UNIVERSITY OF TRENTO

Department of Industrial Engineering

Master in Autonomous Systems

Distributed Systems for Measurement and Automation

Localization of mobile robots using Kalman filtering

Authors:
Marco Moletta
Aristeidis Noulis
Jonathan Smyth

Supervisor:
Daniele Fontanelli, DII

December 12, 2019

1 Introduction and Objectives

The objective of this project was to implement a distributed system of mobile robots and use the Kalman filter to fuse data from a exteroceptive and proprioceptive sensor source. The idea was to use this information to localize the robots as they move within a grid.

Both robots are equipped with an Inertial Measurement Unit as their respective proprioceptive sensor. Two different exteroceptive sensors were then tested and compared. Wifi RSSI and a camera from a birds eye view.

This report outlines the implementation of the project, the experiments conducted, a comparison of the results and some final conclusions and consideration.

2 System Setup

In the system, the frames of 2 mBot Educational Robots have been used and more specifically the base of the robot and their motors. Moreover, on the top of these frames have been placed the following:

- Two Arduino Uno for supporting the power supply of the motors and two simple bread boards, to expand the connectivity.
- Two MPU-6050 sensors. These two IMU sensors, containing both an accelerometer and a gyroscope, are the propioceptive sensors placed on the robots, between the two wheels.
- Two NodeMCU ESP8266 - E12E, which contains the wifi module that allows the flow of data between the MQTT broker (server) and the robots. Moreover, all the code about Kalman, movements and sensor readings of the robots is executed here.
- On the top of both the robots there are 2 black circles (one bigger on the back and on smaller on the front) with white background, so to be recognizable from the camera and be able to get the orientation.

Furthermore, on the three of the four corners of the rectangle system terrain 3 Raspberries Pi 3 have been placed, which are emitting WiFi signals, useful for the triangulation of RSSI. The distance between the WiFi nodes is 2 and 2.5 meters respectively on x and y, (the same dimensions of the camera field of view) which creates the final rectangle in where the robots are moving and localize themselves.
Last but not least, is the usage of a camera on the ceiling, to observe the whole system, recognize the terrain and the robots, and provide the data for the robots' localization.
The MQTT broker for the communication is also implemented on the Raspberry Pi.

3 System Model

3.1 Kinematic Model

The kinematic model of system can be represented with the following sketch and equations:

$$x_{k+1}{}^{\mathrm{F}} = x_k{}^{\mathrm{F}} + v_k{}^{\mathrm{R}} \cos(\theta_k) \cdot dt \quad (1)$$

$$y_{k+1}{}^{\mathrm{F}} = y_k{}^{\mathrm{F}} + v_k{}^{\mathrm{R}} \sin(\theta_k) \cdot dt \quad (2)$$

$$v_{k+1}{}^{\mathrm{R}} = v_k{}^{\mathrm{R}} + a_k{}^{\mathrm{R}} \cdot dt \quad (3)$$

$$\theta_{k+1}{}^{\mathrm{F}} = \theta_k{}^{\mathrm{F}} + \omega_k{}^{\mathrm{R}} \cdot dt \quad (4)$$

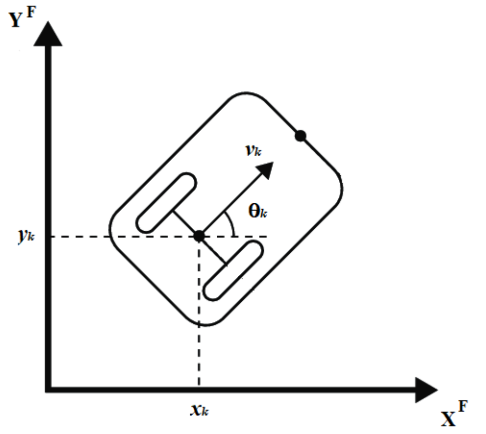

Figure 1: Kinematic model of the robot

Where x_k, y_k, v_k, θ_k are the components of the state vector at every k iteration of the Kalman (F if it is related to fixed frame, R if it is related to robot frame), a_k and ω_k are respectively the input acceleration (in forward direction of the robot frame) and the input angular velocity provided by the IMU, dt is the time (theoretically infinitesimal) between iteration 'x' and '$x+1$'.

