POLITECNICO DI TORINO

Master degree course in Mechatronic Engineering

Master Degree Thesis

Design of an autonomous system able to follow a moving person

Politecnico di Torino

Supervisors
prof. Marco Vacca

Candidato
Nicola Sestu
matricola: 276140

A.a 2020/2021

Abstract

In this thesis the idea is to create a cart that automatically follows a specific human, this idea can be applied for example inside super markets.
To do this I have tried some techniques and technologies that can be used to build a robot able to automatically follow a human.
The initial goal was to build a subsystem that does not move but that was capable to identify and follow a person using a camera able to move.
The first version of this subsystem included a raspberry pi 2b with a raspberry pi camera module V1, moved by a 2DOF hat controlled by the raspberry, using Tensorflow lite and OpenCV for computer vision and image recognition I obtained performances of 1fps, after upgrading the system with the NCS2 that is an accelerator for neural networks from intel I obtained a speed of 4fps using Openvino.
It was necessary to change the hardware, the second version of the subsystem lead us to use a raspberry pi 4b with the NCS2 and a raspberry pi camera module V2, gaining results around 20fps, so an application that was 20 times more powerful than the first idea.
Since the image recognition algorithm was working I have decided to build the moving platform of the robot.
The moving frame of the robot is a four motor platform controlled by the raspberry pi through a motor driver, connected to a battery pack with a USB output able to deliver 5V-2A with 20100mha capacity.
Since the battery had a USB interface a Buck-Boost converter with the same input was needed in order to regulate the voltage and current drained by the driver of the motors.
The system built until this point was able to follow a moving person but it was difficult to keep track of the target, so I tried to calibrate the code in order to recognize the color of the clothes of the user in order to keep track of it even if other people were in the scene.
This method was impossible to apply because even a small change in the light conditions lead to completely lose track of the user, so I moved to infrared signals to identify the user.
The implementation was done connecting an IR receiver to the raspberry and a powerful IR Led to an arduino uno in order to send the signals, the idea worked

but was discarded because in noisy environment the infrared signals were unusable.
The next solution was based on a bluetooth low energy (BLE) technology, used to triangulate the position in space using the RSSI of the target mobile phone.
During the tests the precision was realistic until the device was too far, but also in this case people or walls could interfere with the tracking system, resulting in wrong measurements.
The final solution used a raspberry pi as the core of the system, combined with the stm32f401re microcontroller and its IKS01A3 shield with mounted on board an accelerometer, a gyroscope and a magnetometer, this allowed us to estimate pose and position of the user and of the robot in space with the aim to keep a fixed relative distance between the two, but the drift due to the double integration of the acceleration was too much in order to have a reliable system.
In conclusion the solution works and it is very promising but to get reliable results more complex sensors fusion technologies like a GPS or sensors to measure the speed of the robot's wheels are needed.

Summary

The aim of the thesis was to design, build and test a robot capable of automatically follow a human, testing different techniques and technologies for the identification and tracking of a specific person.
The system should be applied for example to create smart carts inside super markets.

The initial work was focused on learning and searching in literature the most used techniques, hardware and software for this type of application.
The most common techniques, employs computer vision and image recognition, infrared tracking and neural networks.
The hardware goes from basic and less powerful microcontrollers to more efficient and complex solutions that use powerful hardware with high computational power.

Computer vision and image recognition for the tracking of the user was the first idea that was designed and tested.
At first I tested a raspberry pi 2b with a raspberry pi camera module.
Implementing a neural network using python and Tensorflow lite the performance were not enough for a realtime system, the system was operating at an average of 1fps.
To improve the results I have added an accelerator for neural networks made by Intel, the neural compute stick 2 (NCS2), that uses a custom version of OpenCV named OpenVino.
OpenVino is used in order to communicate with the NCS2, this solution helps the raspberry pi because the inference is done on the external dedicated hardware.
Optimizing the code I was able to obtain around 4fps, that were not suitable for the project.
The system needed to be upgraded to gain better performances, a raspberry pi 4b with the NCS2 and a raspberry pi camera module V2 was than used, improving the average speed to 20fps.

The second step of the thesis involved the building of the robot capable to move and follow a moving person.

The idea is pretty simple, a four motor platform controlled by the raspberry pi through a motor driver.
The platform is built in such a way that it allows to easily place a battery used as a generator and power supply for both the raspberry and the drivers.
A Buck-Boost converter with a USB input was used in order to regulate the voltage and current drained by the driver of the motors, the USB interface was necessary because the battery used was a simple battery pack for smartphones and tablet with a 20100mhA capacity and an output of 5V-2A.
At first there were some problems related to the draining of the current when the motors were controlled, due to the high impulse current absorption.
The issue was fixed by simply applying a PWM to control them, resulting in a slightly slower system but more robust.

The system built until this point was able to follow a moving person but it was difficult to keep track of the target, the techniques used in this case were complex algorithm that required a lot of computational power so a new simpler solution was developed, which included other devices and techniques rather than the computer vision and image recognition alone.
This choice was made because the system wasn't able to keep track of a specific user in complex environment where other humans were present, so the system was calibrated to recognize the color of the clothes of the user, in order to keep track of it even if other people were in the scene.
This method was impossible to apply because even a small change in the light conditions lead to completely lose track of the user.
The next solution was to use infrared signals to identify the user.
The implementation was done connecting an IR receiver to the raspberry and a powerful IR Led to an arduino uno in order to send test signals.
The concept was based on the idea of a wearable device that was capable to send signals to the robot in order to easily identify the user even if the system wasn't able to always keep it into its field of vision.
At first the idea worked but was discarded because in noisy environment the infrared signals were not reliable, also in a scene full of other people or walls the signals to noise ratio was too low.

The third solution tested was based in the use of the smartphone of the user as identifier for the tracking algorithm, in order to lock and track the user also in difficult scenarios.
This solution was based on the bluetooth low energy (BLE) technology.
The idea behind this solution was to triangulate the position in space of the smartphone of the user, using three BLE devices able to measure the Receive Signal Strenght Indicator (RSSI) of the target mobile phone.
The hardware used to do this were 3 stm32f401re microcontrollers equipped with

a BlueNRG shield used for bluetooth low energy applications.
The microcontrollers were equally placed on a circumference, resulting in a equilateral triangle on the plane, this arrangement was perfect in order to gain the maximum precision from the system, but in the test phase also this solution resulted not reliable enough.
During the tests in fact the precision was realistic until the device was too far, around 20cm (maximum) from the circumference the measurements where good enough to have the position in space, but also in this case people or walls could interfere with the tracking system, resulting in inaccurate measurements.

Since the above ideas didn't work a final solution have been developed using inertial measurement units (IMU's) to track the user and the robot in space, calculating the distance between them and using it to keep the robot always near the person.
Also in this case a raspberry pi was used as the main computational power for the system, combined with 2 stm32f401re microcontrollers and their IKS01A3 shield that mounted on board an accelerometer, a gyroscope and a magnetometer.
This technology was applied to our project in order to estimate pose and position of the user and of the robot in space with the aim to keep a fixed relative distance between the two.
During the implementation phase the results were not precise and the drift due to the double integration of the acceleration was too much in order to have a reliable system.

The results obtained however demonstrates that this solution works, and it allows to create a robot that is able to follow a specific person in every conditions, if additional sensors and sensor fusion techniques are used in combination with the IMU.
This solutions will be implemented and tested as future work.

Contents

List of Tables

List of Figures

Chapter 1

State of art

The human following robotics is one of the most challenging task in the robotics field, and one of the most important.

This type of application can be very useful in a lot of different scenarios, for example in a warehouse a following robot could be used by an employee to move heavy loads while walking through the facility, or in a shop scenario a human following robot could be used by the customers to carry their shopping while moving completely hands free.

Nowadays the field of interactive robots is becoming more and more effective, this thesis was inspired by some of the most relevant projects that are present on the IEEE portal, most of them based on computer vision and image recognition.

In [1] the hardware used for the application is a combination of a **Jetson TX1** with a **ZED stereo camera**, with this equipment the solution that is presented in the paper uses **Human Detection Algorithm** with **Histogram of Oriented Gradient** features (HOG) by a **Support Vector Machine** (SVM) in combination with a **predictor Kalman filter**.
The target is found using a model pretrained to recognize a specific person, then the extraction of the background and the identification of the colors simplify the tracking, but since the execution time is too high in case this procedure is repeated for every frame, a prediction algorithm is applied in order to reduce the computations and effort of the whole system, achieving a smaller execution time but maintaining also a good precision and accuracy.

For [2] a **PID controller** has been designed, in order to achieve good performance even with low computational power, in fact due to this constraint the paper deals with only a 2D application.

The method described inside the paper shows how they have managed to preprocess every frame, dealing with extreme light conditions or environment change it is still possible to enhance the interested **feature points**.
Like the paper above [1] the background is subtracted in order to follow the back of the head of the user, with a pretrained model that allows to track and follow the designed human operator.
The tracking is made through the mathematical procedure briefly described in the study and thanks to this even with just a PID controller the whole system is capable to maintain a stable 25fps in a dynamic environment.

Paper [3] uses a completely different approach, since there is no computer vision and no image recognition.
In fact the important components used in this experiment are a **Nintendo Wii camera**, a **ATMega 128 processor**, and four powerful **IR-Leds**.
The concept behind this paper is that in poor light conditions a mobile robot can track a human operator knowing it's relative position thanks to the IR technology, in fact as we can see in the study, the procedure shows how through the **Wii camera**, that is used like a wearable device, the robot can calculate its relative position using the data transmitted from the camera attached to the person, through wireless connection and follow the operator.
In the paper are also shown all the calculus and the mathematical procedure in order to estimate the relative position, and also is possible to notice that the accuracy is about 1.5cm, which means that the whole system could track in a precise way the designed target.
It is relevant to say that the drawbacks are the need for a wireless connection and also the camera as a wearable device.

Research [4] is based on a **Raspberry Pi 2B** used as the processing unit, and an **Arduino UNO** as the control unit, used to control the electrical and electronic components such as motors and sensors.
The objective of this paper is to develop a lightweight algorithm capable of tracking a person using computer vision and image recognition, with the help of some other sensors such as a magnetometer, an encoder and a ultrasonic sensor for the distance.
The algorithm described is based on a **color tag** that is placed on the human operator, this approach permits to clearly identify the target without using too much computational effort but leading anyway to a really slow system, the performance reported on the paper says that the raspberry is capable of processing three frames per second, that as shown in the research is enough supposing that the operator is a slow moving object, but in a dynamic environment scenario this technique is not suitable.

In [5] we have a similar approach to the previous paper but with a really higher computational power, it is also mentioned that the human following feature is only one of the whole project, that basically is a robot designed in order to help in the automotive manufacturing, so it is necessary to carry heavy loads and manipulate them, in fact the robot can handle loads up to 20kg.
The peculiarities of this robot is that the vision based algorithm is somehow personalized for every human operator, in fact it is written that at start up are needed 60 seconds in order to take photos of the target and then elaborate a desired image that will help the robot in the identification and tracking of the desired person.
In the paper this is done using a special image on the operator, it is like a tag that helps the robot in its operations, leading to a more efficient tracking.
As we can see it is more complicated and powerful than the approach of [4], as said in the research the tracking is based on a **non-vector space controller**, but the critical drawback is that this algorithm is really heavy and can run only on powerful hardware, plus the need for a wearable tag.

The [6] approach is one of the most accurate, in fact the project is based on a **adaptive control of a non-linear system**, which means that the system is studied and built for this specific purpose, leading to a perfect tracking and following of the desired target, without issues even in dynamic environments with a lot of people or light changing conditions.

The solution proposed in [7] is based on the analysis of the walking pattern of the target.
The core of the robot is the **SICK LMS200** as a **LRF** (laser range finder), which permits to measure precisely the motion, distance and angles covered by the human operator while walking, store the data and remember them in order to track the walking motion of the desired person even in a crowded environment.
Of course this is a very interesting and accurate solution, but the whole system is very expensive and requires a very high computational power.

As we can see in [1, 2] the whole system is designed to track the person itself with specific algorithms that helps to delete the background isolating the user in order to obtain the desired performances, this techniques requires high computational power or ad hoc tuning in the case of the PID design.

In [3] the approach is pretty different, indeed the system is based in a wearable device that the user should use in order to be recognized and tracked by the robot.

The approach that we see in [4, 5] is easier from the computational point of view, in fact the tracking of the user is done through the help of a tag that is placed on the human, this allows the whole system to neglect other people on the scene, resulting in a lighter algorithm that requires less power and can run also on a raspberry pi, or in the case of the second one, it makes easier the work of the robot that even with powerful hardware has the possibility to spend computational power on other relevant and specific features.

The [6, 7] projects are more complex, in fact prediction algorithm are used in order to distinguish the user from other humans and maintain the tracking of the same person even in chaotic scenarios with obstacles and other people in the scene.

As stated above, the techniques are pretty different but most of them use computer vision and image recognition as the core of the whole system, obstacle avoidance is extremely important, and the tracking of the user is different in every project, we have more complex and powerful solution as well as more simple and light, but all of them are reliable and has obtained good performances.

Chapter 2

Introduction

The aim of this thesis is to design from scratch, build and test a human following robot, using computer vision and image recognition for the tracking of the user, with a system that allows also the robot to identify a specific human.

Also all the assembly and motors control algorithm are needed for this project, with the possibility to add features for the obstacle avoidance.

In the following pages the steps for the accomplishment of the main task will be analyzed, dividing it in smaller and easier minor tasks completed in chronological order.

It was decided to base the system on a raspberry pi microcomputer, but during the test phases it was noticed that it doesn't have the necessary computational power for the image recognition, so it was decided to add an accelerator for neural networks.

A raspberry pi cam v2 was used for the computer vision and a 2dof hat for its movement control and tracking of the user. The raspberry was placed on top of a support that has 4 wheels, each driven by a dc motor, attached to a driver placed between motors, power supply and the pi.

The techniques implemented in order to track the human operator were pretty different.
The first two that were tested relies on computer vision and image recognition, one was a technique for the tracking of the user based on a color tracking algorithm and the other one was a tag tracking algorithm.
Another implementation tested for the user identification was an infrared tracking algorithm.

As it is possible to notice this techniques are the same reported in the papers from the state of art chapter.
Moving to a different direction it was needed to implement and test techniques that have a completely different approach.
One of the new techniques is the triangulation and tracking of the user through the BLE (Bluetooth Low Energy) technology.
The last implementation regarding the tracking algorithm is done using IMUs (Inertial Measurement Unit).

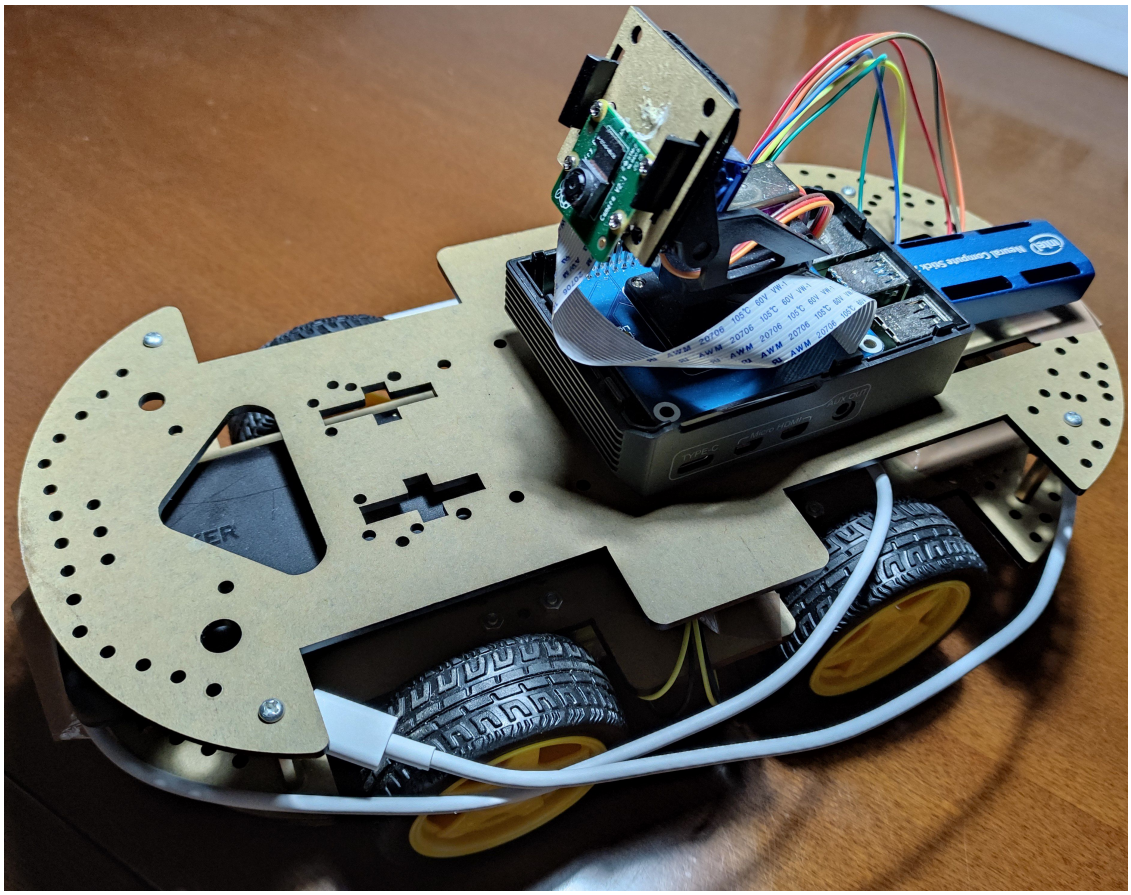

Figure 2.1: Assembled system for only computer vision and image recognition

Chapter 3

Raspberry setup with Openvino

The first idea was to use a **raspberry pi 2b** with a **pi cam v1**, and the most suitable tool for this system was TensorFlow Lite [8].
TensorFlow Lite is a lighter and less powerful tool than TensorFlow, but it is designed for mobile devices that usually do not have the same computational power as the computers, a few example are smartphones, embedded systems or IoT devices. The key features of TensorFlow Lite are the following:

- **Small latency**: since there is no communication with a server.
- **Privacy**: no data leaves the device in which TensorFlow Lite is running.
- **Connectivity**: the connection with internet is not required.
- **Size**: the models are a lot smaller than TensorFlow.
- **Power Consumption**: since no internet connection is needed and the models are lighter, the inference is more efficient resulting in a power consumption optimized for mobile.

After setting up the tools from a fresh Raspbian installation the obtained were of at most **1fps**, that for obvious reasons were not satisfactory for the application, so it was decided to stay on the raspberry pi system but with the upgraded version, using a **raspberry pi 4** with a **pi cam v2**.

The **raspberry pi 4** also has a limited computational power so in order to get better performance it was added the **Neural Compute Stick 2** (from now on it will be called **NCS2** [9]) from Intel.

NCS2 is a USB stick that offers an easy access to neural network functionality, it is equipped with **Intel Movidius Myriad X Vision Processing Unit (VPU)** which has a dedicated hardware accelerator for **Deep Neural Network (DNN)** interference.

Figure 3.1: Neural compute stick 2

The use of **NCS2** requires a custom tool provided by Intel, called **Openvino** [10] in order to interface the raspberry with the accelerator to obtain better performances in the inference algorithm execution, since all the prediction and detection is left to it.
A complete guide for the setup of the tool and its dependencies is left in the **section A.1**.

Openvino is a toolkit that grants access to fast application development and offers a lot of different solutions to a lot of different tasks.
Inside **Openvino** there is a **OpenCV community version** compiled for the intel hardware, also **Openvino** is open source, its source code is available and can be downloaded from github, it means that for everyone that is interested it can be customized and built from source.

A basic knowledge of **OpenCV**[11] is needed in order to understand the toolkit provided by Intel.
OpenCV is a powerful open source library for computer vision and image recognition, available and compatible with a lot of programming language, such as python, c++, java and so on.

The best way to use it is with python, in combination with **numpy** that is another library highly optimized for numerical operations, increasing the capability of the designed app to handle complicated computer vision and image recognition scenarios.

Basically what is done inside the python code is first the initialization of the system, and then the check to assure that the setup is working properly.
The procedure is done as follows:

- Creation of a net object from a model in caffe network format:
  net = cv2.dnn.readNetFromCaffe(prototxt,model)
- Specify the NCS2 as the target for the inference:
  net.setPreferableTarget(cv2.dnn.DNN_TARGET_MYRIAD)
- Creation of the object for image acquisition from camera:
  vs = VideoStream(usePiCamera=True).start()

As it is possible to notice, the two first steps are really important for our specific application, otherwise it is not possible to use the **NCS2**.

After the initialization, the core of the program is an infinite loop in which:

- A frame is captured from the vs object:
  frame = vs.read()
- After some manipulation the image is converted into blob format:
  blob = cv2.dnn.blobFromImage(frame, 0.007843, (300, 300), 127.5)
- Pass the blob to the **NCS2**:
  net.setInput(blob)
- Retrieve the detections from **NCS2**:
  detections = net.forward()
- After some manipulation of the data obtained from the **NCS2**, merging them with the frame it is possible to display it in order to see in real time the input of the camera:
  cv2.imshow("Frame", frame)

This is basically the bone structure of the project, the decisions are based on the detections of the camera, regarding the control of the **2dof hat**, and the control of the position and motion of the robot, but this topics will be discussed in the next chapters.

In this next page there is a visual transposition of the algorithm described above, for sake of completeness and understanding the **Figure 3.2** is a simple workflow to understand the main cycle of the program.

A simple python script to test that the installation and setup went smoothly is attached in **section A.2**, in order to reproduce what was done in this chapter is necessary to use the same models and procedure.

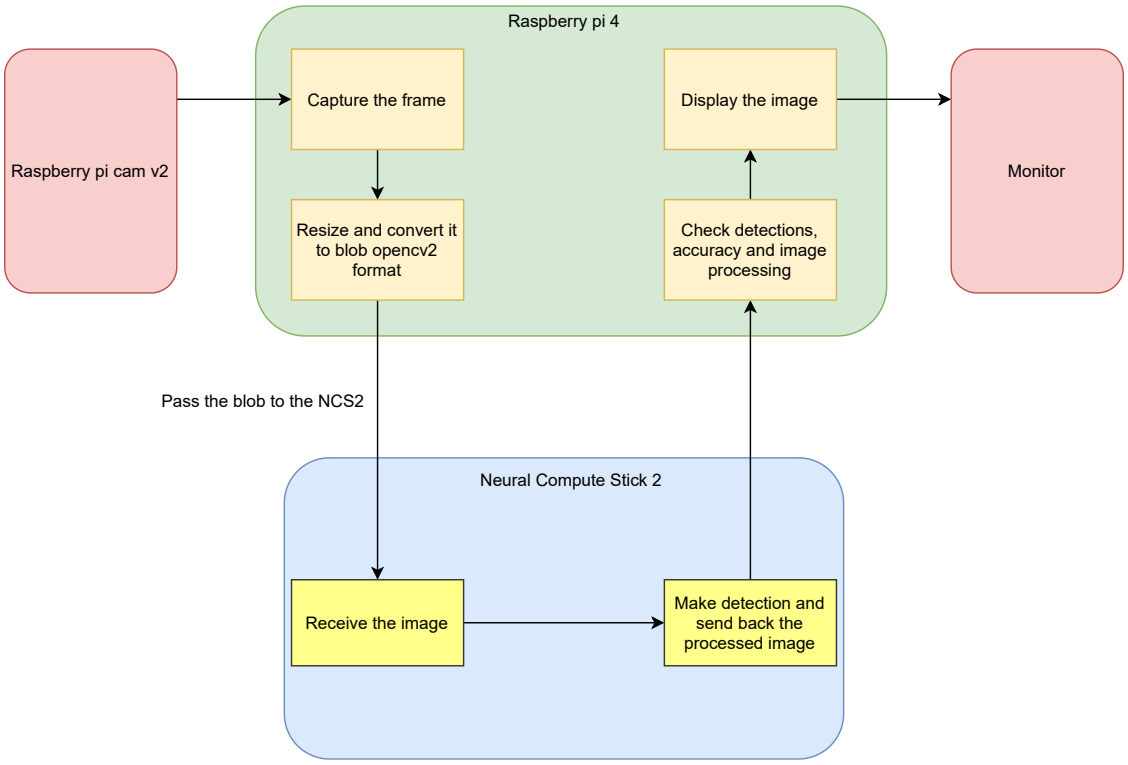

Figure 3.2: Raspberry + NCS2 workflow

In the figure it is clear that the raspberry pi camera is attached to the main computing device that is the raspberry.
The raspberry takes frame by frame the images of the environment from the camera and manipulates them in order to send the correct information to the NCS2.
The image that is received by the NCS2 is than analyzed and the results that come from the inference are then returned to the raspberry that can analyze and print on screen the results in real time.

Deep Neural networks

DNN (Deep Neural networks) are a key aspect for the computer vision and image recognition algorithm that is used in the project, so it is important to clarify some aspects that regard this topic.
The difference between DNN and other type of networks relies in the fact that the DNNs has more than two layers.
The DNNs also has sophisticated mathematical models that allows them to process the data.
The neural networks in general are a technology that tries to mimic the human brain capabilities to understand and manipulate unstructured data, such as pattern recognition and the processing of data through different layers of neural connections.
The DNN are networks that have an input layer, an output layer and at least one hidden layer in between. Each one of them perform a specific task that regards ordering and sorting the data, this process is called **feature hierarchy**.

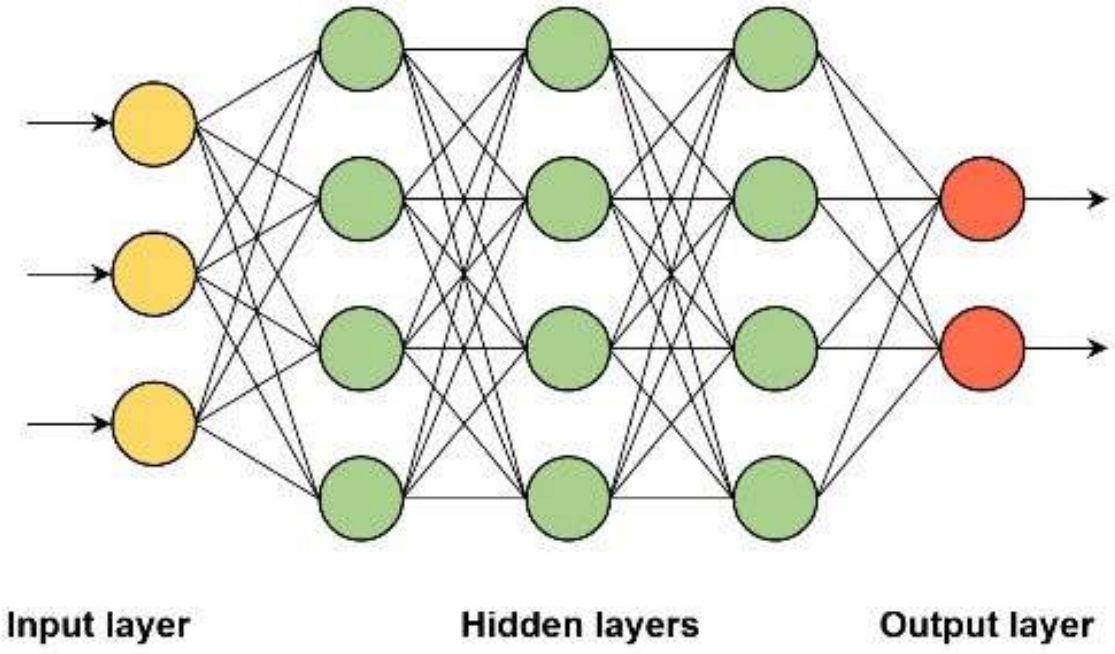

Figure 3.3: Visual representation of deep neural networks

To better understand the DNNs it is needed to know the process that stay behind its development.

At first it is needed the **ML** (Machine Learning), that is a framework used to automate with its algorithms the statistical models, like for example a linear regression model, in order to make better predictions.
A model makes prediction but they are not always accurate, so a model in which is present machine learning can change its weights in order to be more precise and makes less mistakes.
A model that has the ability to learn from the obtained results is the starting point for the creation of a **ANN** (artificial neural network).
The ANNs used their hidden layer to store and measure how the inputs are important for the outputs, and also makes associations between combinations of inputs.

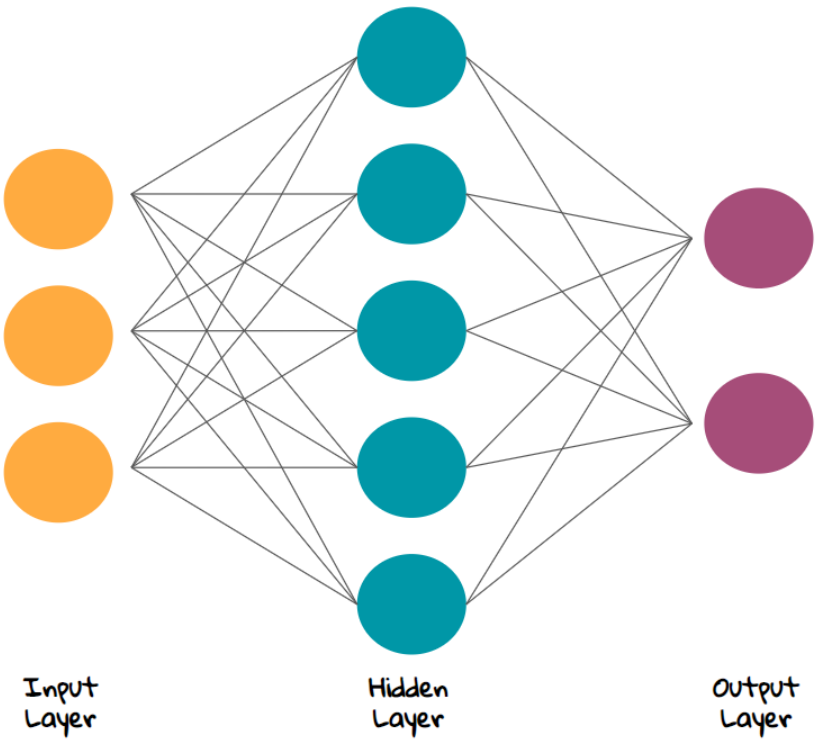

Figure 3.4: Visual representation of artificial neural networks

So as it is noticeable from the two images **Figure 3.3** and **Figure 3.4** they have different hidden layers.
In fact this is a property of the deep neural networks, **DEEP** refers to the fact that they are multiple hidden layers deep.
This feature anyway leads to a problem, since the model is trained to predict and tune using machine learning it is impossible to know how it precisely works, even if the single nodes values are known.
This because the the deep networks technology allows the model to make its own generalization and store them in a hidden layer.

