilTag 2: Efficient and robust fiducial detection». In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2016.

[42] Qunjie Zhou et al. «To learn or not to learn: Visual localization from essential matrices». In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 3319–3326.