3.2 Kalman Filter

Known the model, it is possible to proceed with the implementation of the two Kalman filters: one for RSSI and IMU, the other one for Camera and IMU. The matrices of the Prediction step (where IMU data are used) of both the Kalman algorithms will be:

$$A = \begin{bmatrix} 1 & 0 & \cos(\theta_k)\cdot dt & 0 \\ 0 & 1 & \sin(\theta_k)\cdot dt & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ dt & 0 \\ 0 & dt \end{bmatrix}$$

$$state = \begin{bmatrix} x_k \\ y_k \\ v_k \\ \theta_k \end{bmatrix} u = \begin{bmatrix} a_k \\ \omega_k \end{bmatrix} Q_{imu} = \begin{bmatrix} \sigma^2_{a_{(imu)}} & 0 \\ 0 & \sigma^2_{\omega_{(imu)}} \end{bmatrix}$$

For the Update step of the Kalman filter for camera and IMU setup, the following matrices have been used.

$$H_{cam} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} z_{cam} = \begin{bmatrix} x_{cam} \\ y_{cam} \\ \theta_{cam} \end{bmatrix}$$

$$R_{cam} = \begin{bmatrix} \sigma^2_{x_{(cam)}} & 0 & 0 \\ 0 & \sigma^2_{y_{(cam)}} & 0 \\ 0 & 0 & \sigma^2_{\theta_{(cam)}} \end{bmatrix}$$

Where H_{cam} is the matrix of the measurement model of the camera (linear for x, y and θ), Q_{imu} and R_{cam} are the covariance matrices for IMU and camera and z_{cam} is the vector of camera measurements.
For the Update step of the Kalman filter for RSSI and IMU setup, the following matrices have been used.

$$\rho_{pred} = \begin{bmatrix} \sqrt{(x_k - x_{(nodeA)})^2 + (y_k - y_{(nodeA)})^2)} \\ \sqrt{(x_k - x_{(nodeB)})^2 + (y_k - y_{(nodeB)})^2)} \\ \sqrt{(x_k - x_{(nodeC)})^2 + (y_k - y_{(nodeC)})^2)} \end{bmatrix}$$

$$H_{RSSI} = \begin{bmatrix} \frac{x_k - x_{(nodeA)}}{\rho_{pred_1}} & \frac{y_k - y_{(nodeA)}}{\rho_{pred_1}} & 0 & 0 \\ \frac{x_k - x_{(nodeB)}}{\rho_{pred_2}} & \frac{y_k - y_{(nodeB)}}{\rho_{pred_2}} & 0 & 0 \\ \frac{x_k - x_{(nodeC)}}{\rho_{pred_3}} & \frac{y_k - y_{(nodeC)}}{\rho_{pred_3}} & 0 & 0 \end{bmatrix}$$

$$z_{RSSI} = \begin{bmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \end{bmatrix} R_{RSSI} = \begin{bmatrix} \sigma^2_{\rho_1} & 0 & 0 \\ 0 & \sigma^2_{\rho_2} & 0 \\ 0 & 0 & \sigma^2_{\rho_3} \end{bmatrix}$$

Where ρ_{pred} is the matrix containing the result of the prediction step (which then is updated with the three new measurements $\rho_{1,2,3}$) and the other matrices are equivalent to the camera matrices.

3.2.1 Variances calculations

In an attempt to ensure the calculation of the variances of the different sensors (then inserted in the Kalman filter matrices) were as realistic as possible, various measurements were taken for each sensor, trying to reproduce the same noise conditions that will be present when the robot moves inside the rectangle. In particular, for the Wifi modules the measurements have been taken respectively by all 3 nodes in the exact positions to ensure continuity in the calibration process. The acceleration measurements were instead taken while the robot was moving in order to consider the noise due to the possible non-uniformity of the ground (which in the case of the experiments was on white bed sheets), under the assumption that the speed was constant. A similar procedure was carried out for the gyroscope, by rotating at a constant speed, while for the measurements of the variance in x, y and theta of the camera the robot was simply kept still inside the rectangle.

3.3 Localization using Camera

As an exteroceptive means of localizing the robots within a grid, the idea to use a camera from a birds eye view was decided upon. Allowing a 2D localizing the robots in a the x,y planes and thus being able to derive the orientation of the robots.