Chapter 4

Pan-Tilt camera control

In this chapter will be discussed the hardware and software components used for the control algorithm of the raspberry pi camera, that allows the robot to track efficiently the user, moving the camera such that the target will be always bounded in a central region of its angle of view.

4.1 Hardware and software components

For how the project was thought it was needed a 2dof manipulator in order to deal with the pan and tilt control for the camera mobility, the idea behind this implementation stands in the fact that the camera should always maintain the target in its viewing angle, the most suitable way for this application was to move directly the camera instead of the whole robot.

It was decided to equip the **Raspberry** with the **Waveshare 2-DOF Pan-Tilt Hat**[12], that comes with a board adapted for the **Standard 40PIN GPIO extension header** of the **Raspberry**, see **Figure 4.1**.

The HAT is equipped with the **PCA9685** mounted on board as a **PWM driver**, the PWM resolution of this driver is of 12 bits and it permits to easily control the **two servomotors** mounted inside the joints of the arm in which the camera will be placed.
To control the motion of the Pan-Tilt system in python it is possible to use a library made up for the **PCA9685** that is present in **section B.1**.
The **PCA9685 python library** allows to easily control the motion of the camera with a simple list of methods that uses I2C communication present on the GPIO ports of the raspberry.

The structure needed in order to work properly with this hardware needs to follow the steps indicated below:

- Initialize the PWM for the PCA9685 creating an object from the library:
  pwm = PCA9685()
- Set the PWM frequency, in this case for this servo motors is 50Hz:
  pwm.setPWMFreq(50)
- Control the servos with:
  pwm.setRotationAngle(motor,angle)
  The motor variable can be 1 or 0, 1 stands for the PAN servo that can go from 0 to 180 degrees, while 0 stand for the TILT servo that can go from 90 to 180 degrees.
- Remember to clean the communication port and registers that the class is using with the method present in the library, otherwise it can lead to minor damage or anyway is not possible to use the communication with the next run of the program:
  pwm.exit_PCA9685()

The key methods stated above are the simplest and efficient way to control the HAT with python.

After the installation of the hardware and software components on the raspberry you can test the HAT with the code in **section B.2**.
The test code at run time tilt the camera from 90 to 180 degrees and pan the camera from 0 to 180 degrees in an infinite loop, just to be sure that the library and the installation is working properly without any issue.

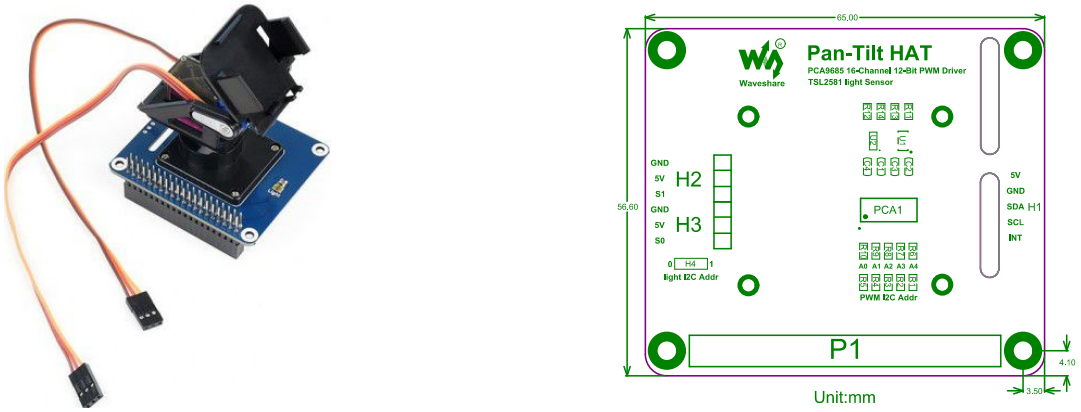

Figure 4.1: 2-DOF Hat with schematics

4.2 Control code for tracking of the user

The code is structured as a basic decision algorithm in python, what is used as a decision variable for the movements of the camera are empirically measured thresholds.
The thresholds are measured in pixels and are placed both vertically and horizontally, therefore the robot is able to adjust its angle of vision in the two axes easily.

Since the bounding box can vary a lot even if the target is still, probably due to light conditions and camera quality it was needed to reduce the uncertainty through a mean that will results almost the same at every iteration.
The mean is calculated for the vertical and for the horizontal axis taking the edge of both data and dividing them by two, retrieving the theoretical center of the target, from which we can draw a cross that helps along with the thresholds, without this method it is not possible to evaluate the position of the human due to the rapid change in the dimensions of the bounding box.
The thresholds are positioned such that if the target moves:

- Under 220 pixels or above 500 pixels horizontally the camera will pan of +10 or -10 degrees:

```
if(startX+endX)/2<220:
    if pan<180:
        pan=pan+10
        pwm.setRotationAngle(1,pan)
    elif(startX+endX)/2>500:
        if pan>0:
            pan=pan-10
            pwm.setRotationAngle(1, pan)
```

- Under 190 pixels or above 390 pixels vertically the camera will tilt of +5 or -5 degrees:

```
if(startY+endY)/2<190:
    if tilt>110:
        tilt=tilt-5
        pwm.setRotationAngle(0,tilt)
elif(startY+endY)/2>390:
    if tilt<180:
        tilt=tilt+5
        pwm.setRotationAngle(0, tilt)
```

- Since the robot is designed to stay on the ground and the point of view that it has of the space is under the target it was added also a constraint that locks the camera from tilting up too much, this condition would lead to a situation in which the system is in a state in which is necessary a reset:

```
if startY<70:
    if tilt>110:
        tilt=tilt-5
    pwm.setRotationAngle(0, tilt)
```

The **Figure 4.2** underneath explains visually how the algorithm works but for a more detailed explanation and comprehension of the control flow, **Appendix C** contains the portion of code inside the main loop of the program in which the control algorithm takes place.

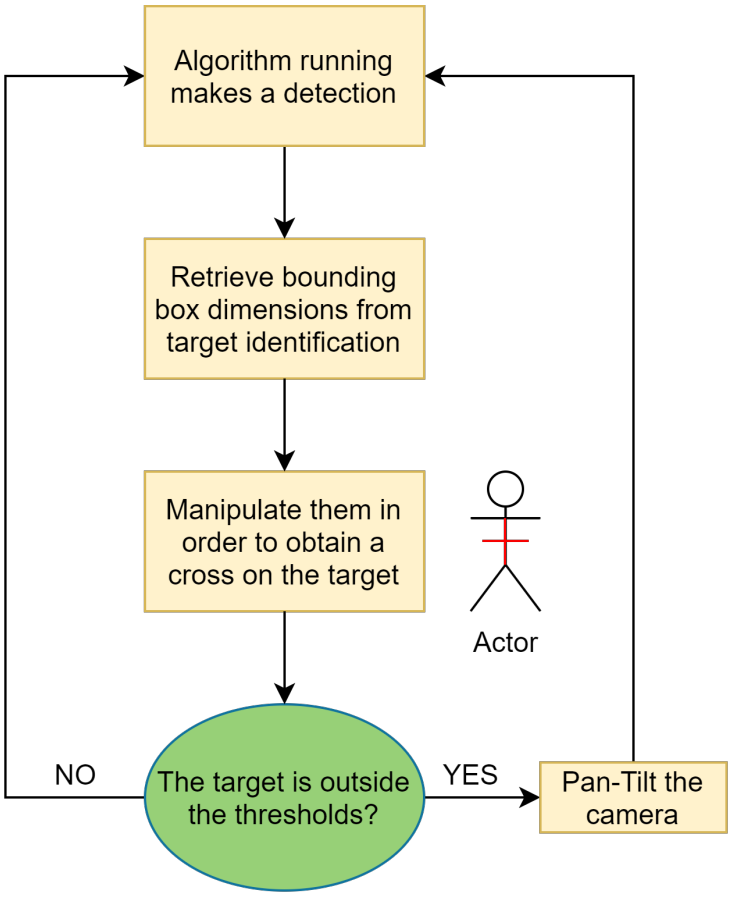

Figure 4.2: Workflow camera control algorithm

As described in the figure above the computer vision and image recognition algorithm will run in the infinite loop that is responsible for the human detection.
If a detection is done then the bounding box measured data are manipulated in order to obtain a mean value that is stable in time, since the bounding box dimensions could change widely even if the target stays still, this is due to light conditions and camera recording quality.
The mean instead is a value that stays almost the same during detections and so it can be used in order to focus on the center of the target and maintain the right moving trajectory.
With the data of the center of the target is possible to draw a cross on the frame that is used during the test phases to have a visual output to understand what the robot is aiming.
Having this data is possible through the thresholds mentioned above, to control the Pan-Tilt HAT at which the camera is attached.
In fact the center of the person at which the robot is aiming is fundamental in order to understand if the view angle of the camera needs to be adjusted.
The control algorithm in fact takes place when the person is moving out the thresholds, acting like described in the previous pages.

Chapter 5

Robot position control

In this chapter are present a brief explanation of the components used in the development of the body of the robot and its movement control, with also a discussion on the code that allows the robot to autonomously move following the target.

5.1 Hardware design and implementation of the circuit

The robot uses **4 DC motors** for its motion, that are driven by a **L6205N** motor driver.

Since the robot is a moving device it needs to be powered by a small source of energy that its capable to carry by himself, in the case of this project it was decided to use a battery pack with 20100mA capacity.

To power up the circuit it was needed a **buck-boost converter** that has a USB interface as input and a positive-negative output in order to apply the needed voltage to our circuit.
The converter and motor driver are shown in the **Figure 5.1** underneath.

Figure 5.1: Buck boost converter and L6205N motor driver

The circuit was soldered by hand on a **matrix board**, cut such that it could easily fit on the robot's platform with also the battery pack and the buck-boost converter.

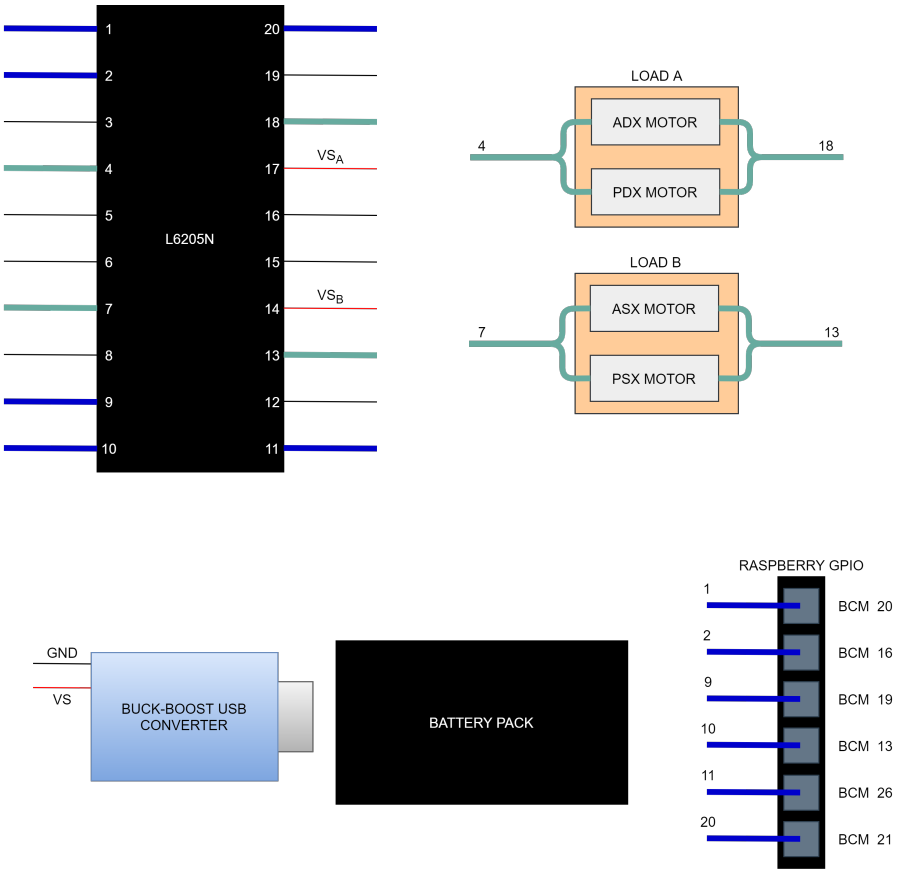

Figure 5.2: Pin and motors electrical scheme

During tests it was noticed a not negligible **voltage drop** from the power supply, due to the transient of the motors, this problem affects also the working conditions of the raspberry pi that was performs better if connected directly to the 220V through its charger.

The drop of the voltage has been easily corrected using the **PWM** (Pulse Width Modulation) to control the motors, in this way the transient does not effect directly the power supply, resulting in a more stable and reliable solution that allows the movements of the robot to be more precise even if slower.
This choice also permits a stable power transmission to the raspberry that can operate without continuous voltage drops due to the motor operations.

5.2 Software implementation for the robot's movements

The code for the motor control algorithm is placed in an external thread in order to operate independently with respect to the main thread completely dedicated to the computer vision and image recognition algorithm.

Since the camera has a limited angle of vision it is necessary to rotate the whole robot in order to always keep track of the target during its movements, this is done thanks to the moving platform equipped with four **DC motors**, each one of them is attached to a wheel.
The two side of the body are controlled separately, with this technique the robot is able to turn right or left staying in place, similar to a tank this moveset is perfect for the application because small, rapid and precise adjustement in the robot pose are needed in order to continuously track the human target.
With this approach the robot is programmed to rotate clockwise or counterclockwise, and to move forward or backwards.

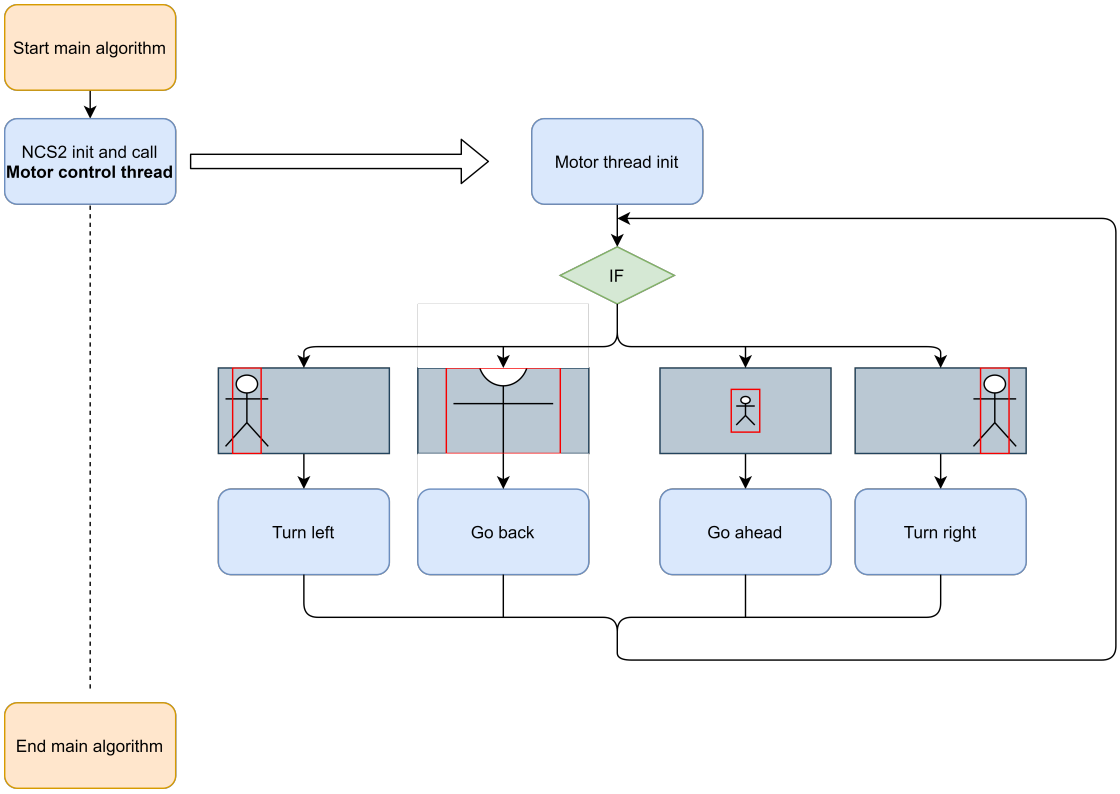

Figure 5.3: Workflow motor control algorithm

In **Figure 5.3** is presented a simplified workflow of the **motor control algorithm**, as stated above the algorithm is implemented inside a thread, this is done in order to always run the motor control algorithm independently, with respect of the main thread.
The program starts with the initialization of all the peripherals, including the **NCS2** and the **2DOF hat**, right before the main cycle dedicated to the computer vision and image recognition algorithm, the motor control thread is started.

After the initialization of the control variables and pins used to control the driver and consequently the motors, the thread starts to run in an infinite loop in which a decision tree is used in order to determine which action is necessary to do to always keep the human inside the camera field of vision.
In the figure is possible to see how the bounding box of the detection is used in order to decide how to move the robot, what basically the algorithm tries to do is to always keep a fixed distance between the robot and the person, resulting in a target that should be always centered inside the camera vision.
If the midpoint of the horizontal values of the bounding box is moved at the extremes of the field of vision of the robot's camera, and the 2DOF Hat is almost at the end of its range of motion then the motors are controlled in order to move the body left or right, trying to keep the person centered.

In the case the robot is already centered with respect of the person, what the algorithm does is decide if the distance from the human is acceptable, this is done computing the area of the bounding box, it is intuitive to say that if the bounding box area stays under a certain threshold then the robot moves forward, while in the contrary, if the bounding box area stays above a certain threshold than the robot moves backwards.

This operations are done in parallel to those of the camera, that in the same time tries to keep the person locked inside its field of vision using the **2DOF hat** presented in the previous chapter.
A brief explanation of the control algorithm of the camera is present in **section 4.2**.

The code relative to the control strategy treated in this chapter is located in **Appendix D**.

Chapter 6

Identification strategies

The system built until this point was capable to move and recognize a single human.
With this approach the robot was able to follow a person but due to the limitations in the hardware and computational power it was not possible to implement an algorithm capable to identify a specific human using just computer vision and image recognition.
At this point of the project it was needed a solution that could be applied to the system that is capable of tracking efficiently a human, the solution needs to be reliable and robust in order to track the person also in difficult scenarios in which are present other people, walls or other obstacles.
In literature most of the techniques that are used for this purpose required high computational power and since it is not suitable for this application it was decided to implement and test different new approaches.
In the following sections the techniques and technologies that will be treated comprehends computer vision and image recognition, infrared communication, BLE triangulation and IMU position tracking.

6.1 Computer vision and image recognition identification techniques

Since the computational power of the system was limited, the approaches with computer vision and image recognition algorithm tested for this application were two in particular.
A color detection algorithm and a QR code detection identifier algorithm were tested in order to recognize, track and follow only a specific person.

6.1.1 Color detection

For the application it was needed a technique that would allow the robot to identify a specific human with respect to the people or obstacle present in the scene.
The first solution implemented to solve this problem was color detection using computer vision.

With OpenCV it is possible to tune **hue, saturation and value (HSV)** in order to isolate specific colors, this means that with a trial and error procedure is very easy to understand which combination of these parameters allows the robot to identify the user by means of its clothes color.
The tuning was done in real-time using a window created with the **cv2.namedWindow()** OpenCV function, with a total of 6 changeable parameters: **Hue min, Hue max, Saturation min, Saturation max, Value min, Value max**.
In this way it is possible to tune the HSV during the execution of the program, having an instant visual output, useful for the trial and error procedure.

This solution was implemented and tested in a simple scenario, the environment had an artificial source of light and only one person in it.
After the manual color detection phase a test phase was done, in order to do this the person that had to be recognized by the camera moved inside the controlled scene but the results were not satisfactory.
In fact the human inside the camera's field of vision is recognized but only in certain points of the room, this was due to the fact that the change in light conditions could lead to a completely different tonality of the color detected by the camera, proving that even the slightest change for this parameter was crucial for the detection.
In order to improve this technique an hardware upgrade could help but it is for sure needed an accurate and precise test phase, the code in **section E.1** could be used in order to test different camera types and evaluate their performance under test.

6.1.2 QR code detection

Since the solution above was not suitable for the application, a new approach was implemented.
The next approach starts from the idea of a wearable tag, in this case it was chosen a **Quick response (QR)** code as identification method.

The QR code was used instead of a barcode because compared to the traditional two dimensional barcode it is easier to read by the digital devices.
This technique allows the human to have a unique identifier thanks to the information that the QR code can store, in this case the test was performed with a tag that stored a random identification string.

The identification and decoding of the informations from the QR code is really fast, thanks to its square-shape and its configuration matrix like, in which the pixels are easily distinguishable since they have just two colors, black and white.
Thanks to this the contrast created inside the square of the QR code makes it easier to read the information, even for old cameras.
The QR code is also really useful because it can store an amount of information such that also really heavy and long strings of data can be used as identifier, giving the possibility to think of an application that can be widespread without having different robots interfering with each other.

During test phases this approach turned out to be the easiest and simpler method to track a specific human, the amount of computational power needed in order to implement this solution were really low and so it was really suitable for the hardware present in the application.
The tracking and identification of the user, following this approach is incredibly robust and reliable, but the main drawbacks reside in the real possible application, in this way indeed the human should always wear a particular jacket or piece of clothing in which every side of it should have at least one or two QR code from every angle.
So for the purpose of this project, being this an application that should not be invasive and usable in every field without the necessity of special equipment, for example in a supermarket this solution was unfeasible.
Another drawback is the fact that in really chaotic scenarios with people and obstacles moving inside the scene, this simple tracking method can not work anymore due to the fact that a direct clean vision of the target is needed for the robot in order to proceed with the human following algorithm, otherwise it is not possible to track and identify a specific human.

The idea is really valid and could be implemented in other projects with other purposes or embedded inside this project for future works.
It is needed to say that an hardware upgrade regarding the camera is needed because the resolution and quality of the frames are not enough if the QR code stays too far away from the camera or under severe change in light conditions.

6.2 Infrared communication technique for the identification of the user

The next solution uses the **infrared (IR)** technology for the identification and tracking of a specific human.
The infrared technology was chosen because it was suitable for the project since it is a wireless mobile technology that can be used for devices communication in short ranges.
This technique was approached since the idea for the robot is to operate in common places, like for example supermarkets, and so it is needed a technology that does not affect animals like ultrasounds and others techniques that could hurt pets in any case.

6.2.1 Infrared protocols

Infrared communication like said before is safe from animals point of view.
An infrared LED in fact produce light that is not visible to the human eye, the wavelength of this type of signals ranges typically around 950 nanometers.
The easiest method to apply in order to communicate with infrared technology is to leave the IR LED on, representing a logical 1 for a certain period of time or turn it off for a certain amount of time, to represent a logical 0.
But unfortunately since the environment is full of many other sources of infrared radiation this approach would not work, to overcome this issue the standard used in IR communication for the signals frequencies, is around 38KHz, slightly different frequencies are available but in general this is the most common.
Flashing on and off the IR LED to represent data bits is not the right approach, in order to work with infrared signals a suitable protocol is needed, otherwise is not possible to understand what the data means, both transmitter and receiver side.

NEC protocol

So to communicate correctly one of the standard used is called **NEC protocol**.
This standard is used a lot in infrared communication, in fact it is common to have applications or devices that communicate with each other using this approach.
The NEC protocol uses a technique called **pulse distance encoding** to distinguish between the two logic states.
The carrier frequency is of 38KHz and the logic states are encoded using pulses of different lengths to distinguish between HIGH and LOW.
In the next figure it is possible to observe how a infrared signal could be encoded in order to approach this communication standard.

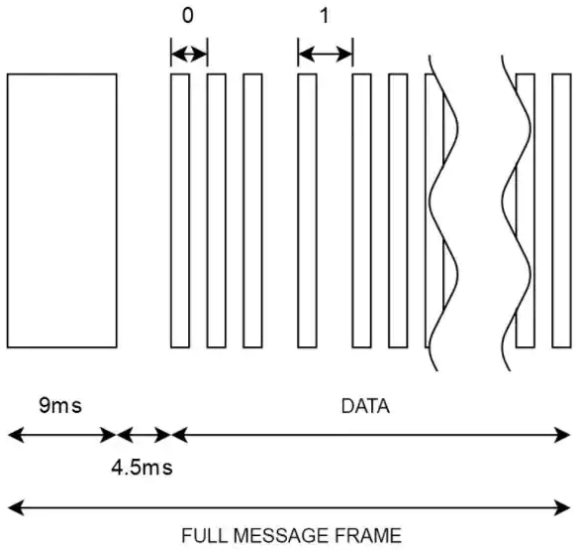

Figure 6.1: Visual representation of the NEC protocol

In **Figure 6.1** there are many elements to explain.
The binary values are encoded as follows:

- Logical 1: 562.2μs high signal followed by a 1.687ms low signal
- Logical 0: 562.2μs high signal followed by a 562.2μs low signal

While the rules to follow regarding the NEC protocol in order to send correct signals are:

- A 9ms leading high pulse
- A 4.5ms low signal
- The 8-bit address of the device
- The logical inverse of the first eight bits
- The 8-bit command
- the logical inverse of the command in bits
- A final 562.2μs high signal to signify the end of the message

The inverse of the address and of the message is used as a checksum to verify that the data received are correct.
The message and the address can be also of 16-bits, NEC protocol also support repeating messages.

R5C protocol

Another protocol is called **R5C** or **Philips protocol**. The R5C uses the **Manchester encoding** to distinguish between the logical bits in transmitted messages. The protocol uses a 36KHz carrier frequency and all the messages requires the same amount of time to be transmitted.

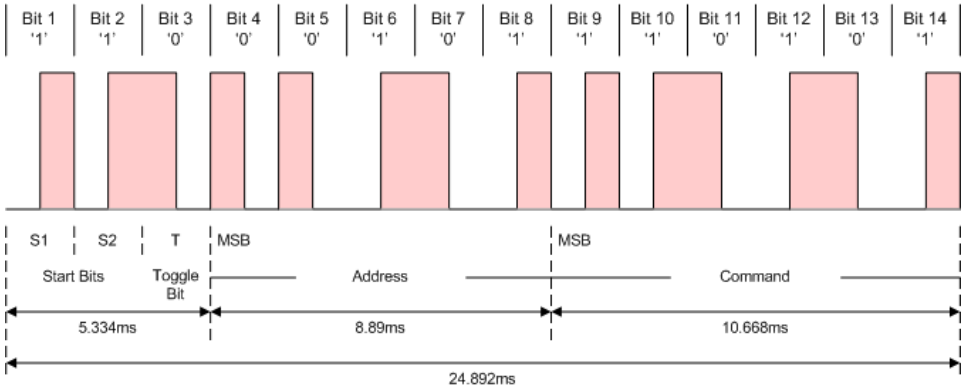

Figure 6.2: Visual representation of the R5C protocol

In **Figure 6.2** the binary values are encoded as follows:

- Logical 1: 889μs low signal followed by a 889μs high signal
- Logical 0: 889μs high signal followed by a 562.2μs low signal

The message frame has the following parts:

- Two high start bits
- A toggle bit that can be used to detect repeated signals
- Five address bits
- Six command bits

Every message is transmitted in a 24.892ms window.

6.2.2 Raspberry Rx - Arduino Tx project

The standards discussed above are the main techniques used in the IR communication.
In the implementation of this idea for the project, an Arduino UNO was used in order to drive the test IR LED transmitter **(TSAL6200)**, while on the other side an IR receiver **(TSOP946)** was connected to the GPIO ports of the Raspberry Pi 4.

For test purposes a simple code was developed, on the Arduino side a continuous train of 1 signal was sent, in order to measure just the performances on the receiver side, and evaluate if it was suitable for our application.
Since the idea was to develop a wearable device, like a pair of bracelets with soldered in its circumference equally spaced IR LEDs, the robot should be able to recognize a human and follow the one that is sending the correct code.
The bracelets from the project concept are able to send signals at 360 degrees, so it should be possible for the robot to receive the signal from every position, having mounted on itself different IR receivers.
The idea was to mount an IR receiver on every side and corner of the robot, trying to maximize the receiving surface with the least amount of receivers, since every one of them has a 45 degrees field of vision.
In order to test this solution a code was developed to analyze the receiver capabilities.
The code in **section E.2** was used to test the receiver previously connected to the raspberry pi 4.
Also a code capable to use this technology with the previously developed computer vision and image recognition algorithm can be found in **section E.3**.

The solution presented in this section was developed and then tested in order to evaluate its performances.
During test phase the algorithm with the IR technology fusion performed really well, the robot was capable to understand if the human that it was following was the real target.
The development phase was stopped obtaining just a system that was able to wait in idle until the person was returned in the scene, since the technology was difficult to apply in the project user case scenario.
The solution worked but an hardware upgrade is needed in order to obtain a robust and functioning system, the main drawbacks are the difficulties that exist in the working environment
Unfortunately the technology has problems working on long to mid ranges, because the IR LED and the IR receiver are not powerful enough to send and receive messages if the distance is more than 40/60cm.
Another reason why this technology would not fit well in our project is the fact that the IR signals can't pass through obstacles, walls or people.
So in a super market or in a warehouse the application of this technique is impossible, but the experiments and tests results are promising and can be used for other purposes and future works.

6.3 Bluetooth low energy triangulation for tracking and Identification of the user

Since the main problem of the previous approaches was that obstacles or people in the scene would interfere in the tracking solution, because it was needed the direct communication from the human to the robot, it was decided to implement a technology that could work efficiently even in this scenarios.
The technology implemented in this solution is the **Bluetooth Low Energy (BLE)**, in particular, the hardware used for this project was a raspberry pi 4 mounted on the robot, connected in serial communication with 3 **stm32f401re** microcontrollers with mounted on top a **X-NUCLEO-BNRG2A1** shield for bluetooth and wifi communication.

6.3.1 BLE technology

For the project was chosen the bluetooth technology because it is a short range (less than 100m), low power, low cost wireless communication that should work better with respect to the other solutions, because the communication can take place even with obstacles between the transmitter and the receiver.
Allowing the person to walk freely without taking care if the robot can physically see it or not.
The bluetooth radio frequency transceiver operates in a 2.4GHz band, the same range of frequencies used by the Wi-Fi.
The process of connecting wireless device is one of the complexity associated with the bluetooth technology.
The method used by the bluetooth and wireless technology is based on the principle of **"device inquiry"** and **"inquiry scan"**.
When the devices that use bluetooth as their method of communication needs to connect with each other, they **"listen"** on the known range of frequencies for devices that are actively inquiring.
When a device is detected, the one that is scanning will respond with it's address and all the information needed in order to be identified.
So the connection process can be described as follows:

- **Inquiry process**: The devices that don't know each other, in order to discover themselves must run an inquiry to understand if there are other devices in the area.
  The device that is inquiring scan the nearby space, while the other devices that are listening will respond with their basic information, such as address and device type.

- **Paging (Connecting) process**: This is the process that occur when connecting two bluetooth devices.

To do this it is necessary for both the devices to know the address of the other, that is known thanks to the inquiry step described above.

- **Connection process**: After the paging process will start the connection between the two devices, during the connection there are different states in which a device can be:
  - **Active mode**: This is the mode in which usually the devices operates in order to send or receive data.
  - **Sniff mode**: The device is less active and falls in power-saving mode listening for requests only at a set interval, for example every 100ms.
  - **Hold mode**: This state is sent by the master device, it tells to go into a sleep mode for a certain amount of time and then return to an active state.
  - **Park mode**: This is the deepest sleep mode, the master commands to a slave to enter this state and if needed to wake up.

Since the application needs to continuously track the device, it is needed an implementation that is capable from the smartphone side to always send data.
So it was suitable for the project to use a smartphone in order to emulate a beacon, in this way it is possible to not directly establish a connection between the robot and the user's device but it is possible to measure precisely the device **received signal strength indicator (RSSI)**.

Beacons

A bluetooth beacon is a device that works based on bluetooth low energy technology.
The device repeatedly transmits a constant signal that can be detected by other devices, just like a lighthouse but instead of transmitting a light beam, the communication happens using radio frequencies, so it is invisible and can be used even if in the environment are present different types of obstacles.
The main applications in which beacons are involved are:

- **Room-level monitoring**: Occupancy of rooms and buildings measuring how many bluetooth devices are present.
- **Manage visitors flow**: This can be done by measuring how many unauthorized bluetooth devices enter a building or a specific area in a company or workplace.
- **Indoor location localization**: It is possible placing different beacons in a specific area and measuring the RSSI relative to every device in the scene to understand the location of the target bluetooth device.
  Such measurements allows to estimate through triangulation the position.

The implementation done in this project uses a similar approach of the **indoor location localization** application described above.

6.3.2 Setup and implementation of the STM32F401RE with X-NUCLEO-BNRG2A1

Since the hardware chosen for the BLE tracking and identification application, is a **stm32f401re** microcontroller with mounted on top a **x-nucleo-bnrg2a1**, a detailed explanation of the software and procedure used to program the tracking platform is needed.
The microcontrollers are programmed using the software platforms developed by STMicroelectronics, **STM32CubeIDE** and **STM32CubeMX**.
After the installation of this two software platforms it is needed to install also the software package X-CUBE-GNSS1 in CubeMX in order to use the shield.

With the software package installed it is possible to create a new project, choose the right board in which the program will run and start to code.
After the steps above the CubeMX software will create a default project with the standard pins already initialized.
For the sake of simplicity STMicroelectronics allows to choose from a variety of different software packages and middleware in which is present also the X-CUBE-GNSS1 package downloaded and installed before.
Clicking in the software packages menu it is possible to include them in the project, it is important to include the package that is needed for the BLE shield in order code properly.
With the package it is also available a guide with all the necessary documentation. In the documentation are present the configuration steps for the pins, the peripherals and the middleware, this guidelines should be followed to configure properly the board in order to correctly communicate with the shield.
The right configuration for this particular application is the following:

- **PA_6 and PA_7**: Respectively SPI1 MISO and MOSI.
- **PA_8**: Reset for the BlueNRG-2 (GPIO_Output).
- **PB_3**: Clock SPI.
- **PA_0 and PA_1**: Respectively for the external interupt (GPIO_EXTI0) and chip select (GPIO_Output).
- **SPI1**: Set the SPI1 from the connectivity list.
  - Master full-duplex mode.
  - 8 bit data.

  - Two edge clock.
  - Prescaler for the Baud Rate set in order to have $\frac{Hclock}{Prescaler} \leq 1000\frac{Kbit}{s}$ For exmple a 64MHz clock with a prescaler of 64.

- **USART**: Already set by default for the stm32f401re.

- **Software packs**: check the **Wireless BlueNRG-2** box and in the platform settings set:

  - Exti Line → PA0-WKUP.
  - BUS IO driver → SPI1.
  - CS Line → PA1.
  - Reset Line → PA8.

- **System view**: go to NVIC in order to set the interupt, then set EXTI Line 0 as external interupt.

After this operations in order to generate all the needed files for the project is mandatory to go to the project manager section to name the project and select advanced as application structure.
When this steps are accomplished it is possible to generate the project folder with all the code needed for the application just clicking on **GENERATE CODE** in the top right corner.
The final pin configuration should look as in **Figure 6.3**.
All the steps described above were needed because this type of hardware are quite complicated and a wrong setup could easily ruin all the work, since this was the first application with bluetooth technology in the project, it was needed quite a careful setup since it is not easy as the infrared implementation for the tracking of the user described in **section 6.2**

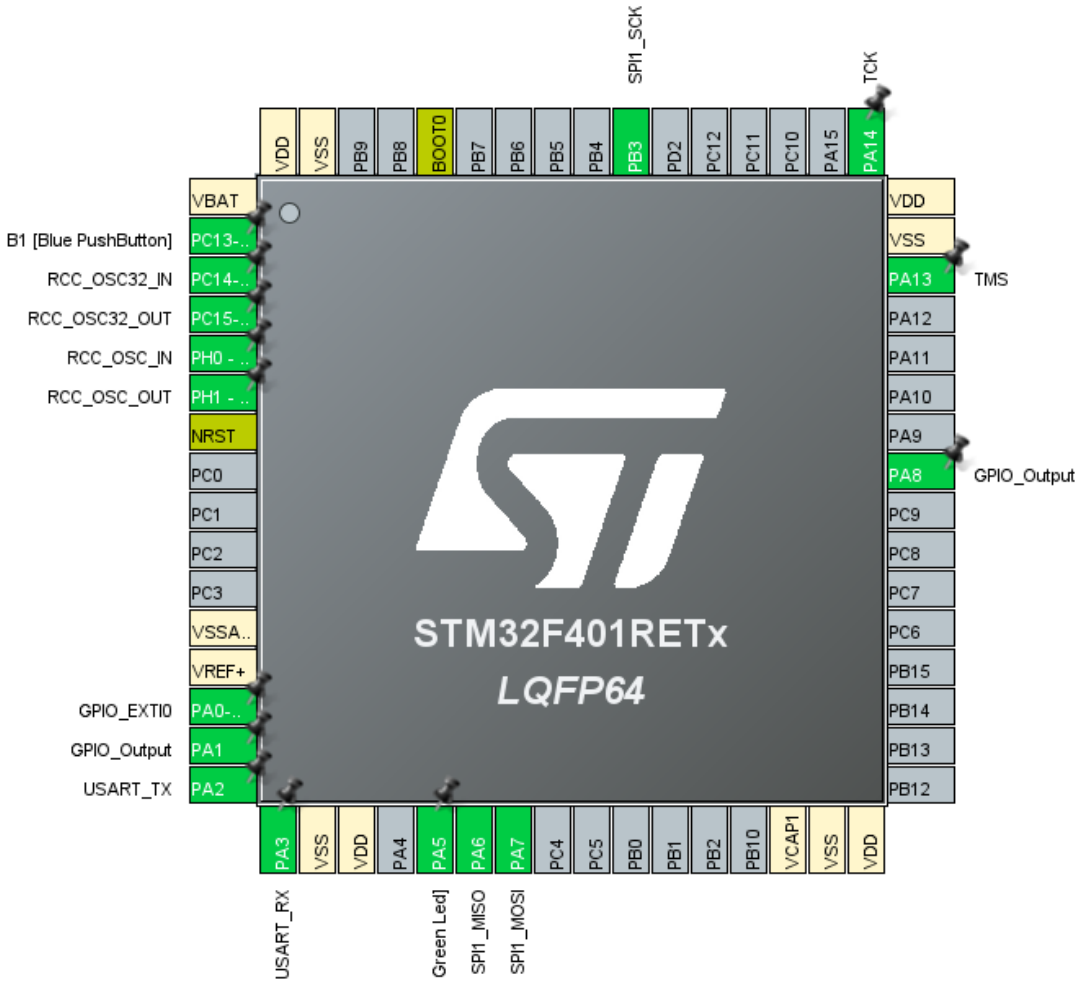

Figure 6.3: Final pin configuration

6.3.3 Coding the BlueNRG2A1

The coding phase that regards the stm32 microcontroller is developed on the software platform STM32CubeIDE, and the beacon simulation is done using an android app installed on the user smartphone called **Beacon simulator by Vincent Hiribarren**.
The code developed for this implementation can be found in **section E.4**.
The majority of the code is autogenerated from the steps that were described above, so the focus has to be placed in the **while(1)** loop, in which is developed the standard bluetooth procedure to discover other devices.

The code part is the following:

```
while (1)
{
    /*Activate and read the event list*/
    hci_user_evt_proc();
    /*Start the general scan*/
    ret = aci_gap_start_general_discovery_proc(0x4000, 0x4000,0x00,0x01);

    if (ret != BLE_STATUS_SUCCESS)
    {
        PRINT_DBG("Failure.\r\n")
        char buffer[100];
        sprintf(buffer,"%x\r\n",ret);
        HAL_UART_Transmit(&huart2,(uint8_t*)buffer,strlen(buffer),10);
    }

    HAL_Delay(100);

    /*Terminate the scan without waiting for a timeout process*/
    ret=aci_gap_terminate_gap_proc(0x02);

    if (ret != BLE_STATUS_SUCCESS)
    {
        PRINT_DBG("Failure.\r\n")
        char buffer[100];
        sprintf(buffer,"%x\r\n",ret);
        HAL_UART_Transmit(&huart2,(uint8_t*)buffer,strlen(buffer),10);
    }
}
```

As the comments highlights, in this piece of code the main functions are:

- **hci_user_evt_proc()**: It is a processing function that must be called in order to read the event list and trigger their respective callback.

- **aci_gap_start_general_discovery_proc(uint16_t scanInterval, uint16_t scanWindow, uint8_t own_address_type, uint8_t filterDuplicates)**: This is the function that commands the controller to start the active scanning procedure.
  To terminate the procedure is needed either the aci_gap_terminate_gap_procedure() function, or a timeout.

The devices discovered in this procedure are returned by evt_le_advertising_report. The parameters are:

  - **scanInterval**: This is the time interval between every scan, from 0x0004 to 0x4000, that corresponds in seconds to 2.5ms to 10240ms.
  - **scanWindow**: This is the time duration of a scan, that should be less or equal to the scan interval.
  - **own_address_type**: Type of address used during advertising public_addr (0) static_random_addr (1).
  - **filterDuplicates**: Duplicate filtering enabled or not, 0x00 will not filter duplicates, 0x01 will filter duplicates.

- **aci_gap_terminate_gap_proc()**: It is the function to terminate the specified GAP procedure.

The functions in the code above trigger an important event called **hci_le_ advertising_report_event**, in order to communicate correctly all the required data using the serial communication, a dedicated portion of code was developed for this event.
The code of the function is:

```
/* This callback is called when an advertising report is received */
void hci_le_advertising_report_event(uint8_t Num_Reports,Advertising_Report_t
,Advertising_Report[])
{
    if(user_detected==0)
    {
        int8_t RSSI = Advertising_Report[0].RSSI;
        if(RSSI>=-60)
        {
            user_detected=1;
            for(uint8_t loop = 0; loop < 6; loop++)
            {
                bdaddr_cb[loop] = Advertising_Report[0].Address[loop];
            }
        }
    }
    else
    {
        user_check=0;
        for(uint8_t loop = 0; loop < 6; loop++)
        {
```

```
            if(bdaddr_cb[loop] == Advertising_Report[0].Address[loop])
            {
                user_check++;
            }
        }
        if(user_check==6)
        {
            char buffer[100];

            /* BLE IDENTIFIER */

            //uint8_t identifier[3]="A\r\n";
            //uint8_t identifier[3]="B\r\n";
            uint8_t identifier[3]="C\r\n";
            HAL_UART_Transmit(&huart2,identifier,3,10);


            for(uint8_t loop = 0; loop < 5; loop++)
            {
                sprintf(buffer,"%d:",bdaddr_cb[loop]);
                HAL_UART_Transmit(&huart2,(uint8_t*)buffer,strlen(buffer),10);
            }
            sprintf(buffer,"%d\r\n",bdaddr_cb[5]);
            HAL_UART_Transmit(&huart2,(uint8_t*)buffer,strlen(buffer),10);


            /* type of the peer address (PUBLIC_ADDR,RANDOM_ADDR) */
            uint8_t bdaddr_type = Advertising_Report[0].Address_Type;

            sprintf(buffer,"%d\r\n",bdaddr_type);

            HAL_UART_Transmit(&huart2,(uint8_t*)buffer,strlen(buffer),10);

            /* event type (advertising packets types) */
            uint8_t evt_type = Advertising_Report[0].Event_Type ;

            sprintf(buffer,"%d\r\n",evt_type);

            HAL_UART_Transmit(&huart2,(uint8_t*)buffer,strlen(buffer),10);

            /* RSSI value */
            int8_t RSSI = Advertising_Report[0].RSSI;
```

```
64
65              sprintf(buffer,"%d\r\n",RSSI);
66
67              HAL_UART_Transmit(&huart2,(uint8_t*)buffer,strlen(buffer),10);
68          }
69      }
70  }
```

The portion of code cited above is structured so that an identification of the human's target smartphone is done between line 5 and 17, if the user is already detected than the normal communication algorithm takes place, sending the data through serial communication.

The identification of the user's smartphone is done scanning the nearby area searching for every bluetooth device
If the smartphone's beacon application is working properly with maximum power transmission, then it will be one of the different devices that will be identified.
Since for the project it is necessary to isolate and observe only the smartphone of the person, the code is structured such that it will store in memory only the address of the device that is sufficiently near to the microcontroller.
If the algorithm aknowledge its presence than the communication algorithm starts from line 17 to the end, sending the required data through serial communication.
The full code that can be replaced in the main file of the project, after its creation described in the subsections above, can be found in **section E.4**, in particular the code parts highlighted in the lines above are located between line 293-322 and 508-583.

6.3.4 Raspberry pi code for tracking and identification using bluetooth

After all the steps done until this point, in the project the focus was directed to the triangulation algorithm.
The 3 microcontrollers with their shield attached were connected through a usb hub (that has all the ports needed) to the usb port of the raspberry.
The whole system where tested under this conditions, the three microcontrollers were placed on a circumference near the raspberry, and the smartphone used for the test was placed at the center of this circumference for the initialization of the shields.

A visual representation of what has just been explained is contained in **Figure 6.4**.

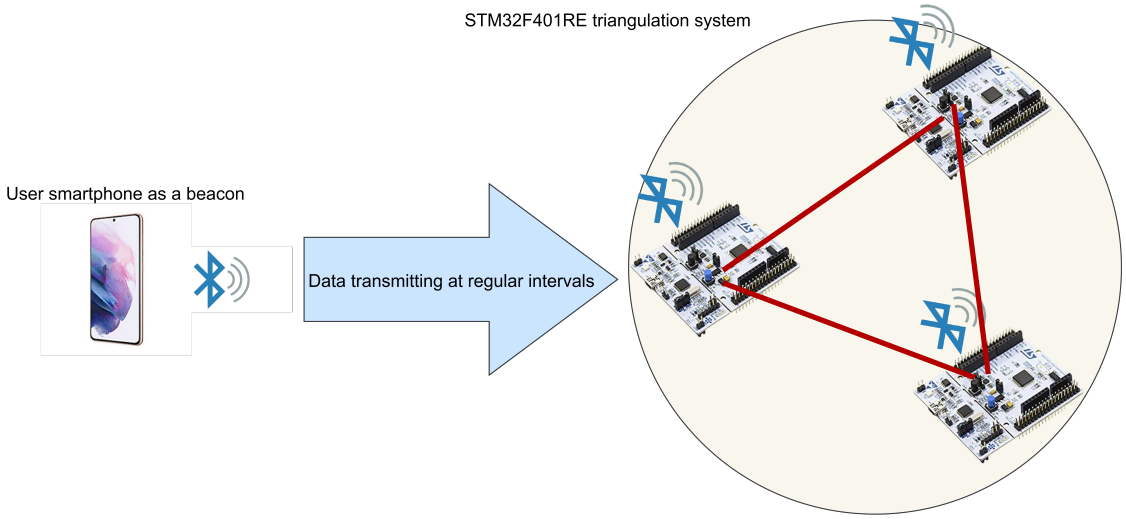

Figure 6.4: Visual representation of the bluetooth system's space configuration

So, through the RSSI data that the triangulation system delivers to the raspberry, the robot can estimate where the human is in space.
The code is structured so that three different serial connections are opened at the same time, this is possible thanks to the fact that every device is administrated by a thread that is able to retrieve the data and filter them in order to have a RSSI signal that is as smooth as possible.
Since the class of the serial communication and signal filtering is coded in order to run inside a designated thread the first lines of the code are reserved to the initialization of the different **Dev objects** (device objects) and to their coordinate identification in space.
The code was developed and tested under different conditions, at first the idea was to know the position in three dimensions, but using the RSSI the results were not precise enough in order to have a reliable system, so after some tests the approach chosen was two dimensional.
The computations done for the triangulation starts from the fact that the contact point between 3 sphere in the space is unique and so it is the point in which the person should be in space.
But unfortunately since there are uncertainties due to the RSSI measurements it is impossible to estimate a exact point in space, but there will be a 3D area in which it is probable to find the user.

As said before since in 3D space it is a difficult calculation and it requires much computational power and a more complex algorithm it was decided to translate the problem in 2D
So in order to obtain a compass that is able to give an estimation of the position of the user the problem was constrained into a two dimensional plane.

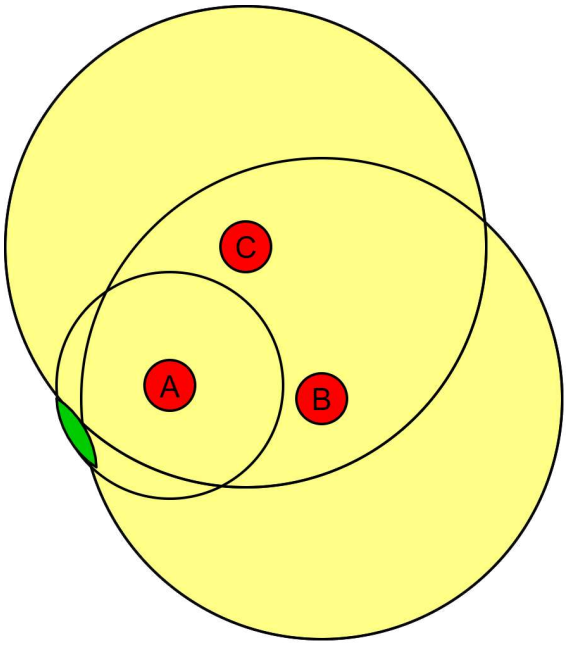

Figure 6.5: Triangulation uncertainty

As shown in **Figure 6.5** the uncertainty is located in a portion of plane in which the three circumferences are near each other but they do not intersect precisely.
After the initialization phase described in the previous page the code is than structured inside an infinite loop as follows.

In the loop a check is made to be sure that every bluetooth device is working properly and each one of them is tracking the same object, if this condition is true than a method called **getRSSI()** is called to obtain the current filtered RSSI value, from this point a calibration is done, and the rest of the code is composed by the mathematical computations needed in order to estimate the position of the user.
In the test phase, the results were precise in the first 20/25cm outside the center of the circle in which the three microcontroller were placed.

BLE results

The solution works, but inside a small area around the microcontrollers, after that the results were completely wrong due to the tolerance of the RSSI that after 30cm was always around -90dB, leading to wrong estimation.
As it is possible to notice from **Figure 6.6** and **Figure 6.7** the error measured after the 20cm circle discussed before is not negligible.
Another drawback for this solution was that obstacles between the transmitter and the receiver would interfere in the measurements of the RSSI values, leading to a wrong estimation of the position.
From the test results it was observed that for the tracking technique it was necessary an hardware upgrade both for receiver and transmitter, since the power transmission and reception were too weak in order to measure medium-long distances.
But the results were promising, in fact even if the signal power was too low in order to directly measure the distance of the device, it was enough to send information.
This idea led the project to a different technique that will be analysed in the next section.

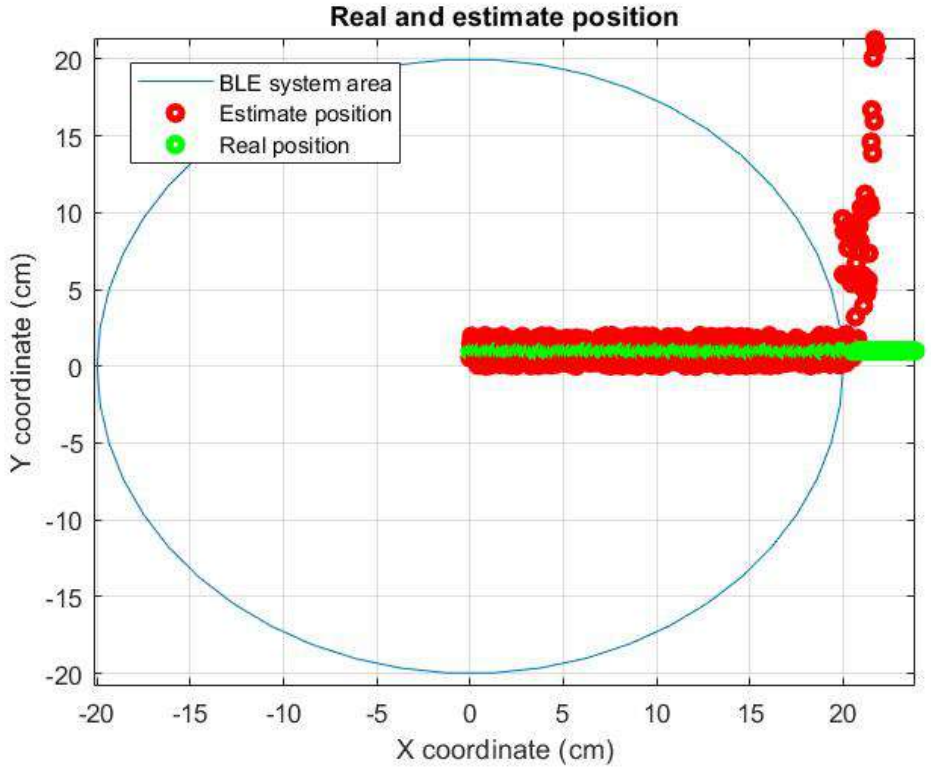

Figure 6.6: Real and estimate position comparison

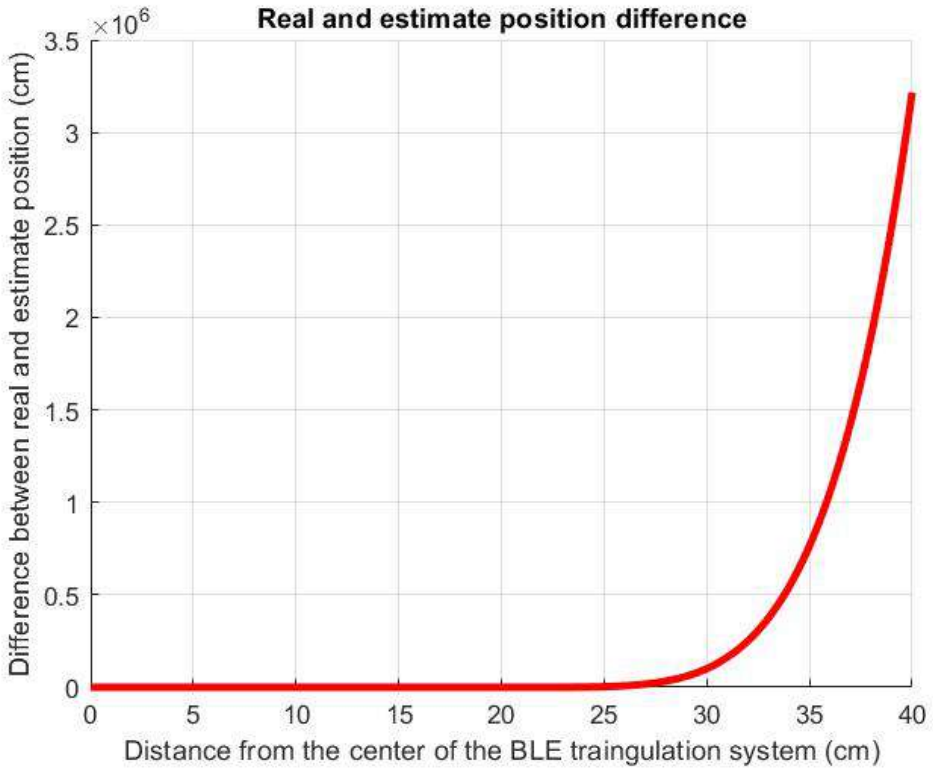

Figure 6.7: Real and estimate position difference

6.4 Inertial Measurement Unit position tracking identification technique

The next implementation for a functioning and robust tracking system was based on **Inertial Measurement Units (IMU)**.
IMUs are devices that measure the acceleration and the angular rate of the object to which they are attached.
They typically consist in a **Gyroscope** to measure angular rate, in a **Accelerometer** to measure force and acceleration, and in a **Magnetometer** to measure the magnetic field that surround the system.
The magnetometer is not always present on IMUs but its implementation combined with filtering algorithms helps in applications known as **Attitude and Heading Reference System (AHRS)**.
The idea is to measure from the different sensors on board, acceleration, angular rate and magnetic orientation, retrieving the orientation and the position in space, both for the robot and for the human.
This data then should be sent to the raspberry in order to be analysed and used for the computation of the relative distance between the robot and the person.

To deeply understand the application it was needed a research in literature to learn how the different sensors works, in this way it is easier to approach and solve the problems that regard this implementation.
The accelerometer is one of the sensors that compose the IMU and its the only one that is responsible for measuring the inertial acceleration and change in velocity over time.
The accelerometers can be found in different forms, mechanical accelerometer, quarts accelerometer and **Micro-electromechanical systems (MEMS)** accelerometer.
The MEMS accelerometer is the one mounted inside the shield used for the application, it is essentially a mass suspended by springs.
The mass is called **proof mass** and the axis in which it is a allowed to move are called **sensitivity axis**.
When a linear acceleration is present the mass is caused to shift to one side with the amount of displacement proportional to the acceleration.

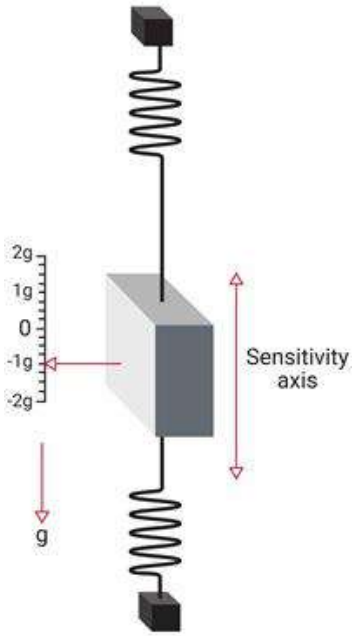

Figure 6.8: Simple accelerometer representation

The inertial sesnsor that measure the angular rate of an object is called gyroscope.
The gyroscopes like the accelerometers are divided in different configuration, such as mechanical, fiber-optic (FOGs), ring-laser (RLGs), and quartz/MEMS gyroscopes.
The MEMS gyroscope used in the application works using a vibrating mechanical element to sense the angular velocity.
Since there are not any mechanical rotating parts that requires bearings is very easy to miniaturize.

The sensor that measures the strength and direction of a magnetic field is called magnetometer.
MEMS magnetometer usually measure the surrounding magnetic fields by using magnetoresistance.
Magnetometers that rely on magnetoresistance are made up of permalloys that change resistance due to changes in the magnetic fields.

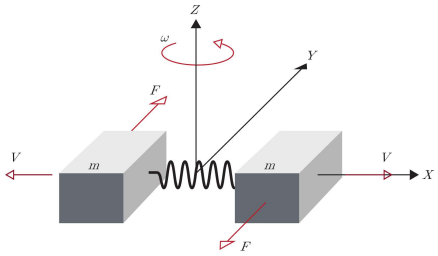

Figure 6.9: Simple gyroscope representation

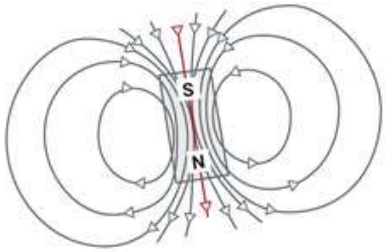

Figure 6.10: Simple magnetometer representation

A single inertial sensor can only sense a measurement along a single axis.
Triad are orthogonal cluster of three individual sensors mounter together in order to measure in three dimensions.
This configuration of individual sensor are usually called 3-axis inertial sensor, because the sensor can provide measurements relative to a 3D space.
The X-NUCLEO-IKS01A3 is a 9-axis system, because it has mounted on top accelerometers, gyroscopes and magnetometers.

6.4.1 Setup and implementation of the STM32F401RE with X-NUCLEO-IKS01A3

As in the previous chapter, for this application were used two stm32f401re microcontrollers but this time the shield is the X-NUCLEO-IKS01A3.
The idea is to send the data of the IMUs through the bluetooth technology, in order to be processed by the raspberry pi 4.
In this subsection will be treated how to get the correct setup in order to start the project with the **X-NUCLEO-IKS01A3**.
The first task is, like in the previous chapter, to install the software package that allows to program the board.
As before in STM32CubeMX with the install/remove button the software package window will pop up, in this case the package that is needed for the implementation is **X-CUBE-MEMS1**.

After installing the software package, in order to create a project STM32CubeMX will help giving different options regarding the target board and shield, after choosing the stm32f401re for the project the pin configuration setup will show up.
The setup will open with the default pin configuration, after that the software package is needed for the project, going in the software packs menu it is possible to select X-CUBE-MEMS1.
For this board different codes and setup were tested and developed, but the last project coded and tested is the most optimized.
In order to achieve this solution the **STM** gives some test application pre-coded in order to better understand the functioning and implementation using the IKS01A3 shield, in particular in the documentation are present a simple description of every application that regards this shield and its previous versions, the IKS01A2 and IKS01A1.
The application that is needed for the project is the **IKS01A3_DataLogFusion**, in this sample application made up by STM are present all the needed information for the robot, including quaternions and linear acceleration (brief theory in **section 6.4.1**).
All the dependencies relative to the application can be auto-filled using the software package manager.
As the application for the BLE triangulation in **subsection 6.3.2** the documentation for this software package can be found inside the package manager, it contains all the main functions and applications explained, but for further documentation the STM site of the shield contains everything that regards the software packages and hardware specification of the board.
In the documentation is described also step by step the pin setup for every application, so in order to start properly the coding phase with the right pin and middleware setup an overview of this steps needs to be taken.

The pin configuration is as follows:

- **PB_9 and PB_8**: Respectively as I2C1_SDA and I2C1_SCL.
  Also enable the I2C1 from the connectivity list with 400KHz clock speed.
- **PA_8**: Reset for the BlueNRG-2 (GPIO_Output).
- **PB_3**: Clock SPI.
- **USART**: Already set by default for the stm32f401re.
- **Software packs**: check the **Board extension IKS01A3**, **Sensor STM32 MotionFX Library** and **Device MEMS1 Application** boxes and in the platform settings set:
  - TIMER → TIM1_8:Internal Clock.

  - BSP BUTTON → GPIO:EXTI.
  - BSP USART → USART Asynchronous.
  - IKS01A3 BUS IO driver → I2C:I2C.
  - BSP LED → GPIO:Output.

- **System view**: go to NVIC in order to set the interupt, then set EXTI Line 15:10 and TIM3 as interupt.

To finish the setup procedure and start coding just it is necessary to go in the project manager, select the toolchain and the name of the project.
After this operations the button **generate code** will create the project folder and open STM32CubeIDE.
At the end of this phase it is important that everything is set as described above, the final pin configuration should look as in **Figure 6.11**.

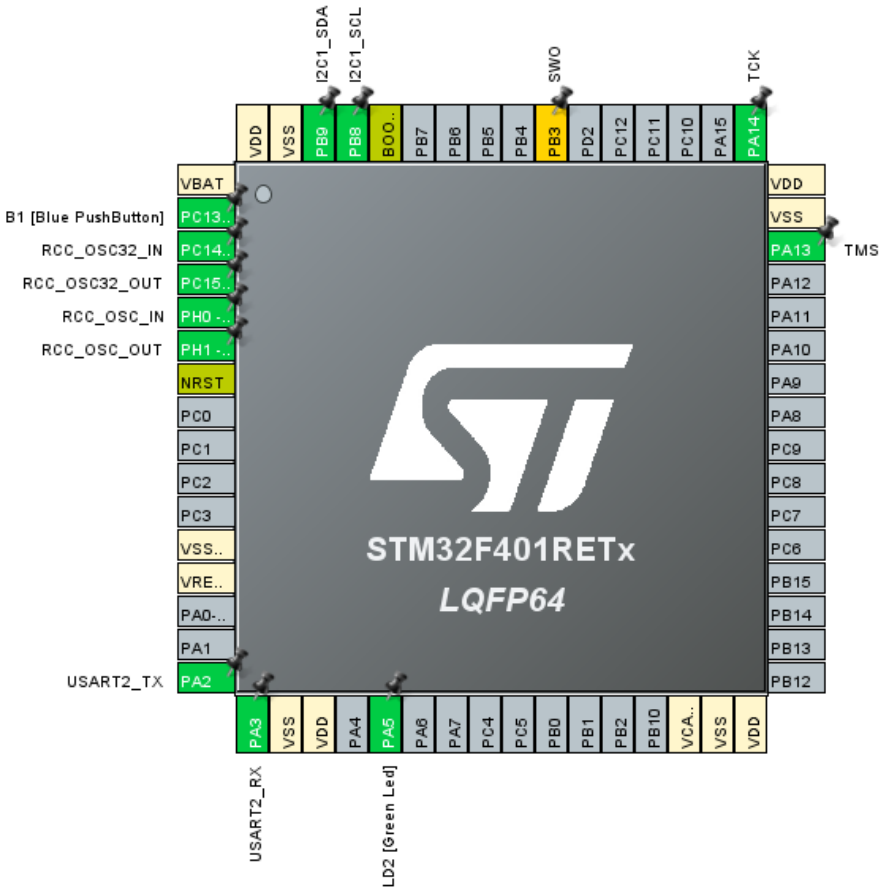

Figure 6.11: Final IMU pin configuration

Brief introduction on quaternions and rotation matrices

Since quaternions and rotation matrices are an important tool in robotics it is necessary to discuss about their use in the project.
The quaternion is one of the many mathematical ways to identify the orientation of an object in space with respect to a fixed reference frame.
Another representation for example are the Euler angles rotation matrices, but the quaternions compared to them are a more compact form, are numerically more stable and computationally more efficient from the computer point of view.
It is important to remark that quaternions gives information only on the orientation of the body but not on the position, to do this it should be used a **Roto-Translation matrix** but it is less efficient from a computational point of view.
Instead the quaternions can be used in order to identify the orientation in which a body is moving and than compute the position in a different vector.
Quaternions are so powerful that they are the default mathematical object used to define the orientation on **Robotic Operative System (ROS)**.
Since quaternions are an extention of complex numbers they can be represented by a vector of four values, the four values are one scalar and three element unit vector.
The representation of the quaternion can be write as:

$$q = a + bi + cj + dk \Longrightarrow q = \begin{bmatrix} a \\ bi \\ cj \\ dk \end{bmatrix} \Longrightarrow q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \Longrightarrow q = \begin{bmatrix} q_0 \\ \mathbf{q} \end{bmatrix}$$

Where q_0 is the angle of rotation represented by a scalar, q_1, q_2 and q_3 is the axis of rotation to which the action of rotation is performed.
While rotation matrices are more intuitive there are problems related to possible singularities that can occur in particular rotation.
Rotation matrices are 3x3 matrices that describe the rotation of a reference frame with respect to another one, using the right hand rule.
There are three main rotation matrices that corresponds to the three primary rotations around x, y and z axis.

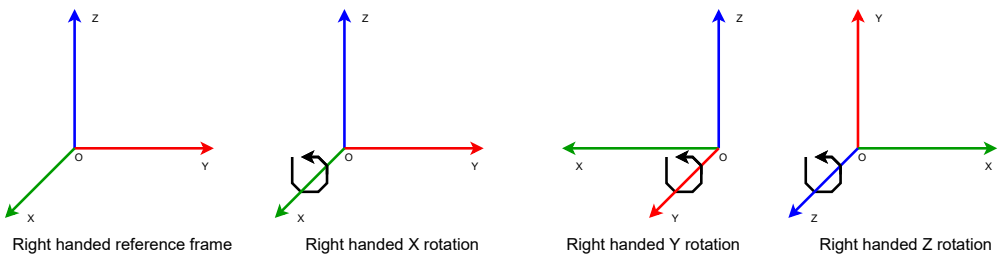

Figure 6.12: Right handed reference frames and primary rotations

The three main rotation around the principal axis can be represented with the following three rotation matrix:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \qquad R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \qquad R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The combination of rotations using the rotation matrices with the Euler angles results in a sequence of matrix multiplication from the first rotation to the last, there are more complex rules in this case if the rotations are done with respect to a fixed frame or with respect to the moving reference frame.
For sake of completeness the following rotations are supposed to be done with respect to the mobile and fixed reference frame.
If for example we have a rotation about the X-axes of ϕ degrees, then a rotation about the fixed Y-axes of θ degrees and a final rotation about the mobile Z-axes of ψ degrees, the resulting rotation matrix from the combination of this rotations can be computed as follows:

$$R_{z(\psi)y(\theta)x(\phi)} = R_{z(\psi)} \cdot R_{y(\theta)} \cdot R_{x(\phi)}$$

$$\Downarrow$$

$$\begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

Computing from the right to the left we have that:

$$R_{y(\theta)} \cdot R_{x(\phi)} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} \cos\theta & \sin\theta\sin\phi & \sin\theta\cos\phi \\ 0 & \cos\phi & -\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

And then:

$$R_{z(\psi)} \cdot R_{y(\theta)x(\phi)} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & \sin\theta\sin\phi & \sin\theta\cos\phi \\ 0 & \cos\phi & -\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

This were the standard operation to be done in order to manipulate an object using the rotation matrices, instead a rotation using quaternions is much more powerful, more robust (since there are no singularities), and has a higher computational efficiency.
To understand the computation of the resulting quaternion is necessary to know the mathematical properties of this object.
Since the quaternion is an extension of complex numbers the multiplication between two of them would be:

$$(q_0 + q_1 i + q_2 j + q_3 k) \cdot (p_0 + p_1 i + p_2 j + p_3 k)$$

$$\Downarrow$$

$$\begin{bmatrix} q_0p_0 - q_1p_1 - q_2p_2 - q_3p_3 \\ (q_0p_1 + q_1p_0 + q_2p_3 - q_3p_2)i \\ (q_0p_2 + q_2p_0 - q_1p_3 + q_3p_1)j \\ (q_0p_3 + q_3p_0 + q_1p_2 - q_2p_1)k \end{bmatrix}$$

Now the properties of a given quaternion $q = q_0 + q_1 i + q_2 j + q_3 k$ are:

- Conjugate:
$$q^* = q_0 - q_1 i - q_2 j - q_3 k$$
- Norm:
$$|q| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$
- Inverse:
$$q^{-1} = \frac{q^*}{|q|^2}$$

With this information is now possible to compute a rotation using quaternions.
A coordinate vector in 3D space can be written as a **pure quaternion**, meaning that his quaternion form has zero real part $q = 0 + xi + yj + zk$.
So a rotation q_R from a coordinate frame q_A to a coordinate frame q_B can be expressed as:

$$q_B = q_R \cdot q_A \cdot q_R^*$$

This were the main properties and computations that are needed for both rotation matrices and quaternions.
It is easy to understand that quaternions are computationally more efficient and

robust, but rotation matrices are really usefull in case there is no need for a really fast and stable system.
Since in the section above this topics were covered, is important to know that conversions can be done both from qaternions to rotation matrix and viceversa, but most importantly from the quaternions it is possible to retrieve roll, pitch and yaw angles.
From quaternion to rotation matrix:

$$R = 2 \cdot \begin{bmatrix} q_0^2 + q_1^2 - 0.5 & q_1q_2 - q_0q_3 & q_0q_2 + q_1q_3 \\ q_0q_3 + q_1q_2 & q_0^2 + q_2^2 - 0.5 & q_2q_3 - q_0q_1 \\ q_1q_3 - q_0q_2 & q_0q_1 + q_2q_3 & q_0^2 + q_3^2 - 0.5 \end{bmatrix}$$

From rotation matrix to quaternion:

$$tr(R) = R_{11} + R_{22} + R_{33} = 4q_0^2 - 1$$

In this equation tr stands for **trace** of R.
So we have now that:

$$|q_0| = \sqrt{\frac{tr(R) + 1}{4}}$$

And so we can compute:

$$|q_1| = \sqrt{\frac{R_{11}}{2} + \frac{1 - tr(R)}{4}}$$

$$|q_2| = \sqrt{\frac{R_{22}}{2} + \frac{1 - tr(R)}{4}}$$

$$|q_3| = \sqrt{\frac{R_{33}}{2} + \frac{1 - tr(R)}{4}}$$

While for the conversion from quaternion to roll, pitch and yaw angles we have that:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} atan2(\frac{2(q_0q_1 + q_2q_3)}{1 - 2(q_1^2 + q_2^2)}) \\ asin(2(q_0q_2 - q_3q_1)) \\ atan2(\frac{2(q_0q_3 + q_1q_2)}{1 - 2(q_2^2 + q_3^2)}) \end{bmatrix}$$

6.4.2 Coding the IKS01A3

Since in the chapter above were treated the setup of the stm32f401re and its IMU shield, with a brief discussion about the rotational mechanics of rigid bodies and some of its mathematical object, it is now possible to fully understand the data and the computations that are needed in order to obtain a full working system.

For this application were tested different algorithms and functions, one of the most important is for sure the **Madgwick AHRS algorithm**.
This solution were implemented on a completely custom code, retrieving the raw data from the shield's accelerometer, gyroscope and magnetometer it was possible to obtain the quaternions that indicates the orientation of the object (in this case the board itself) with as a reference the starting position.
This method was discarded since it was not possible to obtain good measurements because the algorithm didn't take in consideration the pure translation of the object, this particular leads the system to think a rotation was done when instead the board only moved across the table.

It was needed a method that grants through filters and calibration a result that was realistic and that neglect every translation, translating the data in order to only consider pure rotation even if the body is moving.
So in this case after different version of the code and Kalman filter implementations it was decided to change approach, instead of creating a completely custom application the idea was to search for a project that has the specification for this implementation and customize the final code, in order to obtain the required results.

It was really helpful the folder containing the set of applications provided by **STM**, in fact one of this application has most of the important features that were needed for the project, the application name is **IKS01A3_DataLogFusion**.
This application was developed in order to demonstrate how the MotionFX middleware developed by STM could be used.
This application was developed in order to be used with another software developed by STM, called Unicleo-GUI.
The Unicleo-GUI offers different options such as the temperature and humidity measurements and other features that are not needed in the project.
But the whole application has inside a real-time motion sensor data fusion that allows the board to precisely measure its orientation taking as reference frame the magnetic north of the earth.
It also has really important built-in features like the gyroscope bias compensation and hard iron calibration for the magnetometer.
The code is written inside the autogenerated project inside the **source** folder,in the file named **app_mems.c**.

The functions dedicated to the calibration and other major modification like the sensors initialization that were done in order to use this application in the project can be found in **section E.6**.

```
static void FX_Data_Handler()
{
    uint32_t elapsed_time_us = 0U;
    MFX_input_t data_in;
    MFX_input_t *pdata_in = &data_in;
    MFX_output_t data_out;
    MFX_output_t *pdata_out = &data_out;

    /* Convert angular velocity from [mdps] to [dps] */
    data_in.gyro[0] = (float)GyrValue.x * FROM_MDPS_TO_DPS;
    data_in.gyro[1] = (float)GyrValue.y * FROM_MDPS_TO_DPS;
    data_in.gyro[2] = (float)GyrValue.z * FROM_MDPS_TO_DPS;

    /* Convert acceleration from [mg] to [g] */
    data_in.acc[0] = (float)AccValue.x * FROM_MG_TO_G;
    data_in.acc[1] = (float)AccValue.y * FROM_MG_TO_G;
    data_in.acc[2] = (float)AccValue.z * FROM_MG_TO_G;

    /* Convert magnetic field intensity from [mGauss] to [uT / 50] */
    data_in.mag[0] = (float)MagValue.x * FROM_MGAUSS_TO_UT50;
    data_in.mag[1] = (float)MagValue.y * FROM_MGAUSS_TO_UT50;
    data_in.mag[2] = (float)MagValue.z * FROM_MGAUSS_TO_UT50;

    gcvt(data_in.acc[0],6,ax_send);
    gcvt(data_in.acc[1],6,ay_send);
    gcvt(data_in.acc[2],6,az_send);

    /* Run Sensor Fusion algorithm */
    BSP_LED_On(LED2);
    DWT_Start();
    MotionFX_manager_run(pdata_in, pdata_out, MOTION_FX_ENGINE_DELTATIME);
    elapsed_time_us = DWT_Stop();
    BSP_LED_Off(LED2);

    data_to_send.quaternion[0]=pdata_out->quaternion[0];
    data_to_send.quaternion[1]=pdata_out->quaternion[1];
    data_to_send.quaternion[2]=pdata_out->quaternion[2];
    data_to_send.quaternion[3]=pdata_out->quaternion[3];
```

```
39
40      gcvt(data_to_send.quaternion[0],6,quat0_send);
41      gcvt(data_to_send.quaternion[1],6,quat1_send);
42      gcvt(data_to_send.quaternion[2],6,quat2_send);
43      gcvt(data_to_send.quaternion[3],6,quat3_send);
44
45      data_to_send.rotation[0]=pdata_out->rotation[0];
46      data_to_send.rotation[1]=pdata_out->rotation[1];
47      data_to_send.rotation[2]=pdata_out->rotation[2];
48
49      gcvt(data_to_send.rotation[0],6,rot0_send);
50      gcvt(data_to_send.rotation[1],6,rot1_send);
51      gcvt(data_to_send.rotation[2],6,rot2_send);
52
53      data_to_send.linear_acceleration[0]=pdata_out->linear_acceleration[0];
54      data_to_send.linear_acceleration[1]=pdata_out->linear_acceleration[1];
55      data_to_send.linear_acceleration[2]=pdata_out->linear_acceleration[2];
56
57      gcvt(data_to_send.linear_acceleration[0],6,acc0_send);
58      gcvt(data_to_send.linear_acceleration[1],6,acc1_send);
59      gcvt(data_to_send.linear_acceleration[2],6,acc2_send);
60
61      snprintf(dataOut, MAX_BUF_SIZE,
62      "\r\n%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s\r\n", quat0_send,
63      quat1_send,quat2_send,quat3_send,rot0_send,rot1_send,rot2_send,
64      acc0_send,acc1_send,acc2_send,ax_send, ay_send, az_send);
65      printf("%s", dataOut);
66  }
```

The code above is an extract of what is inside the **section E.6**, this is the core of the application.
In this portion of code every useful data is stored and transformed into a string, thanks to the **gcvt()** function it is possible to transform the float values into strings with the precision needed for the project.
The very important function that implements a very well designed Kalman filter is **MotionFX_manager_run()** that allows, given in input the raw data, to estimate with high precision other parameters such as quaternions, gravity vector, linear acceleration, heading and euler angles.
After the process of filtering and estimation, the data retrieved from the Kalman filter function are processed like said above and then everything is formatted in order to be sent through serial with the **snprintf()** and **printf()** functions.

6.4.3 Raspberry pi 4 position estimation coding

The first codes as stated in the previous section had the idea to estimate position and orientation from the raw data retrieved from accelerometer, gyroscope and magnetometer.
Unfortunately this simple idea didn't work well since all the designed Kalman filters and approaches tested during this phase led to inaccurate measurements.
In **Figure 6.13** is possible to notice how the drift due to the double integration led to inaccurate results, the green line represents the real up and down movement but as in figure the red line that represents the estimation, diverge, exploiting the behaviour encountered in test phase.

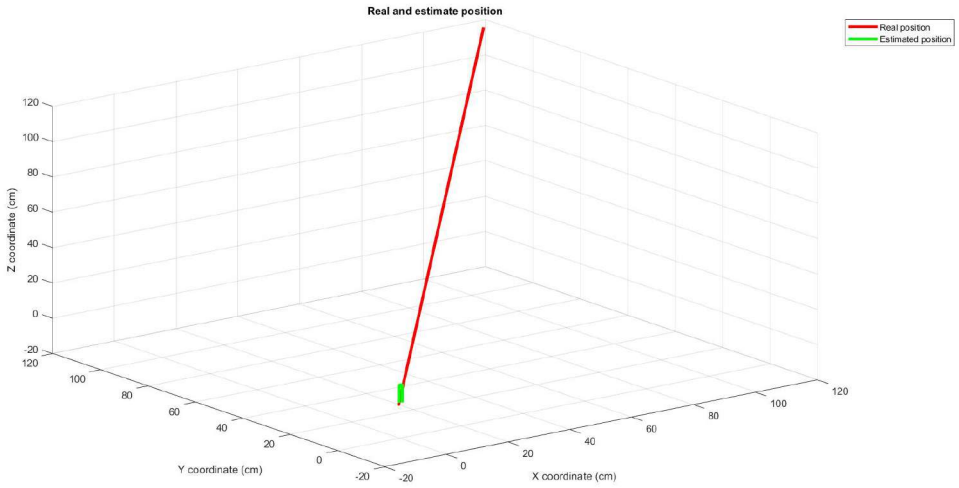

Figure 6.13: Drift representation

After the last implementation on the stm32f401re discussed earlier it was decided to use the precise measurements of the quaternion and linear acceleration made by the microcontroller and use them to compute the position.
It is on the coding phase of the raspberry pi that the first problems regarding the noise shows up.
In fact in order to estimate the next position using the current orientation and current position is necessary to do a **double integration**, that due to the noise on

the measurements is impossible to obtain in a robust and precise manner.
This led to the idea of filters implementation, since the Kalman filter is working on the microcontroller the only thing to implement on the raspberry was a filter capable of getting rid of all the noise that was coming from the measurements.
In fact the code implementation has a built-in **butter highpass digital filter**, both in the 2D and 3D implementation that are present in the appendix of this thesis.

3D version

In this version of the code the main focus were on the quaternion and their usage inside the algorithm to get the position of the board.
In fact if is needed for future project, a function for the conversion from quaternion to rotation matrix can be found in the file of this implementation.
But since this approach didn't work well, due to limitations and singularities it was decided to proceed using directly the filtered roll, pitch and yaw angles delivered from the microcontroller in order to compute the position with a simpler algorithm.
In this code also a visual implementation in real time was developed, but due to the errors in the measurements and computation it was not working.
To avoid and reduce the drift caused by the noise of the measurements, a filter was implemented using the signal object from scipy library.
The scipy library is very useful and offers a lot of tools for different situations, in fact inside the library is present also a signal object, that is used in our project to design and implement a digital filter.
The filter used is a butter highpass digital filter of first order.
This filter is pretty simple but a small reduction in the error was noticed after the tests phase, still not enough to avoid the propagation of the errors in the double integration, the drift is too much in order to have precise and reliable measurements.
It was added also a calibration phase in which a number of predetermined samples are stored at the start of the application, and the mean is then subtracted to the measurements retrieved in the working scenario, trying to reduce the fixed noise component.
It is necessary to say that during time slight improvement were made, but unfortunately it was not enough in order to obtain satisfactory results.
The code discussed can be found in **section E.7**

2D version

The 2D version of the code is implemented also on the raspberry using the same methodologies of the 3D version.
Python and its scipy library were used to implement like in the 3D version a filter capable of reducing the noise in the measurements.
The aim of this test code was to see if with more constraints on the algorithm, so

reducing the degrees of freedom it was possible to obtain better results.
Unfortunately even if the results were better it was not enough for the application. The measurements were stable if the board was stationary but with some rapid movements the drift propagation becomes too big in order to obtain reliable results.
The code discussed for the 2D application can be found in **section E.8**.

Conclusion

From the results obtained in this phase it was clear that the problem was the drift, so even if the measurements were not satisfactory they were useful to understand the functioning and the possible solutions of this problems related to the IMUs usage.
So it was decided to test this type of application using more complex and precise algorithm with the help of matlab, switching from the raspberry to a more powerful and efficient personal computer.

6.4.4 MATLAB position estimation algorithm

As stated in the previous section the decision for the IMU tracking was to develop an algorithm capable of tracking the human position using MATLAB.
This decision was due to the lack of power and code optimization needed in order to obtain a robust and reliable result using the raspberry pi 4 and python.
Since in literature this solution is often implemented with custom and really powerful neural network, or optimized sensor fusion algorithm using also other technologies like the GPS or speed sensors mounted on the wheels, it was needed a test that would help in future work to understand if this approach is possible with just an IMU.
The code developed for this application was a modification of a code made by a student named **Zio Leonardo**.
Basically the structure of the code is made such that it is possible to acquire new data from each run, thanks to the previous coding phase of the stm32f401re microcontroller, this was possible because the format and the data acquisition algorithm are developed in order to retrieve them in real time.
It is also possible to store and run a specific dataset for a future analysis, this allows to test a specific user case scenario and analyze the problems and possible solution without retesting in real life the same conditions.
In this case the Kalman filter already implemented inside the board is not useful anymore and the data needed are just the raw data that comes from the measurements of the accelerometer, the gyroscope and the magnetometer.
This data are then manipulated by the MATLAB builtin function **ahrsfilter**.
This function allows to combine accelerometer, gyroscope and magnetometer measurements to estimate the device orientation.

It is a similar to the Kalman filter already implemented in the stm32f401re microcontroller, but in this way it is possible to have instead of the quaternions, the rotation matrices relative to the reference frame captured at the starting point of the algorithm.
After the implementation of the ahrsfilter it is possible to obtain the accelerations for every time step computed with the directions indicated by the rotation matrices.
From this it is possible to obtain the linear accelerations, subtracting the gravity vector.
Having the linear acceleration always relative to the fixed reference frame it is possible to start the double integration process, obtaining the speed and the position in time relative to the measurements performed at each time step.
But a very important difference with respect to the python code developed on the raspberry pi is the fact that the algorithm does not run in real-time.
This is due to the necessity for the matlab filtering algorithm to have all the data vector to filter.
This technique is extremely more precise, in fact the results are really close to reality.
The need for the complete vector is necessary for the **filtfilt()** matlab filtering function.
The function process the input data performing a zero-phase digital filtering in both forward and backwards direction, meaning that after the first run of the filtering algorithm it comes backwards, obtaining an incredible results especially in starting and ending position.
The results are then plotted using the function **SixDofAnimation()** produced by xioTechnologies that grants a perfect animation of the data obtained.
The system was putted under test in different conditions, one of them and also the first was a simple movement of the board on the desk.
Since this solution worked really well compared to the python implementation discussed earlier it was then putted under stress tests to see how precise the system could be.
For standard and not too fast movements the data measured were very precise, the animation on the pc was equal to the movements done with the board, but for faster movements, like shacking the board or moving it in random directions with fast change in acceleration led to results that were not inline with the working condition.
The other conditions were running and walking with the board attached to the body.
It seems that in this working condition the constant acceleration due to the type of movements are recognized as noise and filtered by the algorithm, not taking into account the displacement that is caused by the walking or running person.
Anyway the animation shows a up and down movement, so it seems that is possible

to count the steps done by the human that is walking or running, but this method is for sure not robust or reliable in order to have precise results.
In **Figure 6.14** is represented the data retrieved from the walking test phase.
The green line represents the movement done by the IMU attached to the human and the red line the estimated position.
As said before it is noticeable that the estimation only track the oscillatory movement without taking into account any displacement due to the walk of the person.

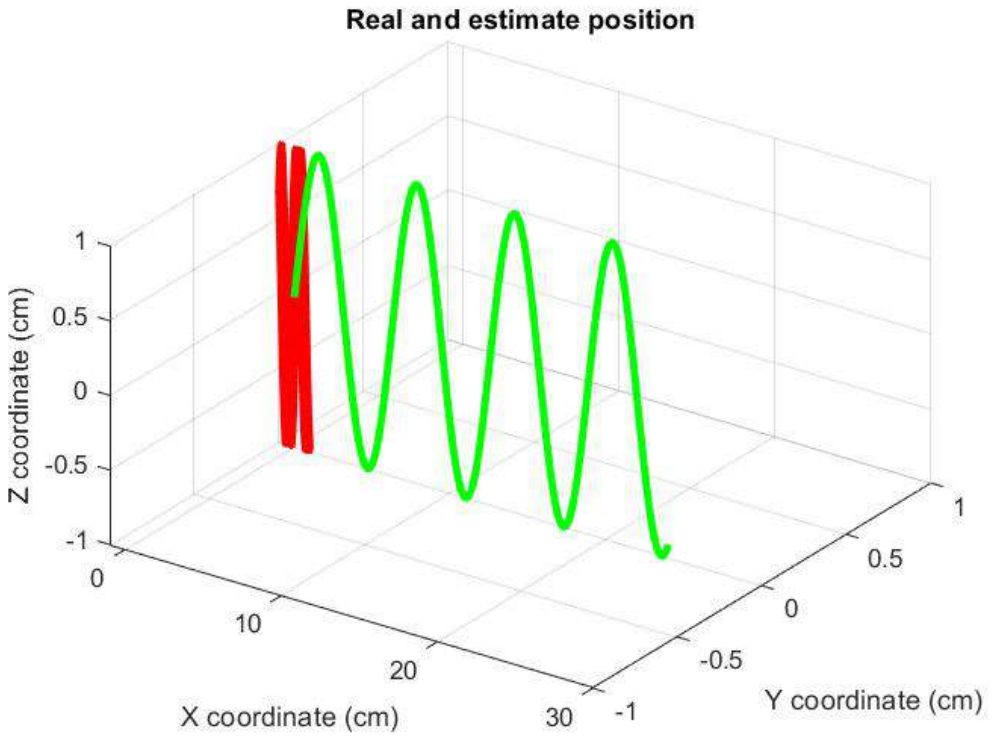

Figure 6.14: Walking motion detection

Some of the result obtained in the test phase of the IMU are illustrated from **Figure 6.15** to **Figure 6.19**.

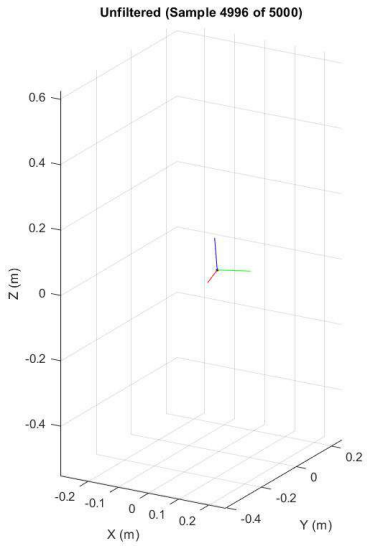

Figure 6.15: IMU's reference frame animation

In the figure above is represented the IMU's reference frame in the final position of the test phase.
This model and animation is really important because allows to observe and understand visually the estimation that is computed through the algorithm.

While in the figure below it is possible to observe the results relative to the acquisition of the raw data regarding the 3 axis of the accelerometer.
It is possible to see that the accelerometer data are corrected through a tilt compensation algorithm that allows the system to have a higher precision.

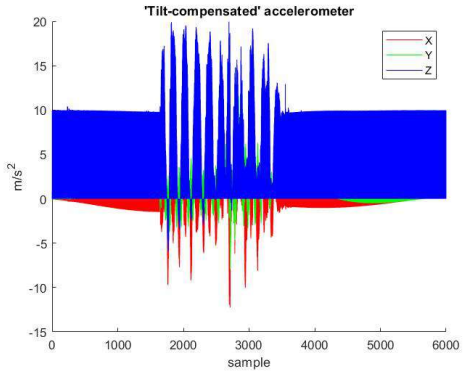

Figure 6.16: Tilt compensated accelerometer

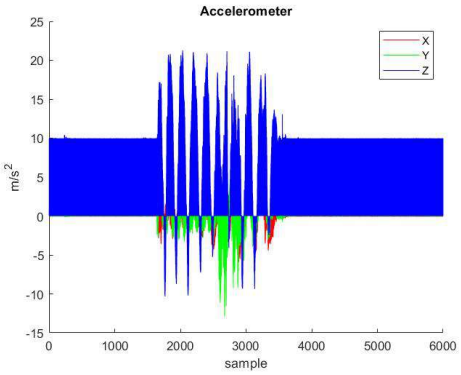

Figure 6.17: Accelerometer

In **Figure 6.18** and in **Figure 6.19** are highlighted the results that regards filtered linear position and velocity.
As it is possible to see both the velocity and the position have a transient in which they reach a peak and after a plateau, meaning that the algorithm is capable to recognize the moving state and the steady state.
In fact all the oscillation and displacements that are detected and plotted in the animation function are recognized from this particular portion of code that allows to retrieve this results.
In particular this are the data that are obtained after the double integration process, and as it is noticeable they are devoid of drift, that was the main problem of the implementation described using python and the raspberry.

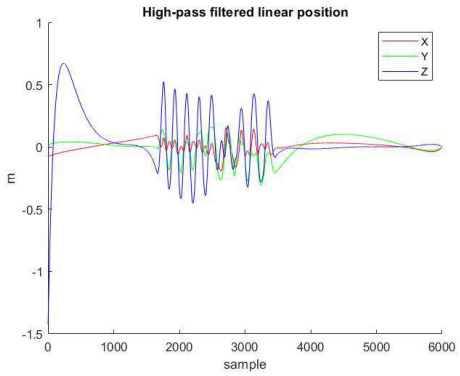

Figure 6.18: High filtered linear position

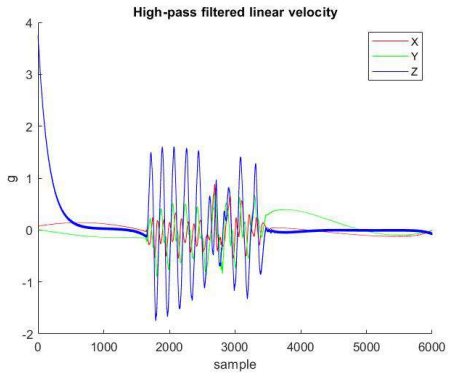

Figure 6.19: High filtered linear velocity

So the results discussed above were satisfactory and even if the system is not complete, this tests shows how it is possible to achieve a working system with the IMU. The results will be used for future work combining other sensors to correct the IMU measurements, probably using GPS and more complex algorithms or neural networks.

Chapter 7

Conclusions and future work

The objective of the thesis was to create a system able to follow a specific person, this final chapter will be used to recap and comment the results obtained during the whole development.

As described and discussed in the previous chapters, the project is possible and the results obtained are very promising, they will be the basis from which future work and projects will start.
This chapter will summarize some of the most important results obtained during the development and testing phase.

Starting from the initial configuration with the raspberry pi model 2B+ and 4 it is really interesting to analyze the results obtained with and without the NCS2 and the openvino environment.
The results obtained for the computer vision and image recognition algorithm are summarised in **Table 7.1**:

	Raspberry pi 2B+	**Raspberry pi 4**
OpenCV	<1 fps	Not tested
TensorFlow Lite	<1 fps	Not tested
Without Openvino	<1 fps	Around 4/5 fps
With Openvino	Around 3/4 fps	Around 16/20 fps

Table 7.1: Computer vision and image recognition results

The image recognition algorithm combined with tag reading or color detection are good approaches but they need a more powerful hardware both camera and raspberry should be upgraded in order to obtain satisfactory results.
At the end of the thesis it was decided to leave the image recognition algorithm for the identification of the user and use other techniques in order to accomplish this task.
In future works this decision will allow to use the camera and computer vision to detect and avoid obstacle in the scene, so the camera will be very useful for the obstacle avoidance algorithm recognizing and avoiding obstacles and people or moving objects on the scene.

The next results are obtained from the infrared and bluetooth technology used for the recognition of the user.
During the test phases both the IR and BLE approach results, were not suitable for the project, due to the problems relative to the possible objects or people that could stand between the human and the robot.
In this case the IR technology would be not enough since the light can't reach the receiver and so it is not possible to identify the user.
For the BLE this problem is almost the same, in the sense that if an object is placed between the human and the robot the radio signals would be less powerful and so the RSSI computation would not be precise, leading to an estimation of the position that is not correct.

The IR technology could anyway be implemented for other tasks such has obstacle detections, the results and study done on this technology for the thesis will be useful for future work in the obstacle avoidance project for the robot.
The BLE technology is not suitable for the triangulation of the user, for this purpose an hardware upgrade is needed with more powerful receiver, but the actual hardware is suitable for the communication algorithm with the IMU of the user, leaving the possibility to analyse and estimate the position of the person with other technologies but with the BLE technique as core for the communication.
The results obtained from this two technologies are summarised in **Table 7.2**:

	IR	**BLE**
Maximum distance	About 40/60cm	About 20/25cm

Table 7.2: Infrared and bluetooth low energy maximum distance

The last technique for the identification of the user uses the IMUs technology to estimate the position of the human.
The idea is to fuse the bluetooth communication and IMU measurements to obtain a wireless working system capable of following the target without the need of a camera.
To test if the IMU approach would work as results stated above the python project inside the raspberry was not satisfactory and a more complex and numerically stable algorithm was developed inside MATLAB using a personal computer.
The results with this technique led to a possible implementation and sensor fusion in future works, using other technologies like the GPS to correct the measurements of the IMU or neural networks designed to avoid the drift and estimate in a precise way the position.
Another idea in the future projects is to use the magnetic field of the user's phone in order to identify and track its movements, but for now the results obtained from all this technologies and tests are promising and would be used as an important base for the future projects.

Appendices

Appendix A

Openvino setup

A.1 Openvino setup tutorial

```
Please start from a fresh raspbian installation in order to avoid any issue

Updates:
sudo apt-get update && sudo apt-get upgrade

Install CMake developer tools:
sudo apt-get install build-essential cmake unzip pkg-config

Video lii needed for openvino:
sudo apt-get install libjpeg-dev libpng-dev libtiff-dev
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get install libxvidcore-dev libx264-dev

GTK:
sudo apt-get install libgtk-3-dev
sudo apt-get install libcanberra-gtk*

Numerical optimization for openCV:
sudo apt-get install libatlas-base-dev gfortran

Python3 developer tools:
sudo apt-get install python3-dev

Install Openvino:
cd ~
wget https://download.01.org/opencv/2020/openvinotoolkit/2020.1  --->
---> /l_openvino_toolkit_runtime_raspbian_p_2020.1.023.tgz
```

```
tar -xf l_openvino_toolkit_runtime_raspbian_p_2020.1.023.tgz
mv l_openvino_toolkit_runtime_raspbian_p_2020.1.023 openvino

Setup Openvino:
nano ~/.bashrc



Scroll at the and of the file and write:
# OpenVINO
source ~/openvino/bin/setupvars.sh

Save and exit

Run the bash:
source ~/.bashrc

Custom USB rules for the NCS2:
sudo usermod -a -G users "$(whoami)"

After this logout and log back in

Then set the rules:
cd ~
sh openvino/install_dependencies/install_NCS_udev_rules.sh

Install Pip:
wget https://bootstrap.pypa.io/get-pip.py
sudo python3 get-pip.py

Install Virtual environment (this is done in order to prevent errors or conflict with the other libraries or project inside the raspberry):
sudo pip install virtualenv virtualenvwrapper
sudo rm -rf ~/get-pip.py ~/.cache/pip

To finish, open:
nano ~/.bashrc

Scroll down write:
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
VIRTUALENVWRAPPER_ENV_BIN_DIR=bin

Save and exit and:
```

```
source ~/.bashrc

Create the virtual environment:
mkvirtualenv openvino -p python3




Install all the packages on the ve:
workon openvino
pip install numpy
pip install "picamera[array]"
pip install imutils

Test python installation:
python
import cv2
cv2.__version__
This should be the result '4.2.0-openvino'
exit()

If you want a fast ready to use script at startup create a bash file with inside:
#!/bin/bash
echo "Starting Python 3.7 with OpenCV-OpenVINO 4.2.0 bindings..."
source ~/openvino/bin/setupvars.sh
workon openvino

Save it and run it with:
source ~/start_openvino.sh
```

A.2 Openvino setup test

To test the **openvino** setup create a .py named openvino_test and copy the script underneath, the caffemodel and the prototxt file can be retrieved from the professor Vacca.

```
# USAGE
# python openvino_test.py --prototxt MobileNetSSD_deploy.prototxt
#--model MobileNetSSD_deploy.caffemodel

# import the necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
```

```
import imutils
import time
import cv2

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True,
  help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
  help="path to Caffe pre-trained model")
args = vars(ap.parse_args())

# initialize the list of class labels MobileNet SSD was trained to
# detect, then generate a set of bounding box colors for each class
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
  "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
  "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
  "sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# specify the target device as the Myriad processor on the NCS
net.setPreferableTarget(cv2.dnn.DNN_TARGET_MYRIAD)

# initialize the video stream, allow the cammera sensor to warmup,
# and initialize the FPS counter
print("[INFO] starting video stream...")
vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)
fps = FPS().start()

# loop over the frames from the video stream
while True:
  # grab the frame from the threaded video stream and resize it
  # to have a maximum width of predetermined pixels
  frame = vs.read()
  frame = imutils.resize(frame, width=800)
```

```
50
51  # grab the frame dimensions and convert it to a blob
52  (h, w) = frame.shape[:2]
53  blob = cv2.dnn.blobFromImage(frame, 0.007843, (300,
     300), 127.5)
54
55  # pass the blob through the network and obtain the
     detections and
56  # predictions
57  net.setInput(blob)
58  detections = net.forward()
59
60  # loop over the detections
61  for i in np.arange(0, detections.shape[2]):
62    # extract the confidence (i.e., probability)
     associated with
63    # the prediction
64    confidence = detections[0, 0, i, 2]
65
66    # filter out weak detections by ensuring the `
     confidence` is
67    # greater than the minimum confidence
68    if confidence > args["confidence"]:
69      # extract the index of the class label from the
70      # `detections`, then compute the (x, y)-coordinates
      of
71      # the bounding box for the object
72      idx = int(detections[0, 0, i, 1])
73      box = detections[0, 0, i, 3:7] * np.array([w, h, w,
     h])
74      (startX, startY, endX, endY) = box.astype("int")
75
76      # draw the prediction on the frame
77      label = "{}: {:.2f}%".format(CLASSES[idx],
78        confidence * 100)
79      cv2.rectangle(frame, (startX, startY), (endX, endY)
     ,
80        COLORS[idx], 2)
81      y = startY - 15 if startY - 15 > 15 else startY +
     15
82      cv2.putText(frame, label, (startX, y),
83        cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
84
85  # show the output frame
86  cv2.imshow("Frame", frame)
87  key = cv2.waitKey(1) & 0xFF
88
89  # if the `q` key was pressed, break from the loop
90  if key == ord("q"):
91    break
```

```
92
93      # update the FPS counter
94      fps.update()
95
96  # stop the timer and display FPS information
97  fps.stop()
98  print("[INFO] elasped time: {:.2f}".format(fps.elapsed())
        )
99  print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
100
101 # do a bit of cleanup
102 cv2.destroyAllWindows()
103 vs.stop()
```

Appendix B

PAN-TILT HAT codes

B.1 PCA9685 control library

```
#!/usr/bin/python

import time
import math
import smbus2 as smbus

# ============================================================
# Raspi PCA9685 16-Channel PWM Servo Driver
# ============================================================

class PCA9685:

  # Registers/etc.
  __SUBADR1            = 0x02
  __SUBADR2            = 0x03
  __SUBADR3            = 0x04
  __MODE1              = 0x00
  __MODE2              = 0x01
  __PRESCALE           = 0xFE
  __LED0_ON_L          = 0x06
  __LED0_ON_H          = 0x07
  __LED0_OFF_L         = 0x08
  __LED0_OFF_H         = 0x09
  __ALLLED_ON_L        = 0xFA
  __ALLLED_ON_H        = 0xFB
  __ALLLED_OFF_L       = 0xFC
  __ALLLED_OFF_H       = 0xFD
```

```
33  def __init__(self, address=0x40, debug=False):
34    self.bus = smbus.SMBus(1)
35    self.address = address
36    self.debug = debug
37    if (self.debug):
38      print("Reseting PCA9685")
39    self.write(self.__MODE1, 0x00)
40
41  def write(self, reg, value):
42    "Writes an 8-bit value to the specified register/
    address"
43    self.bus.write_byte_data(self.address, reg, value)
44    if (self.debug):
45      print("I2C: Write 0x%02X to register 0x%02X" % (
    value, reg))
46
47  def read(self, reg):
48    "Read an unsigned byte from the I2C device"
49    result = self.bus.read_byte_data(self.address, reg)
50    if (self.debug):
51      print("I2C: Device 0x%02X returned 0x%02X from reg
    0x%02X" % (self.address, result & 0xFF, reg))
52    return result
53
54  def setPWMFreq(self, freq):
55    "Sets the PWM frequency"
56    prescaleval = 25000000.0    # 25MHz
57    prescaleval /= 4096.0       # 12-bit
58    prescaleval /= float(freq)
59    prescaleval -= 1.0
60    if (self.debug):
61      print("Setting PWM frequency to %d Hz" % freq)
62      print("Estimated pre-scale: %d" % prescaleval)
63    prescale = math.floor(prescaleval + 0.5)
64    if (self.debug):
65      print("Final pre-scale: %d" % prescale)
66
67    oldmode = self.read(self.__MODE1);
68    newmode = (oldmode & 0x7F) | 0x10        # sleep
69    self.write(self.__MODE1, newmode)        # go to
    sleep
70    self.write(self.__PRESCALE, int(math.floor(prescale))
    )
71    self.write(self.__MODE1, oldmode)
72    time.sleep(0.005)
73    self.write(self.__MODE1, oldmode | 0x80)
74    self.write(self.__MODE2, 0x04)
75
76
77
```

```
78
79    def setPWM(self, channel, on, off):
80        "Sets a single PWM channel"
81        self.write(self.__LED0_ON_L+4*channel, on & 0xFF)
82        self.write(self.__LED0_ON_H+4*channel, on >> 8)
83        self.write(self.__LED0_OFF_L+4*channel, off & 0xFF)
84        self.write(self.__LED0_OFF_H+4*channel, off >> 8)
85        if (self.debug):
86            print("channel: %d  LED_ON: %d LED_OFF: %d" % (
      channel,on,off))
87
88    def setServoPulse(self, channel, pulse):
89        "Sets the Servo Pulse,The PWM frequency must be 50HZ"
90        pulse = pulse*4096/20000        #PWM frequency is 50
      HZ,the period is 20000us
91        self.setPWM(channel, 0, int(pulse))
92
93    def setRotationAngle(self, channel, Angle):
94        if(Angle >= 0 and Angle <= 180):
95            temp = Angle * (2000 / 180) + 501
96            self.setServoPulse(channel, temp)
97        else:
98            print("Angle out of range")
99
100   def start_PCA9685(self):
101       self.write(self.__MODE2, 0x04)
102       #Just restore the stopped state that should be set
      for exit_PCA9685
103
104   def exit_PCA9685(self):
105       self.write(self.__MODE2, 0x00)#Please use
      initialization or __MODE2 =0x04
```

B.2 PAN-TILT HAT test code

```
1  import time
2  import RPi.GPIO as GPIO
3  from PCA9685 import PCA9685
4
5  try:
6      print ("This is an PCA9685 routine")
7      pwm = PCA9685()
8      pwm.setPWMFreq(50)
9      pwm.setRotationAngle(1, 90)
10
11     while True:
12
13         for i in range(0,180,1):
```

```
14              pwm.setRotationAngle(1, i)
15              time.sleep(0.01)
16
17          for i in range(180,0,-1):
18              pwm.setRotationAngle(1, i)
19              time.sleep(0.01)
20
21          for i in range(90,180,1):
22              pwm.setRotationAngle(0, i)
23              time.sleep(0.01)
24
25          for i in range(180,90,-1):
26              pwm.setRotationAngle(0, i)
27              time.sleep(0.01)
28
29  except:
30      pwm.exit_PCA9685()
31      print ("\nProgram end")
32      exit()
```

Appendix C

Camera control algorithm

```
if CLASSES[idx]=="person":
  label = "{}: {:.2f}%".format(CLASSES[idx],
    confidence * 100)
  midX=int((startX+endX)/2)
  midY=int((startY+endY)/2)
  cv2.line(frame,(midX,midY-30),(midX,midY+30),COLORS[idx
   ],2)
  cv2.line(frame,(midX-30,midY),(midX+30,midY),COLORS[idx
   ],2)
  y = startY - 15 if startY - 15 > 15 else startY + 15
  cv2.putText(frame, label, (startX, y),
    cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
  if (startX+endX)/2<220:
    if pan<180:
      pan=pan+10
    pwm.setRotationAngle(1, pan)
  elif (startX+endX)/2>500:
    if pan>0:
      pan=pan-10
    pwm.setRotationAngle(1, pan)
  if (startY+endY)/2<190:
    if tilt>110:
      tilt=tilt-5
    pwm.setRotationAngle(0, tilt)
  elif (startY+endY)/2>390:
    if tilt<180:
      tilt=tilt+5
    pwm.setRotationAngle(0, tilt)
  if startY<70:
    if tilt>110:
      tilt=tilt-5
    pwm.setRotationAngle(0, tilt)
  break
```

Appendix D

Robot's position control

```
class motors(threading.Thread):

  def __init__(self,threadID):
    threading.Thread.__init__(self,daemon=True)
    self.threadID=threadID

  def run(self):
    global startX
    global endX
    global startY
    global endY
    global pan
    global tilt
    global motor_start

    pwm = PCA9685()
    pwm.setPWMFreq(50)

    GPIO.setmode(GPIO.BCM)

    GPIO.setup(21, GPIO.OUT) #enable
    GPIO.setup(20, GPIO.OUT)
    GPIO.setup(16, GPIO.OUT)

    GPIO.setup(26, GPIO.OUT) #enable
    GPIO.setup(19, GPIO.OUT)
    GPIO.setup(13, GPIO.OUT)

    forward = (20,13)
    GPIO.output(forward, GPIO.LOW)
    f_pwm=GPIO.PWM(forward,50)

    backwards = (16,19)
    GPIO.output(backwards, GPIO.LOW)
```

```
35      b_pwm=GPIO.PWM(backwards,50)
36
37      rigth = (13,16)
38      GPIO.output(rigth, GPIO.LOW)
39      r_pwm=GPIO.PWM(right,50)
40
41      left = (20,19)
42      GPIO.output(left, GPIO.LOW)
43      l_pwm=GPIO.PWM(left,50)
44
45      while True:
46        if motor_start:
47          if pan<80:
48            #GPIO.output(left, GPIO.HIGH)
49            l_pwm.start(40)
50            sleep(0.3)
51            #GPIO.output(left, GPIO.LOW)
52            l_pwm.stop()
53            pan=80
54            pwm.setRotationAngle(1, pan)
55          if pan>100:
56            #GPIO.output(rigth, GPIO.HIGH)
57            r_pwm.start(40)
58            sleep(0.3)
59            #GPIO.output(rigth, GPIO.LOW)
60            l_pwm.stop()
61            pan=100
62            pwm.setRotationAngle(1, pan)
63          while (endX-startX)*(endY-startY)<100000:
64            #GPIO.output(forward, GPIO.HIGH)
65            f_pwm.start(40)
66          #GPIO.output(forward, GPIO.LOW)
67          f_pwm.stop()
68
69          while (endX-startX)*(endY-startY)>150000:
70            #GPIO.output(backwards, GPIO.HIGH)
71            b_pwm.start(40)
72          #GPIO.output(backwards, GPIO.LOW)
73          b_pwm.stop()
74
75        else:
76          GPIO.output(forward, GPIO.LOW)
77          GPIO.output(backwards, GPIO.LOW)
```

Appendix E

Identification techniques

In this appendix are present all the codes related to the identification techniques discussed in the chapters above.

E.1 Color detection identification code

```
frameHeight = 480
cap = cv2.VideoCapture(1)
cap.set(3, frameWidth)
cap.set(4, frameHeight)


def empty(a):
    pass

cv2.namedWindow("HSV")
cv2.resizeWindow("HSV", 640, 240)
cv2.createTrackbar("HUE Min", "HSV", 0, 179, empty)
cv2.createTrackbar("HUE Max", "HSV", 179, 179, empty)
cv2.createTrackbar("SAT Min", "HSV", 0, 255, empty)
cv2.createTrackbar("SAT Max", "HSV", 255, 255, empty)
cv2.createTrackbar("VALUE Min", "HSV", 0, 255, empty)
cv2.createTrackbar("VALUE Max", "HSV", 255, 255, empty)


while True:

    success, img = cap.read()
    imgHsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    h_min = cv2.getTrackbarPos("HUE Min", "HSV")
    h_max = cv2.getTrackbarPos("HUE Max", "HSV")
    s_min = cv2.getTrackbarPos("SAT Min", "HSV")
    s_max = cv2.getTrackbarPos("SAT Max", "HSV")
```

```
    v_min = cv2.getTrackbarPos("VALUE Min", "HSV")
    v_max = cv2.getTrackbarPos("VALUE Max", "HSV")
    print(h_min)

    lower = np.array([h_min, s_min, v_min])
    upper = np.array([h_max, s_max, v_max])
    mask = cv2.inRange(imgHsv, lower, upper)
    result = cv2.bitwise_and(img, img, mask=mask)

    mask = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)
    hStack = np.hstack([img, mask, result])
    cv2.imshow('Horizontal Stacking', hStack)
    if cv2.waitKey(1) && 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

E.2 Infrared detection code

```
import RPi.GPIO as GPIO
from time import sleep      # needed for the delay

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.IN)     # IR receiver port
# Define a threaded callback function to run in another
#thread when events are detected
def my_callback(channel):
    if GPIO.input(18):      # if port 25 == 1
        print ("Rising edge detected on 18")
    else:                   # if port 25 != 1
        print ("Falling edge detected on 18")
# when a changing edge is detected on port 25, regardless
    of whatever
# else is happening in the program, the function
   my_callback will be run
GPIO.add_event_detect(18, GPIO.BOTH, callback=my_callback
   )

print ("Program will finish after 30 seconds or if you
   press CTRL+C\n"  )
try:
    sleep(30)          # wait 30 seconds
    print ("Finished" )
finally:
    GPIO.cleanup()     # clean up GPIO ports
```

E.3 Infrared and computer vision fusion

```
# USAGE
# python openvino_real_time_object_detection.py --
   prototxt MobileNetSSD_deploy.prototxt --model
   MobileNetSSD_deploy.caffemodel

# import the necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import time
import cv2

import threading
import RPi.GPIO as GPIO
from time import sleep

from PCA9685 import PCA9685

GPIO.setmode(GPIO.BCM)     # set up BCM GPIO numbering
GPIO.setup(18, GPIO.IN)

def my_callback(channel):
  global user
  global counter

  user=True
  counter=0

class control(threading.Thread):

  def __init__(self,threadID):
    threading.Thread.__init__(self,daemon=True)
    self.threadID=threadID

  def run(self):
    global user
    global pan
    global counter
    global flag
    pwm = PCA9685()
    pwm.setPWMFreq(50)
    while True:
      if flag:
        while True:
          counter+=1
          if counter>10:
```

```
47              if user==True:
48                pwm.setRotationAngle(1, 0)
49                pan=0
50
51              user=False
52
53              if pan<180:
54                pan+=10
55                pwm.setRotationAngle(1, pan)
56              else:
57                pwm.setRotationAngle(1, 0)
58                pan=0
59            sleep(0.3)
60
61
62
63
64 if __name__=="__main__":
65
66   global pan
67   pan=90
68   global tilt
69   tilt=165
70   global user
71   user=True
72   global counter
73   counter=0
74   global flag
75   flag=False
76
77   GPIO.add_event_detect(18, GPIO.BOTH, callback=
      my_callback)  #IR DETECTIONS
78
79   thread=control(1)
80   thread.start()
81
82   pwm = PCA9685()
83   pwm.setPWMFreq(50)
84   pwm.setRotationAngle(1, pan)
85   pwm.setRotationAngle(0, tilt)
86
87   # construct the argument parse and parse the arguments
88   ap = argparse.ArgumentParser()
89   ap.add_argument("-p", "--prototxt", required=True,
90     help="path to Caffe 'deploy' prototxt file")
91   ap.add_argument("-m", "--model", required=True,
92     help="path to Caffe pre-trained model")
93   ap.add_argument("-c", "--confidence", type=float,
      default=0.5,
94     help="minimum probability to filter weak detections")
```

```
95  ap.add_argument("-u", "--movidius", type=bool, default
     =0,
96    help="boolean indicating if the Movidius should be
     used")
97  args = vars(ap.parse_args())
98
99  # initialize the list of class labels MobileNet SSD was
      trained to
100 # detect, then generate a set of bounding box colors
     for each class
101 CLASSES = ["background", "aeroplane", "bicycle", "bird"
     , "boat",
102   "bottle", "bus", "car", "cat", "chair", "cow", "
     diningtable",
103   "dog", "horse", "motorbike", "person", "pottedplant",
      "sheep",
104   "sofa", "train", "tvmonitor"]
105 COLORS = np.random.uniform(0, 255, size=(len(CLASSES),
     3))
106
107 # load our serialized model from disk
108 print("[INFO] loading model...")
109 net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["
     model"])
110
111 # specify the target device as the Myriad processor on
     the NCS
112 net.setPreferableTarget(cv2.dnn.DNN_TARGET_MYRIAD)
113
114 # initialize the video stream, allow the cammera sensor
      to warmup,
115 # and initialize the FPS counter
116 print("[INFO] starting video stream...")
117 vs = VideoStream(usePiCamera=True).start()
118 time.sleep(2.0)
119 fps = FPS().start()
120
121
122
123 # loop over the frames from the video stream
124 while True:
125   flag=True
126   # grab the frame from the threaded video stream and
     resize it
127   # to have a maximum width of 400 pixels
128   frame = vs.read()
129   frame = imutils.resize(frame, width=720 , height=480)
130
131   # grab the frame dimensions and convert it to a blob
132   (h, w) = frame.shape[:2]
```

```
133     blob = cv2.dnn.blobFromImage(frame, 0.007843, (300,
    300), 127.5)
134
135     # pass the blob through the network and obtain the
    detections and
136     # predictions
137     net.setInput(blob)
138     detections = net.forward()
139
140     if user==True:
141       # loop over the detections
142       for i in np.arange(0, detections.shape[2]):
143         # extract the confidence (i.e., probability)
    associated with
144         # the prediction
145         confidence = detections[0, 0, i, 2]
146
147         # filter out weak detections by ensuring the `
    confidence` is
148         # greater than the minimum confidence
149         if confidence > args["confidence"]:
150           # extract the index of the class label from the
151           # `detections`, then compute the (x, y)-
    coordinates of
152           # the bounding box for the object
153           idx = int(detections[0, 0, i, 1])
154           box = detections[0, 0, i, 3:7] * np.array([w, h
    , w, h])
155           (startX, startY, endX, endY) = box.astype("int"
    )
156
157           # draw the prediction on the frame
158           if CLASSES[idx]=="person":
159             label = "{}: {:.2f}%".format(CLASSES[idx],
160               confidence * 100)
161             #cv2.rectangle(frame, (startX, startY), (endX
    , endY),
162             # COLORS[idx], 2)
163             midX=int((startX+endX)/2)
164             midY=int((startY+endY)/2)
165             cv2.line(frame,(midX,midY-30),(midX,midY+30),
    COLORS[idx],2)
166             cv2.line(frame,(midX-30,midY),(midX+30,midY),
    COLORS[idx],2)
167             y = startY - 15 if startY - 15 > 15 else
    startY + 15
168             cv2.putText(frame, label, (startX, y),
169               cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx],
     2)
170             if (startX+endX)/2<220:
```

```
171                 if pan<180:
172                   pan=pan+10
173                 pwm.setRotationAngle(1, pan)
174                 #cv2.putText(frame,"TURN LEFT",(50,240),cv2
    .FONT_HERSHEY_SIMPLEX,4,(255,255,255))
175               elif (startX+endX)/2>500:
176                 if pan>0:
177                   pan=pan-10
178                 pwm.setRotationAngle(1, pan)
179                 #cv2.putText(frame,"TURN RIGHT",(50,240),
    cv2.FONT_HERSHEY_SIMPLEX,4,(255,255,255))
180               if (startY+endY)/2<120:
181                 if tilt>90:
182                   tilt=tilt-10
183                 pwm.setRotationAngle(0, tilt)
184               elif (startY+endY)/2>360:
185                 if tilt<180:
186                   tilt=tilt+10
187                 pwm.setRotationAngle(0, tilt)
188               break
189
190     # show the output frame
191     cv2.imshow("Frame", frame)
192     key = cv2.waitKey(1) & 0xFF
193
194     # if the `q` key was pressed, break from the loop
195     if key == ord("q"):
196       break
197
198     # update the FPS counter
199     fps.update()
200
201   # stop the timer and display FPS information
202   fps.stop()
203   pwm.exit_PCA9685()
204   print("[INFO] elasped time: {:.2f}".format(fps.elapsed
    ()))
205   print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
206
207   # do a bit of cleanup
208   cv2.destroyAllWindows()
209   vs.stop()
```

E.4 STM32 BLE communication code

```
1 /* Includes ------------------------------------*/
2 #include "main.h"
3
```

```
/* Private includes
   ----------------------------------------------*/
/* USER CODE BEGIN Includes */

#include <stdlib.h>
#include <string.h>
#include <stdio.h>


#include "bluenrg1_aci.h"
#include "bluenrg1_hci_le.h"
#include "bluenrg1_events.h"
#include "hci_tl.h"
#include "bluenrg_utils.h"
//#include "osal.h"

/* USER CODE END Includes */

/* Private typedef
   -----------------------------------------------*/
/* USER CODE BEGIN PTD */

void APP_UserEvtRx(void *pData);
void hci_init(void(* UserEvtRx)(void* pData), void* pConf
   );
tBleStatus Add_SWServW2ST_Service(void);
tBleStatus Add_HWServW2ST_Service(void);

/* USER CODE END PTD */

/* Private define
   ------------------------------------------------*/
/* USER CODE BEGIN PD */

uint8_t bdaddr_cb[6];
uint8_t user_detected=0;
uint8_t user_check=0;

/* USER CODE END PD */

/* Private macro
   -----------------------------------------------*/
/* USER CODE BEGIN PM */

__IO uint8_t set_connectable = 1;
__IO uint16_t connection_handle = 0;
__IO uint8_t  notification_enabled = FALSE;
__IO uint32_t connected = FALSE;

/* Hardware Characteristics Service */
```

```
49
50 #define COPY_UUID_128(uuid_struct, uuid_15, uuid_14,
   uuid_13, uuid_12, uuid_11, uuid_10, uuid_9, uuid_8,
   uuid_7, uuid_6, uuid_5, uuid_4, uuid_3, uuid_2, uuid_1
   , uuid_0) \
51 do {\
52     uuid_struct[0] = uuid_0; uuid_struct[1] = uuid_1;
   uuid_struct[2] = uuid_2; uuid_struct[3] = uuid_3; \
53          uuid_struct[4] = uuid_4; uuid_struct[5] = uuid_5;
    uuid_struct[6] = uuid_6; uuid_struct[7] = uuid_7; \
54               uuid_struct[8] = uuid_8; uuid_struct[9] =
   uuid_9; uuid_struct[10] = uuid_10; uuid_struct[11] =
   uuid_11; \
55                    uuid_struct[12] = uuid_12; uuid_struct
   [13] = uuid_13; uuid_struct[14] = uuid_14; uuid_struct
   [15] = uuid_15; \
56 }while(0)
57
58 #define COPY_HW_SENS_W2ST_SERVICE_UUID(uuid_struct)
   COPY_UUID_128(uuid_struct,0x00,0x00,0x00,0x00,0x00,0
   x01,0x11,0xe1,0x9a,0xb4,0x00,0x02,0xa5,0xd5,0xc5,0x1b)
59 #define COPY_ENVIRONMENTAL_W2ST_CHAR_UUID(uuid_struct)
   COPY_UUID_128(uuid_struct,0x00,0x00,0x00,0x00,0x00,0
   x01,0x11,0xe1,0xac,0x36,0x00,0x02,0xa5,0xd5,0xc5,0x1b)
60 #define COPY_ACC_GYRO_MAG_W2ST_CHAR_UUID(uuid_struct)
   COPY_UUID_128(uuid_struct,0x00,0xE0,0x00,0x00,0x00,0
   x01,0x11,0xe1,0xac,0x36,0x00,0x02,0xa5,0xd5,0xc5,0x1b)
61 /* Software Characteristics Service */
62 #define COPY_SW_SENS_W2ST_SERVICE_UUID(uuid_struct)
   COPY_UUID_128(uuid_struct,0x00,0x00,0x00,0x00,0x00,0
   x02,0x11,0xe1,0x9a,0xb4,0x00,0x02,0xa5,0xd5,0xc5,0x1b)
63 #define COPY_QUATERNIONS_W2ST_CHAR_UUID(uuid_struct)
   COPY_UUID_128(uuid_struct,0x00,0x00,0x01,0x00,0x00,0
   x01,0x11,0xe1,0xac,0x36,0x00,0x02,0xa5,0xd5,0xc5,0x1b)
64
65 /* USER CODE END PM */
66
67 /* Private variables
   -----------------------------------------------*/
68 UART_HandleTypeDef huart2;
69
70 /* USER CODE BEGIN PV */
71
72 #define SENSOR_DEMO_NAME    'B','l','u','e','N','R','G'
73 #define BDADDR_SIZE         6
74
75 /* Hardware Characteristics Service */
76
77 /* USER CODE END PV */
78
```

```
79 /* Private function prototypes
   -----------------------------------------------*/
80 void SystemClock_Config(void);
81 static void MX_GPIO_Init(void);
82 void SystemClock_Config(void)
83 {
84   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
85   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
86
87   /** Configure the main internal regulator output
      voltage
88   */
89   __HAL_RCC_PWR_CLK_ENABLE();
90   __HAL_PWR_VOLTAGESCALING_CONFIG(
      PWR_REGULATOR_VOLTAGE_SCALE2);
91   /** Initializes the RCC Oscillators according to the
      specified parameters
92   * in the RCC_OscInitTypeDef structure.
93   */
94   RCC_OscInitStruct.OscillatorType =
      RCC_OSCILLATORTYPE_HSI;
95   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
96   RCC_OscInitStruct.HSICalibrationValue =
      RCC_HSICALIBRATION_DEFAULT;
97   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
98   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
99   RCC_OscInitStruct.PLL.PLLM = 8;
100  RCC_OscInitStruct.PLL.PLLN = 64;
101  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
102  RCC_OscInitStruct.PLL.PLLQ = 7;
103  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
104  {
105    Error_Handler();
106  }
107  /** Initializes the CPU, AHB and APB buses clocks
108  */
109  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|
      RCC_CLOCKTYPE_SYSCLK
110                                |RCC_CLOCKTYPE_PCLK1|
      RCC_CLOCKTYPE_PCLK2;
111  RCC_ClkInitStruct.SYSCLKSource =
      RCC_SYSCLKSOURCE_PLLCLK;
112  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
113  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
114  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
115
116  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
      FLASH_LATENCY_2) != HAL_OK)
117  {
118    Error_Handler();
```

```
119   }
120 }
121
122 static void MX_USART2_UART_Init(void);
123 /* USER CODE BEGIN PFP */
124
125 /* USER CODE END PFP */
126
127 /* Private user code
    ---------------------------------------*/
128 /* USER CODE BEGIN 0 */
129
130 /* USER CODE END 0 */
131
132 /**
133   * @brief  The application entry point.
134   * @retval int
135   */
136 int main(void)
137 {
138   /* USER CODE BEGIN 1 */
139
140   /* USER CODE END 1 */
141
142   /* MCU Configuration
    ----------------------------------------*/
143
144   /* Reset of all peripherals, Initializes the Flash
     interface and the Systick. */
145   HAL_Init();
146
147   /* USER CODE BEGIN Init */
148
149   /* USER CODE END Init */
150
151   /* Configure the system clock */
152   SystemClock_Config();
153
154   /* USER CODE BEGIN SysInit */
155
156   /* USER CODE END SysInit */
157
158   /* Initialize all configured peripherals */
159   MX_GPIO_Init();
160   MX_USART2_UART_Init();
161   /* USER CODE BEGIN 2 */
162
163   hci_init(APP_UserEvtRx, NULL);
164
165     uint8_t ret;
```

```
166     uint16_t service_handle, dev_name_char_handle,
    appearance_char_handle;
167     uint8_t device_name[] = {SENSOR_DEMO_NAME};
168     uint8_t  hwVersion;
169     uint16_t fwVersion;
170     uint8_t  bdaddr_len_out;
171     uint8_t  config_data_stored_static_random_address = 0
    x80; /* Offset of the static random address stored in
    NVM */
172
173     uint8_t bdaddr[BDADDR_SIZE];
174
175     /* Sw reset of the device */
176     hci_reset();
177     /**
178      *  To support both the BlueNRG-2 and the BlueNRG-2N
    a minimum delay of 2000ms is required at device boot
179      */
180     HAL_Delay(2000);
181
182     /* get the BlueNRG HW and FW versions */
183     getBlueNRGVersion(&hwVersion, &fwVersion);
184
185     PRINT_DBG("HWver %d\nFWver %d\r\n", hwVersion,
    fwVersion);
186
187     ret = aci_hal_read_config_data(
    config_data_stored_static_random_address,
188                                    &bdaddr_len_out,
    bdaddr);
189
190     if (ret) {
191       PRINT_DBG("Read Static Random address failed.\r\n")
    ;
192     }
193
194     if ((bdaddr[5] & 0xC0) != 0xC0) {
195       PRINT_DBG("Static Random address not well formed.\r
    \n");
196       while(1);
197     }
198
199     /* Set the TX power -2 dBm */
200     ret = aci_hal_set_tx_power_level(1, 4);
201     if (ret != BLE_STATUS_SUCCESS)
202     {
203       PRINT_DBG("Error in aci_hal_set_tx_power_level() 0x
    %04x\r\n", ret);
204     }
205     else
```

```
206     {
207       PRINT_DBG("aci_hal_set_tx_power_level() --> SUCCESS
    \r\n");
208     }
209 
210     /* GATT Init */
211     ret = aci_gatt_init();
212     if (ret != BLE_STATUS_SUCCESS)
213     {
214       PRINT_DBG("aci_gatt_init() failed: 0x%02x\r\n", ret
    );
215     }
216     else
217     {
218       PRINT_DBG("aci_gatt_init() --> SUCCESS\r\n");
219     }
220 
221     /* GAP Init */
222     ret = aci_gap_init(GAP_CENTRAL_ROLE, 0, 0x07, &
    service_handle, &dev_name_char_handle,
223                        &appearance_char_handle);
224     if (ret != BLE_STATUS_SUCCESS)
225     {
226       PRINT_DBG("aci_gap_init() failed: 0x%02x\r\n", ret)
    ;
227     }
228     else
229     {
230       PRINT_DBG("aci_gap_init() --> SUCCESS\r\n");
231     }
232 
233     /* Update device name */
234     ret = aci_gatt_update_char_value(service_handle,
    dev_name_char_handle, 0, sizeof(device_name),
235                                      device_name);
236     if(ret != BLE_STATUS_SUCCESS)
237     {
238       PRINT_DBG("aci_gatt_update_char_value() failed: 0x
    %02x\r\n", ret);
239     }
240     else
241     {
242       PRINT_DBG("aci_gatt_update_char_value() --> SUCCESS
    \r\n");
243     }
244 
245     /* BLE Security v4.2 is supported: BLE stack FW
    version >= 2.x (new API prototype) */
246     ret = aci_gap_set_authentication_requirement(BONDING,
```

```
247
   MITM_PROTECTION_REQUIRED,
248
   SC_IS_SUPPORTED,
249
   KEYPRESS_IS_NOT_SUPPORTED,
                                                          7,
250
                                                          16,
251
252
   USE_FIXED_PIN_FOR_PAIRING,
                                                          123456,
253
                                                          0x00);
254
    if(ret != BLE_STATUS_SUCCESS)
255
    {
256
      PRINT_DBG("aci_gap_set_authentication_requirement()
257
   failed: 0x%02x\r\n", ret);
    }
258
    else
259
    {
260
      PRINT_DBG("aci_gap_set_authentication_requirement()
261
    --> SUCCESS\r\n");
    }
262
263
    PRINT_DBG("BLE Stack Initialized with SUCCESS\r\n");
264
265
266
  /* USER CODE END 2 */
267
268
  /* Infinite loop */
269
  /* USER CODE BEGIN WHILE */
270
271
    //uint8_t myTxData[13]="Hello world\r\n";
272
          //HAL_UART_Transmit(&huart2,myTxData,13,10);
273
  while (1)
274
  {
275
    hci_user_evt_proc();  //NEEDED TO READ EVENT LIST AND
276
    ACTIVATE THEM
    ret = aci_gap_start_general_discovery_proc(0x4000, 0
277
   x4000,0x00,0x01);
        //uint8_t myTxData[13]="Hello world\r\n";
278
        if (ret != BLE_STATUS_SUCCESS)
279
          {
280
              PRINT_DBG("Failure.\r\n")
281
            char buffer[100];
282
             sprintf(buffer,"%x\r\n",ret);
283
             HAL_UART_Transmit(&huart2,(uint8_t*)buffer,
284
   strlen(buffer),10);
          }
285
286
      //HAL_UART_Transmit(&huart2,myTxData,13,10);
287
```

```
288           HAL_Delay(100);
289         ret=aci_gap_terminate_gap_proc(0x02);
290         if (ret != BLE_STATUS_SUCCESS)
291                   {
292                       PRINT_DBG("Failure.\r\n")
293                     char buffer[100];
294                      sprintf(buffer,"%x\r\n",ret);
295                      HAL_UART_Transmit(&huart2,(uint8_t*)
    buffer,strlen(buffer),10);
296                   }
297       /* USER CODE END WHILE */
298
299
300       /* USER CODE BEGIN 3 */
301     }
302     /* USER CODE END 3 */
303 }
304
305 /**
306     * @brief System Clock Configuration
307     * @retval None
308     */
309 /**
310     * @brief USART2 Initialization Function
311     * @param None
312     * @retval None
313     */
314 static void MX_USART2_UART_Init(void)
315 {
316
317     /* USER CODE BEGIN USART2_Init 0 */
318
319     /* USER CODE END USART2_Init 0 */
320
321     /* USER CODE BEGIN USART2_Init 1 */
322
323     /* USER CODE END USART2_Init 1 */
324     huart2.Instance = USART2;
325     huart2.Init.BaudRate = 115200;
326     huart2.Init.WordLength = UART_WORDLENGTH_8B;
327     huart2.Init.StopBits = UART_STOPBITS_1;
328     huart2.Init.Parity = UART_PARITY_NONE;
329     huart2.Init.Mode = UART_MODE_TX_RX;
330     huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
331     huart2.Init.OverSampling = UART_OVERSAMPLING_16;
332     if (HAL_UART_Init(&huart2) != HAL_OK)
333     {
334       Error_Handler();
335     }
336     /* USER CODE BEGIN USART2_Init 2 */
```

```
337
338   /* USER CODE END USART2_Init 2 */
339
340 }
341
342 /**
343   * @brief GPIO Initialization Function
344   * @param None
345   * @retval None
346   */
347 static void MX_GPIO_Init(void)
348 {
349   GPIO_InitTypeDef GPIO_InitStruct = {0};
350
351   /* GPIO Ports Clock Enable */
352   __HAL_RCC_GPIOC_CLK_ENABLE();
353   __HAL_RCC_GPIOH_CLK_ENABLE();
354   __HAL_RCC_GPIOA_CLK_ENABLE();
355   __HAL_RCC_GPIOB_CLK_ENABLE();
356
357   /*Configure GPIO pin Output Level */
358   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1|LD2_Pin|GPIO_PIN_8,
      GPIO_PIN_RESET);
359
360   /*Configure GPIO pin : B1_Pin */
361   GPIO_InitStruct.Pin = B1_Pin;
362   GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
363   GPIO_InitStruct.Pull = GPIO_NOPULL;
364   HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
365
366   /*Configure GPIO pin : PA0 */
367   GPIO_InitStruct.Pin = GPIO_PIN_0;
368   GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
369   GPIO_InitStruct.Pull = GPIO_NOPULL;
370   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
371
372   /*Configure GPIO pins : PA1 LD2_Pin PA8 */
373   GPIO_InitStruct.Pin = GPIO_PIN_1|LD2_Pin|GPIO_PIN_8;
374   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
375   GPIO_InitStruct.Pull = GPIO_NOPULL;
376   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
377   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
378
379   /* EXTI interrupt init*/
380   HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
381   HAL_NVIC_EnableIRQ(EXTI0_IRQn);
382
383 }
384
385 /* USER CODE BEGIN 4 */
```

```
386
387 /* USER CODE END 4 */
388
389 /**
390   * @brief  This function is executed in case of error
       occurrence.
391   * @retval None
392   */
393 void Error_Handler(void)
394 {
395   /* USER CODE BEGIN Error_Handler_Debug */
396   /* User can add his own implementation to report the
       HAL error return state */
397   __disable_irq();
398   while (1)
399   {
400   }
401   /* USER CODE END Error_Handler_Debug */
402 }
403
404 #ifdef  USE_FULL_ASSERT
405 /**
406   * @brief  Reports the name of the source file and the
       source line number
407   *         where the assert_param error has occurred.
408   * @param  file: pointer to the source file name
409   * @param  line: assert_param error line source number
410   * @retval None
411   */
412 void assert_failed(uint8_t *file, uint32_t line)
413 {
414   /* USER CODE BEGIN 6 */
415   /* User can add his own implementation to report the
       file name and line number,
416      ex: printf("Wrong parameters value: file %s on line
       %d\r\n", file, line) */
417   /* USER CODE END 6 */
418 }
419 #endif /* USE_FULL_ASSERT */
420
421
422 uint8_t getBlueNRGVersion(uint8_t *hwVersion, uint16_t *
       fwVersion)
423 {
424   uint8_t status;
425   uint8_t hci_version, lmp_pal_version;
426   uint16_t hci_revision, manufacturer_name,
       lmp_pal_subversion;
427
```

```
428   status = hci_read_local_version_information(&
      hci_version, &hci_revision, &lmp_pal_version,
429                                               &manufacturer_name,
       &lmp_pal_subversion);
430
431   if (status == BLE_STATUS_SUCCESS) {
432     *hwVersion = hci_revision >> 8;
433     *fwVersion = (hci_revision & 0xFF) << 8;
      // Major Version Number
434     *fwVersion |= ((lmp_pal_subversion >> 4) & 0xF) << 4;
      // Minor Version Number
435     *fwVersion |= lmp_pal_subversion & 0xF;
      // Patch Version Number
436   }
437   return status;
438 }
439
440 void APP_UserEvtRx(void *pData)
441 {
442   uint32_t i;
443
444   hci_spi_pckt *hci_pckt = (hci_spi_pckt *)pData;
445
446   if(hci_pckt->type == HCI_EVENT_PKT)
447   {
448     hci_event_pckt *event_pckt = (hci_event_pckt*)
      hci_pckt->data;
449
450     if(event_pckt->evt == EVT_LE_META_EVENT)
451     {
452       evt_le_meta_event *evt = (void *)event_pckt->data;
453
454       for (i = 0; i < (sizeof(hci_le_meta_events_table)/
      sizeof(hci_le_meta_events_table_type)); i++)
455       {
456         if (evt->subevent == hci_le_meta_events_table[i].
      evt_code)
457         {
458           hci_le_meta_events_table[i].process((void *)evt
      ->data);
459         }
460       }
461     }
462     else if(event_pckt->evt == EVT_VENDOR)
463     {
464       evt_blue_aci *blue_evt = (void*)event_pckt->data;
465
466       for (i = 0; i < (sizeof(
      hci_vendor_specific_events_table)/sizeof(
      hci_vendor_specific_events_table_type)); i++)
```

```
467         {
468           if (blue_evt->ecode ==
    hci_vendor_specific_events_table[i].evt_code)
469           {
470             hci_vendor_specific_events_table[i].process((
    void *)blue_evt->data);
471           }
472         }
473       }
474       else
475       {
476         for (i = 0; i < (sizeof(hci_events_table)/sizeof(
    hci_events_table_type)); i++)
477         {
478           if (event_pckt->evt == hci_events_table[i].
    evt_code)
479           {
480             hci_events_table[i].process((void *)event_pckt
    ->data);
481           }
482         }
483       }
484     }
485 }
486
487
488 /* This callback is called when an advertising report is
    received */
489  void hci_le_advertising_report_event(uint8_t Num_Reports
    ,Advertising_Report_t Advertising_Report[])
490 {
491     /* Advertising_Report contains all the expected
    parameters.
492     User application should add code for decoding the
    received
493     Advertising_Report event databased on the specific
    evt_type
494     (ADV_IND, SCAN_RSP, ..)
495     */
496     /* Example: store the received Advertising_Report
    fields */
497
498     if(user_detected==0){
499       int8_t RSSI = Advertising_Report[0].RSSI;
500       if(RSSI>=-60){
501         user_detected=1;
502         for(uint8_t loop = 0; loop < 6; loop++) {
503             bdaddr_cb[loop] = Advertising_Report[0].
    Address[loop];
504           }
```

```
505       }
506     }else{
507       user_check=0;
508       for(uint8_t loop = 0; loop < 6; loop++) {
509         if(bdaddr_cb[loop] == Advertising_Report[0].
    Address[loop]){
510         user_check++;
511        }
512       }
513       if(user_check==6){
514         char buffer[100];
515 
516         /* BLE IDENTIFIER */
517 
518         //uint8_t identifier[3]="A\r\n";
519         //uint8_t identifier[3]="B\r\n";
520         uint8_t identifier[3]="C\r\n";
521         HAL_UART_Transmit(&huart2,identifier,3,10);
522 
523 
524         for(uint8_t loop = 0; loop < 5; loop++) {
525             sprintf(buffer,"%d:",bdaddr_cb[loop]);
526             HAL_UART_Transmit(&huart2,(uint8_t*)buffer,
    strlen(buffer),10);
527              }
528           sprintf(buffer,"%d\r\n",bdaddr_cb[5]);
529           HAL_UART_Transmit(&huart2,(uint8_t*)buffer,
    strlen(buffer),10);
530 
531 
532           /* type of the peer address (PUBLIC_ADDR,
    RANDOM_ADDR) */
533           uint8_t bdaddr_type = Advertising_Report[0].
    Address_Type;
534 
535             sprintf(buffer,"%d\r\n",bdaddr_type);
536 
537           HAL_UART_Transmit(&huart2,(uint8_t*)buffer,
    strlen(buffer),10);
538 
539           /* event type (advertising packets types) */
540           uint8_t evt_type = Advertising_Report[0].
    Event_Type ;
541 
542           sprintf(buffer,"%d\r\n",evt_type);
543 
544           HAL_UART_Transmit(&huart2,(uint8_t*)buffer,
    strlen(buffer),10);
545 
546           /* RSSI value */
```

```
547           int8_t RSSI = Advertising_Report[0].RSSI;
548
549           sprintf(buffer,"%d\r\n",RSSI);
550
551         HAL_UART_Transmit(&huart2,(uint8_t*)buffer,strlen
    (buffer),10);
552       }
553     }
554
555     /* address of the peer device found during discovery
    procedure */
556     //Osal_MemCpy(bdaddr_cb, Advertising_Report[0].Address
    ,6);
557
558     /* length of advertising or scan response data */
559     //uint8_t data_length = Advertising_Report[0].
    Length_Data;
560
561     /* data_length octets of advertising or scan response
    data formatted are
562     on Advertising_Report[0].Data field: to be stored/
    filtered based on
563     specific user application scenario*/
564 } /* hci_le_advertising_report_event() */
565
566
567 /************************ (C) COPYRIGHT
    STMicroelectronics *****END OF FILE****/
```

E.5 Raspberry triangulation code

```
1  import serial;
2  import io;
3  import time;
4  import os;
5  import threading
6
7
8  # the ports are devA = '/dev/ttyACM0'
9  # devB = '/dev/ttyACM1' and devC = '/dev/ttyACM2'
10
11 class BLE(threading.Thread):
12
13     def __init__(self,port):
14         threading.Thread.__init__(self,daemon=True)
15         self.startDev=True
16         self.port=port
17         self.prevRSSI=100
```

```
18          self.currRSSI=100
19          self.smoothRSSI=0
20          self.alpha=0.75
21          self.s=serial.Serial(port=self.port, baudrate
    =115200,
22          timeout=1,xonxoff=False, rtscts=False, dsrdtr=
    True)
23
24      def run(self):
25          while True:
26              try :
27                  # configure the serial connections (the
    parameters differs on
28                  #the device you are connecting to)
29
30                  #s=serial.Serial(port='/dev/ttyACM0',
    baudrate=115200,
31                  #timeout=1,xonxoff=False, rtscts=False,
    dsrdtr=True)
32                  try:
33                      for line in self.s:
34                          #print(line)
35                          if len(line)>5 and self.startDev:
36                              self.address=line
37                              self.startDev=False
38                          if len(line)==5:
39                              if self.prevRSSI==100:
40                                  self.prevRSSI=int(str(
    line)[2:5])
41                              else:
42                                  self.currRSSI=int(str(
    line)[2:5])
43                                  self.smoothRSSI=(self.
    alpha*
44                                  self.currRSSI+(1-self.
    alpha)*self.prevRSSI)
45                                  self.prevRSSI=self.
    currRSSI
46                              break
47
48                  except KeyboardInterrupt:
49                      self.s.close()
50                      print('Program exit!')
51                      break
52              except :
53                  print("BLE ", self.port ," not connected"
    )
54                  print('Program exit!')
55                  break
56
```

```
57      def getAdd(self):
58          if not(self.startDev):
59              return self.address
60          else:
61              return -1
62
63      def getRSSI(self):
64          if not(self.startDev):
65              return self.smoothRSSI
66          else:
67              return -1
68
69      #IF TRUE NOT PAIRED, IF FALSE PAIRED
70      def getStatus(self):
71          return self.startDev
72
73
74  if __name__ == '__main__' :
75
76      DevA=BLE('/dev/ttyACM0')
77      DevA.start()
78      DevB=BLE('/dev/ttyACM1')
79      DevB.start()
80      DevC=BLE('/dev/ttyACM2')
81      DevC.start()
82      #Devices center
83      xa=0.004
84      ya=0
85      xb=-0.2
86      yb=0
87      xc=0
88      yc=0.1
89      while True:
90          if DevA.getAdd()!=-1 and DevA.getAdd()==DevB.
    getAdd() and
91          DevB.getAdd()==DevC.getAdd():
92              if not(DevA.getStatus()):
93                  #print("The RSSI A is ",DevA.getRSSI())
    #+43)
94                  da=10**((-68-DevA.getRSSI())/(10))
95                  #print("The distance from A is: ",da,"m")
96                  time.sleep(0.5)
97              if not(DevB.getStatus()):
98                  #print("The RSSI B is ",DevB.getRSSI())
    #+43)
99                  db=10**((-68-DevB.getRSSI())/(10))
100                 #print("The distance from B is: ",db,"m")
101                 time.sleep(0.5)
102             if not(DevC.getStatus()):
```

```
                    #print("The RSSI C is ",DevC.getRSSI())
    #+48)
                    dc=10**((-68-DevC.getRSSI())/(10))
                    #print("The distance from C is: ",dc,"m")
                    time.sleep(0.5)
                va=((db**2-dc**2)-(xb**2-xc**2)-(yb**2-yc**2)
    )/2
                vb=((db**2-da**2)-(xb**2-xa**2)-(yb**2-ya**2)
    )/2
                y=(vb*(xc-xb)-va*(xa-xb))/((ya-yb)*(xc-xb)-(
    yc-yb)*(xa-xb))
                x=(va-y*(yc-yb))/(xc-xb)
                print("User position X: ",x,"m Y: ",y,"m")
```

E.6 STM32 IMU code

```
#ifdef __cplusplus
extern "C" {
#endif

/* Includes
    ------------------------------------------------------------
    */
#include "app_mems.h"
#include "main.h"
#include <stdio.h>

#include "stm32f4xx_hal.h"
#include "stm32f4xx_nucleo.h"
#include "com.h"
#include "demo_serial.h"
#include "bsp_ip_conf.h"
#include "fw_version.h"
#include "motion_fx_manager.h"

/* Private typedef
    -----------------------------------------------------------
    */
/* Private define
    ------------------------------------------------------------
    */
#define DWT_LAR_KEY  0xC5ACCE55 /* DWT register unlock
    key */
#define ALGO_FREQ  100U /* Algorithm frequency 100Hz */
#define ACC_ODR  ((float)ALGO_FREQ)
#define ACC_FS  4 /* FS = <-4g, 4g> */
#define ALGO_PERIOD  (1000U / ALGO_FREQ) /* Algorithm
    period [ms] */
```

```
25 #define MOTION_FX_ENGINE_DELTATIME  0.01f
26 #define FROM_MG_TO_G  0.001f
27 #define FROM_G_TO_MG  1000.0f
28 #define FROM_MDPS_TO_DPS  0.001f
29 #define FROM_DPS_TO_MDPS  1000.0f
30 #define FROM_MGAUSS_TO_UT50  (0.1f/50.0f)
31 #define FROM_UT50_TO_MGAUSS  500.0f
32
33 #define MAX_BUF_SIZE 256
34
35 typedef struct
36 {
37   float rotation[MFX_NUM_AXES];             /* yaw, pitch
    and roll */
38   float quaternion[MFX_QNUM_AXES];          /* quaternion
    */
39   float gravity[MFX_NUM_AXES];              /* device
    frame gravity */
40   float linear_acceleration[MFX_NUM_AXES]; /* device
    frame linear acceleration */
41   float heading;                            /* heading */
42   float headingErr;                         /* heading
    error in deg */
43 } dataformat;
44
45 /* Public variables
    ---------------------------------------------------------
    */
46 volatile uint8_t DataLoggerActive = 0;
47 volatile uint32_t SensorsEnabled = 0;
48 char LibVersion[35];
49 int LibVersionLen;
50 volatile uint8_t SensorReadRequest = 0;
51 uint8_t UseOfflineData = 0;
52 offline_data_t OfflineData[OFFLINE_DATA_SIZE];
53 int OfflineDataReadIndex = 0;
54 int OfflineDataWriteIndex = 0;
55 int OfflineDataCount = 0;
56 uint32_t AlgoFreq = ALGO_FREQ;
57 uint8_t Enabled6X = 0;
58 static int32_t PushButtonState = GPIO_PIN_RESET;
59
60 /* Extern variables
    -----------------------------------------*/
61 /* Private macro -------------------------------------*/
62 /* Private variables
    ---------------------------------------------*/
63 static MOTION_SENSOR_Axes_t AccValue;
64 static MOTION_SENSOR_Axes_t GyrValue;
65 static MOTION_SENSOR_Axes_t MagValue;
```

```
66 static float PressValue;
67 static float TempValue;
68 static float HumValue;
69 static volatile uint32_t TimeStamp = 0;
70 static volatile uint8_t MagCalRequest = 0;
71 static MOTION_SENSOR_Axes_t MagOffset;
72 static uint8_t MagCalStatus = 0;
73
74 static char dataOut[MAX_BUF_SIZE];
75 static char ax_send[MAX_BUF_SIZE],ay_send[MAX_BUF_SIZE],
      az_send[MAX_BUF_SIZE];
76 static char gx_send[MAX_BUF_SIZE],gy_send[MAX_BUF_SIZE],
      gz_send[MAX_BUF_SIZE];
77 static char mx_send[MAX_BUF_SIZE],my_send[MAX_BUF_SIZE],
      mz_send[MAX_BUF_SIZE];
78 static char quat0_send[MAX_BUF_SIZE],quat1_send[
      MAX_BUF_SIZE],quat2_send[MAX_BUF_SIZE],quat3_send[
      MAX_BUF_SIZE];
79 static char rot0_send[MAX_BUF_SIZE],rot1_send[
      MAX_BUF_SIZE],rot2_send[MAX_BUF_SIZE];
80 static char grav0_send[MAX_BUF_SIZE],grav1_send[
      MAX_BUF_SIZE],grav2_send[MAX_BUF_SIZE];
81 static char acc0_send[MAX_BUF_SIZE],acc1_send[
      MAX_BUF_SIZE],acc2_send[MAX_BUF_SIZE];
82
83 dataformat data_to_send;
84
85 /* Private function prototypes
      -----------------------------------------------*/
86 static void MX_DataLogFusion_Init(void);
87 static void MX_DataLogFusion_Process(void);
88 static void FX_Data_Handler(TMsg *Msg);
89 static void Init_Sensors(void);
90 static void RTC_Handler(TMsg *Msg);
91 static void Accelero_Sensor_Handler(TMsg *Msg);
92 static void Gyro_Sensor_Handler(TMsg *Msg);
93 static void Magneto_Sensor_Handler(TMsg *Msg);
94 static void Pressure_Sensor_Handler(TMsg *Msg);
95 static void Temperature_Sensor_Handler(TMsg *Msg);
96 static void Humidity_Sensor_Handler(TMsg *Msg);
97 static void TIM_Config(uint32_t Freq);
98 static void DWT_Init(void);
99 static void DWT_Start(void);
100 static uint32_t DWT_Stop(void);
101
102 void MX_MEMS_Init(void)
103 {
104   /* USER CODE BEGIN SV */
105
106   /* USER CODE END SV */
```

```
107
108   /* USER CODE BEGIN MEMS_Init_PreTreatment */
109
110   /* USER CODE END MEMS_Init_PreTreatment */
111
112   /* Initialize the peripherals and the MEMS components
       */
113
114   MX_DataLogFusion_Init();
115
116   /* USER CODE BEGIN MEMS_Init_PostTreatment */
117
118   /* USER CODE END MEMS_Init_PostTreatment */
119 }
120
121 /*
122  * LM background task
123  */
124 void MX_MEMS_Process(void)
125 {
126   /* USER CODE BEGIN MEMS_Process_PreTreatment */
127
128   /* USER CODE END MEMS_Process_PreTreatment */
129
130   MX_DataLogFusion_Process();
131
132   /* USER CODE BEGIN MEMS_Process_PostTreatment */
133
134   /* USER CODE END MEMS_Process_PostTreatment */
135 }
136
137 /* Exported functions
       ---------------------------------------*/
138 /**
139  * @brief  Period elapsed callback
140  * @param  htim pointer to a TIM_HandleTypeDef structure
       that contains
141  *              the configuration information for TIM
       module.
142  * @retval None
143  */
144 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *
       htim)
145 {
146   if (htim->Instance == BSP_IP_TIM_Handle.Instance)
147   {
148     SensorReadRequest = 1;
149   }
150 }
151
```

```
152 /* Private functions
      ---------------------------------------*/
153 /**
154   * @brief  Initialize the application
155   * @retval None
156   */
157 static void MX_DataLogFusion_Init(void)
158 {
159   float ans_float;
160 
161   /* Initialize button */
162   BSP_PB_Init(BUTTON_KEY, BUTTON_MODE_EXTI);
163 
164   /* Check what is the Push Button State when the button
       is not pressed. It can change across families */
165   PushButtonState = (BSP_PB_GetState(BUTTON_KEY)) ?  0 :
       1;
166 
167   /* Initialize LED */
168   BSP_LED_Init(LED2);
169 
170   /* Initialize Virtual COM Port */
171   BSP_COM_Init(COM1);
172 
173   /* Initialize Timer */
174   BSP_IP_TIM_Init();
175 
176   /* Configure Timer to run with desired algorithm
       frequency */
177   TIM_Config(ALGO_FREQ);
178 
179   /* Initialize (disabled) sensors */
180   Init_Sensors();
181 
182   /* Sensor Fusion API initialization function */
183   MotionFX_manager_init();
184 
185   /* OPTIONAL */
186   /* Get library version */
187   MotionFX_manager_get_version(LibVersion, &LibVersionLen
       );
188 
189   /* Enable magnetometer calibration */
190   MotionFX_manager_MagCal_start(ALGO_PERIOD);
191 
192   /* Test if calibration data are available */
193   MFX_MagCal_output_t mag_cal_test;
194   MotionFX_MagCal_getParams(&mag_cal_test);
195 
```

```
196   /* If calibration data are available load HI
       coefficients */
197   if (mag_cal_test.cal_quality == MFX_MAGCALGOOD)
198   {
199     ans_float = (mag_cal_test.hi_bias[0] *
      FROM_UT50_TO_MGAUSS);
200     MagOffset.x = (int32_t)ans_float;
201     ans_float = (mag_cal_test.hi_bias[1] *
      FROM_UT50_TO_MGAUSS);
202     MagOffset.y = (int32_t)ans_float;
203     ans_float = (mag_cal_test.hi_bias[2] *
      FROM_UT50_TO_MGAUSS);
204     MagOffset.z = (int32_t)ans_float;
205
206     MagCalStatus = 1;
207   }
208
209   DWT_Init();
210
211   BSP_LED_On(LED2);
212   HAL_Delay(500);
213   BSP_LED_Off(LED2);
214
215   /* Start receiving messages via DMA */
216   UART_StartReceiveMsg();
217 }
218
219 /**
220   * @brief  Process of the application
221   * @retval None
222   */
223 static void MX_DataLogFusion_Process(void)
224 {
225   static TMsg msg_dat;
226   static TMsg msg_cmd;
227
228   if (MagCalRequest == 1U)
229   {
230     //Debouncing
231     HAL_Delay(50);
232
233     //Wait until the button is released
234     while ((BSP_PB_GetState( BUTTON_KEY ) ==
      PushButtonState));
235
236     //Debouncing
237     HAL_Delay(50);
238
239     MagCalRequest = 0;
240
```

```
241     //Reset magnetometer calibration value
242     MagCalStatus = 0;
243     MagOffset.x = 0;
244     MagOffset.y = 0;
245     MagOffset.z = 0;
246 
247     //Enable magnetometer calibration
248     MotionFX_manager_MagCal_start(ALGO_PERIOD);
249   }
250 
251 /*  if (SensorReadRequest == 1U)
252   {
253     SensorReadRequest = 0;*/
254 
255     //Acquire data from enabled sensors and fill Msg
    stream
256     //RTC_Handler(&msg_dat);
257     Accelero_Sensor_Handler(&msg_dat);
258     Gyro_Sensor_Handler(&msg_dat);
259     Magneto_Sensor_Handler(&msg_dat);
260     //Humidity_Sensor_Handler(&msg_dat);
261     //Temperature_Sensor_Handler(&msg_dat);
262     //Pressure_Sensor_Handler(&msg_dat);
263 
264     //Sensor Fusion specific part
265     FX_Data_Handler(&msg_dat);
266 
267     //Send data stream
268 /*    INIT_STREAMING_HEADER(&msg_dat);
269     msg_dat.Len = STREAMING_MSG_LENGTH;
270 
271     if (UseOfflineData == 1U)
272     {
273       OfflineDataCount--;
274       if (OfflineDataCount < 0)
275       {
276         OfflineDataCount = 0;
277       }
278 
279       OfflineDataReadIndex++;
280       if (OfflineDataReadIndex >= OFFLINE_DATA_SIZE)
281       {
282         OfflineDataReadIndex = 0;
283       }
284 
285       if (OfflineDataCount > 0)
286       {
287         SensorReadRequest = 1;
288       }
289     }
```

```
290     UART_SendMsg(&msg_dat);
291   }*/
292     //printf("CIAO");
293 }
294 
295 /**
296  * @brief  Initialize all sensors
297  * @param  None
298  * @retval None
299  */
300 static void Init_Sensors(void)
301 {
302   BSP_SENSOR_ACC_Init();
303   BSP_SENSOR_GYR_Init();
304   BSP_SENSOR_MAG_Init();
305   BSP_SENSOR_PRESS_Init();
306   BSP_SENSOR_TEMP_Init();
307   BSP_SENSOR_HUM_Init();
308 
309   BSP_SENSOR_ACC_SetOutputDataRate(ACC_ODR);
310   BSP_SENSOR_ACC_SetFullScale(ACC_FS);
311 }
312 
313 /**
314  * @brief  Handles the time+date getting/sending
315  * @param  Msg the time+date part of the stream
316  * @retval None
317  */
318 static void RTC_Handler(TMsg *Msg)
319 {
320   uint8_t sub_sec = 0;
321   RTC_DateTypeDef sdatestructureget;
322   RTC_TimeTypeDef stimestructure;
323   uint32_t ans_uint32;
324   int32_t ans_int32;
325   uint32_t RtcSynchPrediv = hrtc.Init.SynchPrediv;
326 
327   if (UseOfflineData == 1)
328   {
329     Msg->Data[3] = (uint8_t)OfflineData[
    OfflineDataReadIndex].hours;
330     Msg->Data[4] = (uint8_t)OfflineData[
    OfflineDataReadIndex].minutes;
331     Msg->Data[5] = (uint8_t)OfflineData[
    OfflineDataReadIndex].seconds;
332     Msg->Data[6] = (uint8_t)OfflineData[
    OfflineDataReadIndex].subsec;
333   }
334   else
335   {
```

```
336     (void)HAL_RTC_GetTime(&hrtc, &stimestructure,
    FORMAT_BIN);
337     (void)HAL_RTC_GetDate(&hrtc, &sdatestructureget,
    FORMAT_BIN);
338
339     /* To be MISRA C-2012 compliant the original
    calculation:
340         sub_sec = ((((((int)RtcSynchPrediv) - ((int)
    stimestructure.SubSeconds)) * 100) / (RtcSynchPrediv +
     1)) & 0xFF);
341         has been split to separate expressions */
342     ans_int32 = (RtcSynchPrediv - (int32_t)stimestructure
    .SubSeconds) * 100;
343     ans_int32 /= RtcSynchPrediv + 1;
344     ans_uint32 = (uint32_t)ans_int32 & 0xFFU;
345     sub_sec = (uint8_t)ans_uint32;
346
347     Msg->Data[3] = (uint8_t)stimestructure.Hours;
348     Msg->Data[4] = (uint8_t)stimestructure.Minutes;
349     Msg->Data[5] = (uint8_t)stimestructure.Seconds;
350     Msg->Data[6] = sub_sec;
351   }
352 }
353
354 /**
355  * @brief  Sensor Fusion data handler
356  * @param  Msg the Sensor Fusion data part of the stream
357  * @retval None
358  */
359 static void FX_Data_Handler(TMsg *Msg)
360 {
361   uint32_t elapsed_time_us = 0U;
362   MFX_input_t data_in;
363   MFX_input_t *pdata_in = &data_in;
364   MFX_output_t data_out;
365   MFX_output_t *pdata_out = &data_out;
366
367 /*  if ((SensorsEnabled & ACCELEROMETER_SENSOR) ==
    ACCELEROMETER_SENSOR)
368   {
369     if ((SensorsEnabled & GYROSCOPE_SENSOR) ==
    GYROSCOPE_SENSOR)
370     {
371       if ((SensorsEnabled & MAGNETIC_SENSOR) ==
    MAGNETIC_SENSOR)
372       {*/
373         /* Convert angular velocity from [mdps] to [dps]
    */
374         data_in.gyro[0] = (float)GyrValue.x *
    FROM_MDPS_TO_DPS;
```

```
375         data_in.gyro[1] = (float)GyrValue.y *
    FROM_MDPS_TO_DPS;
376         data_in.gyro[2] = (float)GyrValue.z *
    FROM_MDPS_TO_DPS;
377
378         /* Convert acceleration from [mg] to [g] */
379         data_in.acc[0] = (float)AccValue.x * FROM_MG_TO_G
    ;
380         data_in.acc[1] = (float)AccValue.y * FROM_MG_TO_G
    ;
381         data_in.acc[2] = (float)AccValue.z * FROM_MG_TO_G
    ;
382
383         /* Convert magnetic field intensity from [mGauss]
     to [uT / 50] */
384         data_in.mag[0] = (float)MagValue.x *
    FROM_MGAUSS_TO_UT50;
385         data_in.mag[1] = (float)MagValue.y *
    FROM_MGAUSS_TO_UT50;
386         data_in.mag[2] = (float)MagValue.z *
    FROM_MGAUSS_TO_UT50;
387
388         gcvt(data_in.acc[0],6,ax_send);
389     gcvt(data_in.acc[1],6,ay_send);
390     gcvt(data_in.acc[2],6,az_send);
391
392 /*    gcvt(data_in.gyro[0],6,gx_send);
393     gcvt(data_in.gyro[1],6,gy_send);
394     gcvt(data_in.gyro[2],6,gz_send);
395
396     gcvt(data_in.mag[0],6,mx_send);
397     gcvt(data_in.mag[1],6,my_send);
398     gcvt(data_in.mag[2],6,mz_send);*/
399
400 //       snprintf(dataOut, MAX_BUF_SIZE, "\r\n%s, %s, %s,
    %s, %s, %s, %s, %s, %s\r\n", ax_send, ay_send, az_send
    ,
401 //           gx_send, gy_send, gz_send, mx_send, my_send,
    mz_send);
402 //       printf("%s", dataOut);
403
404
405         /* Run Sensor Fusion algorithm */
406         BSP_LED_On(LED2);
407         DWT_Start();
408         MotionFX_manager_run(pdata_in, pdata_out,
    MOTION_FX_ENGINE_DELTATIME);
409         elapsed_time_us = DWT_Stop();
410         BSP_LED_Off(LED2);
411
```

```
412         data_to_send.quaternion[0]=pdata_out->quaternion
    [0];
413         data_to_send.quaternion[1]=pdata_out->quaternion
    [1];
414         data_to_send.quaternion[2]=pdata_out->quaternion
    [2];
415         data_to_send.quaternion[3]=pdata_out->quaternion
    [3];
416
417     gcvt(data_to_send.quaternion[0],6,quat0_send);
418     gcvt(data_to_send.quaternion[1],6,quat1_send);
419     gcvt(data_to_send.quaternion[2],6,quat2_send);
420     gcvt(data_to_send.quaternion[3],6,quat3_send);
421
422 //      snprintf(dataOut, MAX_BUF_SIZE, "\r\nQuaternion:
    [ %s , %s, %s, %s]\r\n", quat0_send,quat1_send,
    quat2_send,quat3_send);
423 //      printf("%s", dataOut);
424
425       data_to_send.rotation[0]=pdata_out->rotation[0];
426     data_to_send.rotation[1]=pdata_out->rotation[1];
427     data_to_send.rotation[2]=pdata_out->rotation[2];
428
429     gcvt(data_to_send.rotation[0],6,rot0_send);
430     gcvt(data_to_send.rotation[1],6,rot1_send);
431     gcvt(data_to_send.rotation[2],6,rot2_send);
432
433 //    snprintf(dataOut, MAX_BUF_SIZE, "\r\nRotation: [ %s
    , %s, %s]\r\n", rot0_send,rot1_send,rot2_send);
434 //    printf("%s", dataOut);
435
436 /*    data_to_send.gravity[0]=pdata_out->gravity[0];
437     data_to_send.gravity[1]=pdata_out->gravity[1];
438     data_to_send.gravity[2]=pdata_out->gravity[2];
439
440     gcvt(data_to_send.gravity[0],6,grav0_send);
441     gcvt(data_to_send.gravity[1],6,grav1_send);
442     gcvt(data_to_send.gravity[2],6,grav2_send);*/
443
444 //    snprintf(dataOut, MAX_BUF_SIZE, "\r\nGravity: [ %s
    , %s, %s]\r\n", grav0_send,grav1_send,grav2_send);
445 //    printf("%s", dataOut);
446
447     data_to_send.linear_acceleration[0]=pdata_out->
    linear_acceleration[0];
448     data_to_send.linear_acceleration[1]=pdata_out->
    linear_acceleration[1];
449     data_to_send.linear_acceleration[2]=pdata_out->
    linear_acceleration[2];
450
```

```
451     gcvt(data_to_send.linear_acceleration[0],6,acc0_send)
    ;
452     gcvt(data_to_send.linear_acceleration[1],6,acc1_send)
    ;
453     gcvt(data_to_send.linear_acceleration[2],6,acc2_send)
    ;
454
455 //    snprintf(dataOut, MAX_BUF_SIZE, "\r\nLinear
    acceleration: [ %s , %s, %s]\r\n", acc0_send,acc1_send
    ,acc2_send);
456 //    printf("%s", dataOut);
457
458     snprintf(dataOut, MAX_BUF_SIZE, "\r\n%s,%s,%s,%s,%s,%
    s,%s,%s,%s,%s,%s,%s,%s\r\n", quat0_send,
459         quat1_send,quat2_send,quat3_send,rot0_send,
    rot1_send,rot2_send,
460         acc0_send,acc1_send,acc2_send,ax_send, ay_send,
    az_send);
461     printf("%s", dataOut);
462
463         (void)memcpy(&Msg->Data[55], (void *)pdata_out->
    quaternion, 4U * sizeof(float));
464         (void)memcpy(&Msg->Data[71], (void *)pdata_out->
    rotation, 3U * sizeof(float));
465         (void)memcpy(&Msg->Data[83], (void *)pdata_out->
    gravity, 3U * sizeof(float));
466         (void)memcpy(&Msg->Data[95], (void *)pdata_out->
    linear_acceleration, 3U * sizeof(float));
467
468         (void)memcpy(&Msg->Data[107], (void *) & (
    pdata_out->heading), sizeof(float));
469         (void)memcpy(&Msg->Data[111], (void *) & (
    pdata_out->headingErr), sizeof(float));
470
471         Serialize_s32(&Msg->Data[115], (int32_t)
    elapsed_time_us, 4);
472 /*      }
473     }
474   }*/
475 }
476
477 /**
478   * @brief  BSP Push Button callback
479   * @param  Button Specifies the pin connected EXTI line
480   * @retval None.
481   */
482 void BSP_PB_Callback(Button_TypeDef Button)
483 {
484   MagCalRequest = 1U;
485 }
```

```
486
487 /**
488  * @brief  Handles the ACC axes data getting/sending
489  * @param  Msg the ACC part of the stream
490  * @retval None
491  */
492 static void Accelero_Sensor_Handler(TMsg *Msg)
493 {
494 /*  if ((SensorsEnabled & ACCELEROMETER_SENSOR) ==
       ACCELEROMETER_SENSOR)
495   {
496     if (UseOfflineData == 1)
497     {
498       AccValue.x = OfflineData[OfflineDataReadIndex].
       acceleration_x_mg;
499       AccValue.y = OfflineData[OfflineDataReadIndex].
       acceleration_y_mg;
500       AccValue.z = OfflineData[OfflineDataReadIndex].
       acceleration_z_mg;
501     }
502     else
503     {*/
504       BSP_SENSOR_ACC_GetAxes(&AccValue);
505 /*    }
506
507     Serialize_s32(&Msg->Data[19], (int32_t)AccValue.x, 4)
       ;
508     Serialize_s32(&Msg->Data[23], (int32_t)AccValue.y, 4)
       ;
509     Serialize_s32(&Msg->Data[27], (int32_t)AccValue.z, 4)
       ;
510   }*/
511 }
512
513 /**
514  * @brief  Handles the GYR axes data getting/sending
515  * @param  Msg the GYR part of the stream
516  * @retval None
517  */
518 static void Gyro_Sensor_Handler(TMsg *Msg)
519 {
520 //  if ((SensorsEnabled & GYROSCOPE_SENSOR) ==
       GYROSCOPE_SENSOR)
521 //  {
522 /*    if (UseOfflineData == 1)
523     {
524       GyrValue.x = OfflineData[OfflineDataReadIndex].
       angular_rate_x_mdps;
525       GyrValue.y = OfflineData[OfflineDataReadIndex].
       angular_rate_y_mdps;
```

```
526       GyrValue.z = OfflineData[OfflineDataReadIndex].
    angular_rate_z_mdps;
527     }*/
528     //else
529     //{
530       BSP_SENSOR_GYR_GetAxes(&GyrValue);
531     //}
532
533 /*    Serialize_s32(&Msg->Data[31], GyrValue.x, 4);
534     Serialize_s32(&Msg->Data[35], GyrValue.y, 4);
535     Serialize_s32(&Msg->Data[39], GyrValue.z, 4);*/
536 //  }
537 }
538
539 /**
540  * @brief  Handles the MAG axes data getting/sending
541  * @param  Msg the MAG part of the stream
542  * @retval None
543  */
544 static void Magneto_Sensor_Handler(TMsg *Msg)
545 {
546   float ans_float;
547   MFX_MagCal_input_t mag_data_in;
548   MFX_MagCal_output_t mag_data_out;
549
550 /*  if ((SensorsEnabled & MAGNETIC_SENSOR) ==
    MAGNETIC_SENSOR)
551   {
552     if (UseOfflineData == 1)
553     {
554       MagValue.x = OfflineData[OfflineDataReadIndex].
    magnetic_field_x_mgauss;
555       MagValue.y = OfflineData[OfflineDataReadIndex].
    magnetic_field_y_mgauss;
556       MagValue.z = OfflineData[OfflineDataReadIndex].
    magnetic_field_z_mgauss;
557     }
558     else
559     {*/
560       BSP_SENSOR_MAG_GetAxes(&MagValue);
561
562       if (MagCalStatus == 0U)
563       {
564         mag_data_in.mag[0] = (float)MagValue.x *
    FROM_MGAUSS_TO_UT50;
565         mag_data_in.mag[1] = (float)MagValue.y *
    FROM_MGAUSS_TO_UT50;
566         mag_data_in.mag[2] = (float)MagValue.z *
    FROM_MGAUSS_TO_UT50;
567
```

```
568         mag_data_in.time_stamp = (int)TimeStamp;
569         TimeStamp += (uint32_t)ALGO_PERIOD;
570 
571         MotionFX_manager_MagCal_run(&mag_data_in, &
    mag_data_out);
572 
573         if (mag_data_out.cal_quality == MFX_MAGCALGOOD)
574         {
575           MagCalStatus = 1;
576 
577           ans_float = (mag_data_out.hi_bias[0] *
    FROM_UT50_TO_MGAUSS);
578           MagOffset.x = (int32_t)ans_float;
579           ans_float = (mag_data_out.hi_bias[1] *
    FROM_UT50_TO_MGAUSS);
580           MagOffset.y = (int32_t)ans_float;
581           ans_float = (mag_data_out.hi_bias[2] *
    FROM_UT50_TO_MGAUSS);
582           MagOffset.z = (int32_t)ans_float;
583 
584           /* Disable magnetometer calibration */
585           MotionFX_manager_MagCal_stop(ALGO_PERIOD);
586         }
587       }
588 
589       MagValue.x = (int32_t)(MagValue.x - MagOffset.x);
590       MagValue.y = (int32_t)(MagValue.y - MagOffset.y);
591       MagValue.z = (int32_t)(MagValue.z - MagOffset.z);
592 /*     }
593 
594     Serialize_s32(&Msg->Data[43], MagValue.x, 4);
595     Serialize_s32(&Msg->Data[47], MagValue.y, 4);
596     Serialize_s32(&Msg->Data[51], MagValue.z, 4);
597   }*/
598 }
599 
600 /**
601  * @brief  Handles the PRESS sensor data getting/sending.
602  * @param  Msg the PRESS part of the stream
603  * @retval None
604  */
605 static void Pressure_Sensor_Handler(TMsg *Msg)
606 {
607   if ((SensorsEnabled & PRESSURE_SENSOR) ==
    PRESSURE_SENSOR)
608   {
609     if (UseOfflineData == 1)
610     {
611       PressValue = OfflineData[OfflineDataReadIndex].
    pressure;
```

```
612     }
613     else
614     {
615       BSP_SENSOR_PRESS_GetValue(&PressValue);
616     }
617 
618     (void)memcpy(&Msg->Data[7], (void *)&PressValue,
    sizeof(float));
619   }
620 }
621 
622 /**
623  * @brief  Handles the TEMP axes data getting/sending
624  * @param  Msg the TEMP part of the stream
625  * @retval None
626  */
627 static void Temperature_Sensor_Handler(TMsg *Msg)
628 {
629   if ((SensorsEnabled & TEMPERATURE_SENSOR) ==
    TEMPERATURE_SENSOR)
630   {
631     if (UseOfflineData == 1)
632     {
633       TempValue = OfflineData[OfflineDataReadIndex].
    temperature;
634     }
635     else
636     {
637       BSP_SENSOR_TEMP_GetValue(&TempValue);
638     }
639 
640     (void)memcpy(&Msg->Data[11], (void *)&TempValue,
    sizeof(float));
641   }
642 }
643 
644 /**
645  * @brief  Handles the HUM axes data getting/sending
646  * @param  Msg the HUM part of the stream
647  * @retval None
648  */
649 static void Humidity_Sensor_Handler(TMsg *Msg)
650 {
651   if ((SensorsEnabled & HUMIDITY_SENSOR) ==
    HUMIDITY_SENSOR)
652   {
653     if (UseOfflineData == 1)
654     {
655       HumValue = OfflineData[OfflineDataReadIndex].
    humidity;
```

```
656     }
657     else
658     {
659       BSP_SENSOR_HUM_GetValue(&HumValue);
660     }
661 
662     (void)memcpy(&Msg->Data[15], (void *)&HumValue,
    sizeof(float));;
663   }
664 }
665 
666 /**
667  * @brief  Timer configuration
668  * @param  Freq the desired Timer frequency
669  * @retval None
670  */
671 static void TIM_Config(uint32_t Freq)
672 {
673   const uint32_t tim_counter_clock = 2000; /* TIM counter
      clock 2 kHz */
674   uint32_t prescaler_value = (uint32_t)((SystemCoreClock
     / tim_counter_clock) - 1);
675   uint32_t period = (tim_counter_clock / Freq) - 1;
676 
677   BSP_IP_TIM_Handle.Init.Prescaler = prescaler_value;
678   BSP_IP_TIM_Handle.Init.CounterMode = TIM_COUNTERMODE_UP
     ;
679   BSP_IP_TIM_Handle.Init.Period = period;
680   BSP_IP_TIM_Handle.Init.ClockDivision =
     TIM_CLOCKDIVISION_DIV1;
681   BSP_IP_TIM_Handle.Init.AutoReloadPreload =
     TIM_AUTORELOAD_PRELOAD_DISABLE;
682   if (HAL_TIM_Base_Init(&BSP_IP_TIM_Handle) != HAL_OK)
683   {
684     Error_Handler();
685   }
686 }
687 
688 /**
689  * @brief  Initialize DWT register for counting clock
     cycles purpose
690  * @param  None
691  * @retval None
692  */
693 static void DWT_Init(void)
694 {
695   CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk;
696 
697   DWT->CTRL &= ~DWT_CTRL_CYCCNTENA_Msk; /* Disable
     counter */
```

```
}

/**
 * @brief  Start counting clock cycles
 * @param  None
 * @retval None
 */
static void DWT_Start(void)
{
  DWT->CYCCNT = 0; /* Clear count of clock cycles */
  DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk; /* Enable counter */
}

/**
 * @brief  Stop counting clock cycles and calculate elapsed time in [us]
 * @param  None
 * @retval Elapsed time in [us]
 */
static uint32_t DWT_Stop(void)
{
  volatile uint32_t cycles_count = 0U;
  uint32_t system_core_clock_mhz = 0U;

  DWT->CTRL &= ~DWT_CTRL_CYCCNTENA_Msk; /* Disable counter */
  cycles_count = DWT->CYCCNT; /* Read count of clock cycles */

  /* Calculate elapsed time in [us] */
  system_core_clock_mhz = SystemCoreClock / 1000000U;
  return cycles_count / system_core_clock_mhz;
}

#ifdef __cplusplus
}
#endif

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

E.7 Raspberry 3D IMU code

```
import serial;
import io;
import time;
import os;
```

```
5
6  import numpy as np
7  from numpy.linalg import inv, norm
8  from operator import add
9  import math
10 from scipy import signal
11
12 def quaternion_rotation_matrix(Q):
13     """
14     Covert a quaternion into a full three-dimensional
   rotation matrix.
15
16     Input
17     :param Q: A 4 element array representing the
   quaternion (q0,q1,q2,q3)
18
19     Output
20     :return: A 3x3 element matrix representing the full 3
   D rotation matrix.
21              This rotation matrix converts a point in the
    local reference
22              frame to a point in the global reference
   frame.
23     """
24     # Extract the values from Q
25     q0 = Q[0]
26     q1 = Q[1]
27     q2 = Q[2]
28     q3 = Q[3]
29
30     # First row of the rotation matrix
31     r00 = 2 * (q0 * q0 + q1 * q1) - 1
32     r01 = 2 * (q1 * q2 - q0 * q3)
33     r02 = 2 * (q1 * q3 + q0 * q2)
34
35     # Second row of the rotation matrix
36     r10 = 2 * (q1 * q2 + q0 * q3)
37     r11 = 2 * (q0 * q0 + q2 * q2) - 1
38     r12 = 2 * (q2 * q3 - q0 * q1)
39
40     # Third row of the rotation matrix
41     r20 = 2 * (q1 * q3 - q0 * q2)
42     r21 = 2 * (q2 * q3 + q0 * q1)
43     r22 = 2 * (q0 * q0 + q3 * q3) - 1
44
45     # 3x3 rotation matrix
46     rot_matrix = np.array([[r00, r01, r02],
47                            [r10, r11, r12],
48                            [r20, r21, r22]])
49
```

```
50      return rot_matrix
51
52  if __name__ == '__main__' :
53      start_serial=True
54
55      dt=0.01
56      p_old=np.array([0,0,0])
57      v_old=np.array([0,0,0])
58
59      #time.sleep(5)
60
61      calibration=True
62      cal=0
63
64      px=0
65      py=0
66      pz=0
67
68      vx=0
69      vy=0
70      vz=0
71
72      pFIX=[0,0,0]
73
74      sos=signal.butter(1,0.2/(1/dt),btype='highpass',
    analog=False,output='sos')
75      while True:
76          try :
77              # configure the serial connections (the
    parameters differs on the device you are connecting to
    )
78              with serial.Serial(port='/dev/ttyACM0',
    baudrate=115200, timeout=0.1,xonxoff=False, rtscts=
    False, dsrdtr=True) as s:
79                  try:
80                      for line in s:
81                          #t1 = time.time()
82                          #print(line.decode("utf-8"))
83                          data=line.decode("utf-8").split("
    ,")
84                          i=0
85                          if len(data)>2:
86                              for num in data:
87                                  data[i]=float(num)
88                                  i=i+1
89                              #print(data)
90                              quaternions=data[0:4]
91                              #print(quaternions)
92                              rotation=data[4:7]
93                              #print(rotation)
```

```
94                              linear_acceleration=[x*9.8
    for x in data[7:10]]
95                              accelerometer_raw=[x*9.8 for
    x in data[10:13]]
96                              #print(accelerometer_raw)
97                              if calibration:
98                                  if cal==0:
99                                      acc_cal=[
    linear_acceleration[0],linear_acceleration[1],
    linear_acceleration[2]]
100                                     #acc_cal=[
    accelerometer_raw[0],accelerometer_raw[1],
    accelerometer_raw[2]]
101
102                                 else:
103                                     #acc_cal=[(acc_cal
    [0]+linear_acceleration[0])/2,(acc_cal[1]+
    linear_acceleration[1])/2,(acc_cal[2]+
    linear_acceleration[2])/2]
104                                     #acc_cal=[acc_cal[0]+
    accelerometer_raw[0],acc_cal[1]+accelerometer_raw[1],
    acc_cal[2]+accelerometer_raw[2]]
105                                     #acc_cal=list(map(add
    ,acc_cal,accelerometer_raw))
106                                     acc_cal=list(map(add,
    acc_cal,linear_acceleration))
107                                 cal=cal+1
108                                 if cal==500:
109                                     calibration=False
110                                     print(acc_cal)
111                                     acc_cal=[x/cal for x
    in acc_cal]
112                                     print(acc_cal)
113
114                                 #print(acc_cal)
115
116                             else:
117                                 #accelerometer_raw=[
    accelerometer_raw[0]-acc_cal[0],accelerometer_raw[1]-
    acc_cal[1],accelerometer_raw[2]-acc_cal[2]]
118                                 #print(acc_cal)
119                                 for i in range(3):
120                                     if abs(
    accelerometer_raw[i])<0.1:
121                                         accelerometer_raw
    [i]=0
122                                 #print(accelerometer_raw)
```

```
123                                     linear_acceleration=np.
    array([linear_acceleration[0]-acc_cal[0],
    linear_acceleration[1]-acc_cal[1],linear_acceleration
    [2]-acc_cal[2]])
124                                     #print("Non filtrato")
125                                     #print(
    linear_acceleration)
126
127                                     #linear_accelerationHP=
    signal.sosfilt(sos,linear_acceleration)
128                                     #print("Filtrato")
129                                     #print(
    linear_accelerationHP)
130
131                                     #vx=v_old[0] +
    linear_acceleration[0]*dt
132                                     #vy=v_old[1] +
    linear_acceleration[1]*dt
133                                     #vz=v_old[2] +
    linear_acceleration[2]*dt
134
135                                     #vx=(accelerometer_raw
    [0]*math.cos(rotation[0])+accelerometer_raw[1]*math.
    cos(rotation[0]+90))*dt
136                                     #vy=(accelerometer_raw
    [0]*math.sin(rotation[0])+accelerometer_raw[1]*math.
    sin(rotation[0]+90))*dt
137
138                                     vx=vx+linear_acceleration
    [0]*dt
139                                     vy=vy+linear_acceleration
    [1]*dt
140                                     vz=vz+linear_acceleration
    [2]*dt
141
142                                     v=np.array([vx,vy,vz])
143                                     #print("Velocity not
    filtered: ",v)
144                                     vHP=signal.sosfilt(sos,v)
145                                     #print("Velocity filtered
    : ",vHP)
146                                     #v=np.array([vx,vy,vz])
147
148                                     #v_old=v
149
150                                     #print(v)
151
152                                     #px=p_old[0] + v[0]*dt
153                                     #py=p_old[1] + v[1]*dt
154                                     #pz=p_old[2] + v[2]*dt
```

```
155
156                                     #p=np.array([px,py,pz])
157
158                                     #p_old=p
159
160                                     #px=px + vx*dt*1000
161                                     #py=py + vy*dt*1000
162
163                                     pxMOB=vHP[0]*dt
164                                     pyMOB=vHP[1]*dt
165                                     pzMOB=vHP[2]*dt
166
167                                     pMOB=np.array([pxMOB,
    pyMOB,pzMOB])
168
169
170
171                                     #print("Position not
    filtered",p)
172                                     pHP=signal.sosfilt(sos,
    pMOB)
173                                     #print("Position filtered
    : ",pHP)
174
175                                     rot_mat=
    quaternion_rotation_matrix(quaternions)
176
177                                     pFIX=list(map(add,pFIX,
    rot_mat@pHP))
178
179                                     print(pFIX)
180
181                                     #print(px,py)
182
183                 except KeyboardInterrupt:
184                     s.close()
185                     print('Program exit!')
186                     break
187         except :
188             s.close()
189             print("An error occurred")
190             print('Restart')
```

E.8 Raspberry 2D IMU code

```
1 import serial;
2 import io;
3 import time;
```

```
import os;

import numpy as np
from numpy.linalg import inv, norm
from operator import add
import math
from scipy import signal

if __name__ == '__main__' :
    start_serial=True

    dt=0.01
    p_old=np.array([0,0,0])
    v_old=np.array([0,0,0])

    #time.sleep(5)

    calibration=True
    cal=0

    px=0
    py=0

    vx=0
    vy=0

    sos=signal.butter(1,0.2/(1/dt),btype='highpass',analog=False,output='sos')
    while True:
        try :
            # configure the serial connections (the parameters differs on the device you are connecting to)
            with serial.Serial(port='/dev/ttyACM0', baudrate=115200, timeout=0.1,xonxoff=False, rtscts=False, dsrdtr=True) as s:
                try:
                    for line in s:
                        #t1 = time.time()
                        #print(line.decode("utf-8"))
                        data=line.decode("utf-8").split(",")
                        i=0
                        if len(data)>2:
                            for num in data:
                                data[i]=float(num)
                                i=i+1
                            #print(data)
                            quaternions=data[0:4]
                            #print(quaternions)
```

```
48                              rotation=data[4:7]
49                              #print(rotation)
50                              linear_acceleration=[x*9.8
    for x in data[7:10]]
51                              accelerometer_raw=[x*9.8 for
    x in data[10:13]]
52                              #print(accelerometer_raw)
53                              if calibration:
54                                  if cal==0:
55                                      acc_cal=[
    linear_acceleration[0],linear_acceleration[1],
    linear_acceleration[2]]
56                                      #acc_cal=[
    accelerometer_raw[0],accelerometer_raw[1],
    accelerometer_raw[2]]
57
58                                  else:
59                                      #acc_cal=[(acc_cal
    [0]+linear_acceleration[0])/2,(acc_cal[1]+
    linear_acceleration[1])/2,(acc_cal[2]+
    linear_acceleration[2])/2]
60                                      #acc_cal=[acc_cal[0]+
    accelerometer_raw[0],acc_cal[1]+accelerometer_raw[1],
    acc_cal[2]+accelerometer_raw[2]]
61                                      #acc_cal=list(map(add
    ,acc_cal,accelerometer_raw))
62                                      acc_cal=list(map(add,
    acc_cal,linear_acceleration))
63                                  cal=cal+1
64                                  if cal==500:
65                                      calibration=False
66                                      print(acc_cal)
67                                      acc_cal=[x/cal for x
    in acc_cal]
68                                      print(acc_cal)
69
70                                  #print(acc_cal)
71
72                              else:
73                                  #accelerometer_raw=[
    accelerometer_raw[0]-acc_cal[0],accelerometer_raw[1]-
    acc_cal[1],accelerometer_raw[2]-acc_cal[2]]
74                                  #print(acc_cal)
75                                  for i in range(3):
76                                      if abs(
    accelerometer_raw[i])<0.1:
77                                          accelerometer_raw
    [i]=0
78                                  #print(accelerometer_raw)
```

```
79                                  linear_acceleration=np.
    array([linear_acceleration[0]-acc_cal[0],
    linear_acceleration[1]-acc_cal[1],linear_acceleration
    [2]-acc_cal[2]])
80                                  #print("Non filtrato")
81                                  #print(
    linear_acceleration)
82
83                                  #linear_accelerationHP=
    signal.sosfilt(sos,linear_acceleration)
84                                  #print("Filtrato")
85                                  #print(
    linear_accelerationHP)
86
87                                  #vx=v_old[0] +
    linear_acceleration[0]*dt
88                                  #vy=v_old[1] +
    linear_acceleration[1]*dt
89                                  #vz=v_old[2] +
    linear_acceleration[2]*dt
90
91                                  #vx=(accelerometer_raw
    [0]*math.cos(rotation[0])+accelerometer_raw[1]*math.
    cos(rotation[0]+90))*dt
92                                  #vy=(accelerometer_raw
    [0]*math.sin(rotation[0])+accelerometer_raw[1]*math.
    sin(rotation[0]+90))*dt
93
94                                  vx=vx+(
    linear_acceleration[0]*math.cos(rotation[0])+
    linear_acceleration[1]*math.cos(rotation[0]+90))*dt
95                                  vy=vy+(
    linear_acceleration[0]*math.sin(rotation[0])+
    linear_acceleration[1]*math.sin(rotation[0]+90))*dt
96
97                                  v=np.array([vx,vy])
98                                  #print("Velocity not
    filtered: ",v)
99                                  vHP=signal.sosfilt(sos,v)
100                                 #print("Velocity filtered
    : ",vHP)
101                                 #v=np.array([vx,vy,vz])
102
103                                 #v_old=v
104
105                                 #print(v)
106
107                                 #px=p_old[0] + v[0]*dt
108                                 #py=p_old[1] + v[1]*dt
109                                 #pz=p_old[2] + v[2]*dt
```

```
110
111                                     #p=np.array([px,py,pz])
112
113                                     #p_old=p
114
115                                     #px=px + vx*dt*1000
116                                     #py=py + vy*dt*1000
117
118                                     px=px + vHP[0]*dt
119                                     py=py + vHP[1]*dt
120                                     p=np.array([px,py])
121                                     #print("Position not
    filtered",p)
122                                     pHP=signal.sosfilt(sos,p)
123                                     print("Position filtered:
     ",pHP)
124
125                                     #print(px,py)
126
127                 except KeyboardInterrupt:
128                     s.close()
129                     print('Program exit!')
130                     break
131         except :
132             s.close()
133             print("An error occurred")
134             print('Restart')
```

E.9 MATLAB IMU algorithm

```
1  clear all
2  close all
3  clc
4
5  %% Data acquisition
6
7  STM32F401RE = serialport("COM3",115200);
8
9  samples = 3000;
10
11 for i=1:samples
12     data = readline(STM32F401RE);
13     data_split = strsplit(data,',');
14     if length(data_split) < 2
15         i = i-1;
16     else
17         data_converted = str2double(data_split);
18         Data(i,:)=data_converted;
```

```
        if i>=1000
            disp(data_converted)
        end
    end
end

%% Data manipulation
ts=10/1000; %sempling time

%save data for x, y, z axis. /1000*9.81 to transform from
    milliG to m/s^2
%save gyr data for x, y, z axis /1000 to transform from
   milli-dps to dps
acc=[Data(:,1)*9.81, Data(:,2)*9.81, Data(:,3)*9.81];
gyr=[Data(:,4), Data(:,5), Data(:,6)];
mag=[Data(:,7), Data(:,8), Data(:,9)];


%Process data through AHRS algorithm (calcualte
   orientation)
R = zeros(3,3,length(acc));
ifilt = ahrsfilter('SampleRate', 1/ts,'OrientationFormat'
   ,'Rotation matrix');
R = ifilt(acc, gyr * (pi/180),mag);% gyr must be rad/s,
   acc must be m/s^2
for i = 1:length(acc)
    R(:,:,i) =  R(:,:,i)';
end

%Calculate 'tilt-compensated' accelerometer
tcAcc = zeros(size(acc));  % accelerometer in Earth frame

for i = 1:length(acc)
    tcAcc(i,:) = R(:,:,i) * acc(i,:)';
end

%Calculate linear acceleration in Earth frame (
   subtracting gravity)
linAcc = tcAcc - [zeros(length(tcAcc), 1), zeros(length(
   tcAcc), 1),
ones(length(tcAcc), 1)]*9.81;


%integrate to find velocity
linVel = zeros(size(linAcc));
for i = 2:length(linAcc)
    linVel(i,:) = linVel(i-1,:) + linAcc(i,:) * ts;
end
%high pass filter the velocity to remove drift
order = 1;
```

```
filtCutOff = 0.1;
[b, a] = butter(order, (2*filtCutOff)/(1/ts), 'high');
linVelHP = filtfilt(b, a, linVel);

%integrate to find position
linPos = zeros(size(linVel));
for i = 2:length(linVelHP)
    linPos(i,:) = linPos(i-1,:) + linVelHP(i,:) * ts;
end
%high pass filter the position to remove drift
order = 1;
filtCutOff = 0.1;
[b, a] = butter(order, (2*filtCutOff)/(1/ts), 'high');
linPosHP = filtfilt(b, a, linPos);

%plots
figure(1);
hold on;
plot(acc(:,1), 'r');
plot(acc(:,2), 'g');
plot(acc(:,3), 'b');
xlabel('sample');
ylabel('m/s^2');
title('Accelerometer');
legend('X', 'Y', 'Z');

% Plot
figure(2);
hold on;
plot(tcAcc(:,1), 'r');
plot(tcAcc(:,2), 'g');
plot(tcAcc(:,3), 'b');
xlabel('sample');
ylabel('m/s^2');
title('''Tilt-compensated'' accelerometer');
legend('X', 'Y', 'Z');

% Plot
figure(3);
hold on;
plot(linVelHP(:,1), 'r');
plot(linVelHP(:,2), 'g');
plot(linVelHP(:,3), 'b');
xlabel('sample');
ylabel('g');
title('High-pass filtered linear velocity');
legend('X', 'Y', 'Z');
% Plot
figure(4);
hold on;
```

```
plot(linPosHP(:,1), 'r');
plot(linPosHP(:,2), 'g');
plot(linPosHP(:,3), 'b');
xlabel('sample');
ylabel('m');
title('High-pass filtered linear position');
legend('X', 'Y', 'Z');


% viewer = HelperOrientationViewer('Title',{'with mag'})
% ifilt = ahrsfilter('SampleRate', 1/ts);
% for i=1:size(acc,1)
%     qimu = ifilt(acc(i,:), gyr(i,:) * (pi/180),mag(i,:));
%     viewer(qimu);
%
% end

SamplePlotFreq = 5;
samplePeriod=ts;
SixDOFanimation(linPosHP(1000:samples-1,:), R, ...
                'SamplePlotFreq', SamplePlotFreq, 'Trail', 'Off', ...
                'Position', [9 39 1280 720], ...
                'AxisLength', 0.1, 'ShowArrowHead', false, ...
                'Xlabel', 'X (m)', 'Ylabel', 'Y (m)', 'Zlabel', 'Z (m)', 'ShowLegend', false, 'Title', 'Unfiltered',...
                'CreateAVI', false, 'AVIfileNameEnum', false, 'AVIfps', ((1/samplePeriod) / SamplePlotFreq));
```

E.10 MATLAB plotting algorithm

```
function fig = SixDOFanimation(varargin)

    %% Create local variables

    % Required arguments
    p = varargin{1};                % position of body
    R = varargin{2};                % rotation matrix of body
    [numSamples dummy] = size(p);

    % Default values of optional arguments
    SamplePlotFreq = 1;
    Trail = 'Off';
    LimitRatio = 1;
```

```
14      Position = [];
15      FullScreen = false;
16      View = [30 20];
17      AxisLength = 1;
18      ShowArrowHead = 'on';
19      Xlabel = 'X';
20      Ylabel = 'Y';
21      Zlabel = 'Z';
22      Title = '6DOF Animation';
23      ShowLegend = true;
24      CreateAVI = false;
25      AVIfileName = '6DOF Animation';
26      AVIfileNameEnum = true;
27      AVIfps = 30;
28
29      for i = 3:2:nargin
30          if  strcmp(varargin{i}, 'SamplePlotFreq'),
    SamplePlotFreq = varargin{i+1};
31          elseif  strcmp(varargin{i}, 'Trail')
32              Trail = varargin{i+1};
33              if(~strcmp(Trail, 'Off') && ~strcmp(Trail, '
    DotsOnly') && ~strcmp(Trail, 'All'))
34                  error('Invalid argument.  Trail must be '
    'Off'', ''DotsOnly'' or ''All''.');
35              end
36          elseif  strcmp(varargin{i}, 'LimitRatio'),
    LimitRatio = varargin{i+1};
37          elseif  strcmp(varargin{i}, 'Position'), Position
     = varargin{i+1};
38          elseif  strcmp(varargin{i}, 'FullScreen'),
    FullScreen = varargin{i+1};
39          elseif  strcmp(varargin{i}, 'View'), View =
    varargin{i+1};
40          elseif  strcmp(varargin{i}, 'AxisLength'),
    AxisLength = varargin{i+1};
41          elseif  strcmp(varargin{i}, 'ShowArrowHead'),
    ShowArrowHead = varargin{i+1};
42          elseif  strcmp(varargin{i}, 'Xlabel'), Xlabel =
    varargin{i+1};
43          elseif  strcmp(varargin{i}, 'Ylabel'), Ylabel =
    varargin{i+1};
44          elseif  strcmp(varargin{i}, 'Zlabel'), Zlabel =
    varargin{i+1};
45          elseif  strcmp(varargin{i}, 'Title'), Title =
    varargin{i+1};
46          elseif  strcmp(varargin{i}, 'ShowLegend'),
    ShowLegend = varargin{i+1};
47          elseif  strcmp(varargin{i}, 'CreateAVI'),
    CreateAVI = varargin{i+1};
```

```
48          elseif  strcmp(varargin{i}, 'AVIfileName'),
    AVIfileName = varargin{i+1};
49          elseif  strcmp(varargin{i}, 'AVIfileNameEnum'),
    AVIfileNameEnum = varargin{i+1};
50          elseif  strcmp(varargin{i}, 'AVIfps'), AVIfps =
    varargin{i+1};
51          else error('Invalid argument.');
52          end
53      end;
54
55      %% Reduce data to samples to plot only
56
57      p = p(1:SamplePlotFreq:numSamples, :);
58      R = R(:, :, 1:SamplePlotFreq:numSamples) * AxisLength
    ;
59      if(numel(View) > 2)
60          View = View(1:SamplePlotFreq:numSamples, :);
61      end
62      [numPlotSamples dummy] = size(p);
63
64      %% Setup AVI file
65
66      aviobj = [];
                            % create null object
67      if(CreateAVI)
68          fileName = strcat(AVIfileName, '.avi');
69          if(exist(fileName, 'file'))
70              if(AVIfileNameEnum)
                            % if file name exists and enum
    enabled
71                  i = 0;
72                  while(exist(fileName, 'file'))
                                % find un-used file name by
    appending enum
73                      fileName = strcat(AVIfileName,
    sprintf('%i', i), '.avi');
74                      i = i + 1;
75                  end
76              else
                                % else file name exists and
    enum disabled
77                  fileName = [];
                                % file will not be created
78              end
79          end
80          if(isempty(fileName))
81              sprintf('AVI file not created as file already
     exists.')
82          else
```

```
83              aviobj = avifile(fileName, 'fps', AVIfps, '
    compression', 'Cinepak', 'quality', 100);
84          end
85      end
86
87      %% Setup figure and plot
88
89      % Create figure
90      fig = figure('NumberTitle', 'off', 'Name', '6DOF
    Animation');
91      if(FullScreen)
92          screenSize = get(0, 'ScreenSize');
93          set(fig, 'Position', [0 0 screenSize(3)
    screenSize(4)]);
94      elseif(~isempty(Position))
95          set(fig, 'Position', Position);
96      end
97      set(gca, 'drawmode', 'fast');
98      lighting phong;
99      set(gcf, 'Renderer', 'zbuffer');
100     hold on;
101     axis equal;
102     grid on;
103     view(View(1, 1), View(1, 2));
104     title(i);
105     xlabel(Xlabel);
106     ylabel(Ylabel);
107     zlabel(Zlabel);
108
109     % Create plot data arrays
110     if(strcmp(Trail, 'DotsOnly') || strcmp(Trail, 'All'))
111         x = zeros(numPlotSamples, 1);
112         y = zeros(numPlotSamples, 1);
113         z = zeros(numPlotSamples, 1);
114     end
115     if(strcmp(Trail, 'All'))
116         ox = zeros(numPlotSamples, 1);
117         oy = zeros(numPlotSamples, 1);
118         oz = zeros(numPlotSamples, 1);
119         ux = zeros(numPlotSamples, 1);
120         vx = zeros(numPlotSamples, 1);
121         wx = zeros(numPlotSamples, 1);
122         uy = zeros(numPlotSamples, 1);
123         vy = zeros(numPlotSamples, 1);
124         wy = zeros(numPlotSamples, 1);
125         uz = zeros(numPlotSamples, 1);
126         vz = zeros(numPlotSamples, 1);
127         wz = zeros(numPlotSamples, 1);
128     end
129     x(1) = p(1,1);
```

```
130     y(1) = p(1,2);
131     z(1) = p(1,3);
132     ox(1) = x(1);
133     oy(1) = y(1);
134     oz(1) = z(1);
135     ux(1) = R(1,1,1:1);
136     vx(1) = R(2,1,1:1);
137     wx(1) = R(3,1,1:1);
138     uy(1) = R(1,2,1:1);
139     vy(1) = R(2,2,1:1);
140     wy(1) = R(3,2,1:1);
141     uz(1) = R(1,3,1:1);
142     vz(1) = R(2,3,1:1);
143     wz(1) = R(3,3,1:1);
144 
145     % Create graphics handles
146     orgHandle = plot3(x, y, z, 'k.');
147     if(ShowArrowHead)
148         ShowArrowHeadStr = 'on';
149     else
150         ShowArrowHeadStr = 'off';
151     end
152     quivXhandle = quiver3(ox, oy, oz, ux, vx, wx,  'r', '
    ShowArrowHead', ShowArrowHeadStr, 'MaxHeadSize',
    0.999999, 'AutoScale', 'off');
153     quivYhandle = quiver3(ox, oy, oz, uy, vy, wy,  'g', '
    ShowArrowHead', ShowArrowHeadStr, 'MaxHeadSize',
    0.999999, 'AutoScale', 'off');
154     quivZhandle = quiver3(ox, ox, oz, uz, vz, wz,  'b', '
    ShowArrowHead', ShowArrowHeadStr, 'MaxHeadSize',
    0.999999, 'AutoScale', 'off');
155 
156     % Create legend
157     if(ShowLegend)
158         legend('Origin', 'X', 'Y', 'Z');
159     end
160 
161     % Set initial limits
162     Xlim = [x(1)-AxisLength x(1)+AxisLength] * LimitRatio
    ;
163     Ylim = [y(1)-AxisLength y(1)+AxisLength] * LimitRatio
    ;
164     Zlim = [z(1)-AxisLength z(1)+AxisLength] * LimitRatio
    ;
165     set(gca, 'Xlim', Xlim, 'Ylim', Ylim, 'Zlim', Zlim);
166 
167     % Set initial view
168     view(View(1, :));
169 
170     %% Plot one sample at a time
```

```
171
172     for i = 1:numPlotSamples
173
174         % Update graph title
175         if(strcmp(Title, ''))
176             titleText = sprintf('Sample %i of %i', 1+((i
    -1)*SamplePlotFreq), numSamples);
177         else
178             titleText = strcat(Title, ' (', sprintf('
    Sample %i of %i', 1+((i-1)*SamplePlotFreq), numSamples
    ), ')');
179         end
180         title(titleText);
181
182         % Plot body x y z axes
183         if(strcmp(Trail, 'DotsOnly') || strcmp(Trail, '
    All'))
184             x(1:i) = p(1:i,1);
185             y(1:i) = p(1:i,2);
186             z(1:i) = p(1:i,3);
187         else
188             x = p(i,1);
189             y = p(i,2);
190             z = p(i,3);
191         end
192         if(strcmp(Trail, 'All'))
193             ox(1:i) = p(1:i,1);
194             oy(1:i) = p(1:i,2);
195             oz(1:i) = p(1:i,3);
196             ux(1:i) = R(1,1,1:i);
197             vx(1:i) = R(2,1,1:i);
198             wx(1:i) = R(3,1,1:i);
199             uy(1:i) = R(1,2,1:i);
200             vy(1:i) = R(2,2,1:i);
201             wy(1:i) = R(3,2,1:i);
202             uz(1:i) = R(1,3,1:i);
203             vz(1:i) = R(2,3,1:i);
204             wz(1:i) = R(3,3,1:i);
205         else
206             ox = p(i,1);
207             oy = p(i,2);
208             oz = p(i,3);
209             ux = R(1,1,i);
210             vx = R(2,1,i);
211             wx = R(3,1,i);
212             uy = R(1,2,i);
213             vy = R(2,2,i);
214             wy = R(3,2,i);
215             uz = R(1,3,i);
216             vz = R(2,3,i);
```

```
217             wz = R(3,3,i);
218         end
219         set(orgHandle, 'xdata', x, 'ydata', y, 'zdata', z
    );
220         set(quivXhandle, 'xdata', ox, 'ydata', oy, 'zdata
    ', oz,'udata', ux, 'vdata', vx, 'wdata', wx);
221         set(quivYhandle, 'xdata', ox, 'ydata', oy, 'zdata
    ', oz,'udata', uy, 'vdata', vy, 'wdata', wy);
222         set(quivZhandle, 'xdata', ox, 'ydata', oy, 'zdata
    ', oz,'udata', uz, 'vdata', vz, 'wdata', wz);
223
224         % Adjust axes for snug fit and draw
225         axisLimChanged = false;
226         if((p(i,1) - AxisLength) < Xlim(1)), Xlim(1) = p(
    i,1) - LimitRatio*AxisLength; axisLimChanged = true;
    end
227         if((p(i,2) - AxisLength) < Ylim(1)), Ylim(1) = p(
    i,2) - LimitRatio*AxisLength; axisLimChanged = true;
    end
228         if((p(i,3) - AxisLength) < Zlim(1)), Zlim(1) = p(
    i,3) - LimitRatio*AxisLength; axisLimChanged = true;
    end
229         if((p(i,1) + AxisLength) > Xlim(2)), Xlim(2) = p(
    i,1) + LimitRatio*AxisLength; axisLimChanged = true;
    end
230         if((p(i,2) + AxisLength) > Ylim(2)), Ylim(2) = p(
    i,2) + LimitRatio*AxisLength; axisLimChanged = true;
    end
231         if((p(i,3) + AxisLength) > Zlim(2)), Zlim(2) = p(
    i,3) + LimitRatio*AxisLength; axisLimChanged = true;
    end
232         if(axisLimChanged), set(gca, 'Xlim', Xlim, 'Ylim'
    , Ylim, 'Zlim', Zlim); end
233         drawnow;
234
235         % Adjust view
236         if(numel(View) > 2)
237             view(View(i, :));
238         end
239
240         % Add frame to AVI object
241         if(~isempty(aviobj))
242             frame = getframe(fig);
243             aviobj = addframe(aviobj, frame);
244         end
245
246     end
247
248     hold off;
249
```

```
250     % Close AVI file
251     if(~isempty(aviobj))
252         aviobj = close(aviobj);
253     end
254 
255 end
```

Bibliography

[1] T.-H. Tsai and C.-H. Yao, "A Real-time Tracking Algorithm for Human Following Mobile Robot," in *2018 International SoC Design Conference (ISOCC)*, (Daegu, Korea (South)), pp. 78–79, IEEE, Nov. 2018.

[2] S. Jiang, L. Li, M. Hang, and T.-y. Kuc, "An Adaptive 2D Tracking Approach for Person Following Robot," in *2017 International Symposium on Computer Science and Intelligent Controls (ISCSIC)*, (Budapest), pp. 147–151, IEEE, Oct. 2017.

[3] Q. K. Dang and Y. S. Suh, "Human-following robot using infrared camera," in *2011 11th International Conference on Control, Automation and Systems*, pp. 1054–1058, 2011.

[4] M. S. Hassan, A. F. Khan, M. W. Khan, M. Uzair, and K. Khurshid, "A computationally low cost vision based tracking algorithm for human following robot," in *2016 2nd International Conference on Control, Automation and Robotics (ICCAR)*, (Hong Kong, Hong Kong), pp. 62–65, IEEE, Apr. 2016.

[5] L. Jiang, W. Wang, Y. Chen, and Y. Jia, "Personalize Vison-based Human Following for Mobile Robots by Learning from Human-Driven Demonstrations," in *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, (Nanjing), pp. 726–731, IEEE, Aug. 2018.

[6] P. Petrov, V. Georgieva, I. Kralov, and S. Nikolov, "An Adaptive Control Scheme for Human Following Behavior of Mobile Robots," in *2020 XI National Conference with International Participation (ELECTRONICA)*, (Sofia, Bulgaria), pp. 1–4, IEEE, July 2020.

[7] Yoonchang Sung and Woojin Chung, "Human tracking of a mobile robot with an onboard LRF (Laser Range Finder) using human walking motion analysis," in *2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, (Incheon), pp. 366–370, IEEE, Nov. 2011.

[8] "TensorFlow Lite | ML per dispositivi mobili e periferici." TensorFlow Lite site.

[9] "Intel® Neural Compute Stick 2." NCS2 site.

[10] "OpenVINO™ Toolkit Overview - OpenVINO™ Toolkit." Openvino site.

[11] "Opencv." OpenCV site.

[12] "2-DOF Pan-Tilt HAT - Waveshare Wiki." 2-DOF Waveshare Wiki.