3.3.1 OpenCV

A RaspberryPi3 and a camera were used as the hardware for the exteroceptive sensing. The development environment was set up with OpenCV in python, this computer vision framework was used to identify the robots in the video frames using shape detection to attain the orientation and position of the robots.

Utilizing some well known computer vision methodologies to filter the images, in an attempt to optimize the image processing to best detect the shapes.

```
gray = cv2.cvtColor(image,
    cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (9, 9), 0)
```

First, the incoming frames from the video were grayed and a Gaussian blur was applied with a standard deviation to X and Y kernel components of 9, trial and error of the detection algorithm proved these values were optimal for the given tests. This helped to smooth the images and try remove any Gaussian distributed noise for the further steps of processing.

Two different approaches were taken for the next step, initially it was thought to use the HarrisCorner Algorithm to detect the images of the shapes, preliminary trials with this algorithm proved promising. However, further into development when real-time tests were conducted it was realized that a combination of poor quality images and noise from the camera meant that the corner detection was too acute, as the robots moved around in the frame, the algorithm produced more corners that were actually one the shape, thus messing up the computation on the orientation and centre points of the robots.

An attempt to reduce this was to play around with the blurring of the images in pre-processing and with the parameters of the HarrisCorner algorithm, however in the end it was decided to change approach, due to the problems in the real time image processing.

```
    edged = cv2.Canny(gray, 10, 160)
    edged = cv2.erode(edged, None,
        iterations=1)
    edged = cv2.dilate(edged, None,
        iterations=1)
    cnts = cv2.findContours(edged.copy(),
        cv2.RETR_EXTERNAL,
         cv2.CHAIN_APPROX_SIMPLE)
```

The next attempt was to implement detection of the contours in the images, and use this methodology to detect the shapes based on the amount of contours each given shape had. This was achieved by implementing the Canny algorithm, which is a popular edge detection algorithm. This algorithm works by computing the edge gradients of the pixels in the X and Y directions. After this is applied to the image, the process was to erode the image to allow for further noise reduction, then dilate it to re enlarge the areas of interest (the shapes). This result was then fed into a 'findContours()' function with 'cv2.CHAIN_APPROX_SIMPLE' parameter to reduce redundant points and save memory.
The final step at this point was to compute the shape detection, based on the amount of contours the shape had. It was decided that any polygon with more than 6 sides would be declared as a circle, this was to further reduce the likelihood of the image being distorted during movement which had been seen in prior testing with real time images. The results yielded were very good, the algorithm was successful in detecting the circles in the grid and even when there was slight distortion on the contours of the images, the algorithm proved robust and was still able to attain good centre points of the image.
The next step was to give some context to the shapes in the grid. To localize the shapes position one shape was sufficient, however to give some rotational orientation with respect to the reference frame (0,0) it was decided that 2 shapes per robot would be used. The front and back of the robots were signified by a small and large circle, respectively. Using this approach allowed for the mid point of the robot to be computed and then the orientation could then be computed as the angular difference as below: Once the ori-

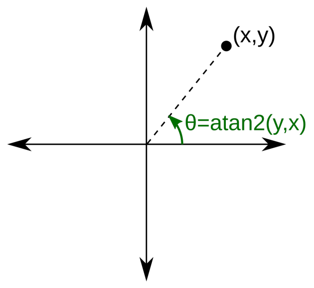

Figure 2: Arctan2 for Orientation

entation and position is computed in the pixel frame of the image, the next step was to calculate the real world position to give to the robots.

Some issue were encountered along the way, due to the sensitivity of the camera to lighting conditions and the texture of the floor in which the experiments were conducted. Due to these factors, unavoidable noise was introduced into the system, thus creating issues with the detection of the shapes. To combat this, the idea to create a vast contrast in the shapes and their background. This was achieved by putting a white sheet on the ground to enhance the contrast. See below, how the translation from the pixel domain to the real world domain was achieved.

```
pHeight=600
pWidth=800
metricH=2.00
metricW=2.50
xmp=2.00/pHeight
ymp=2.50/pWidth
```

The communication between the robots and the central node was to pass position and orientation. All X and Y positions which are sent and received make use of the above 'xmp' and 'ymp' metrics, which is the translation in the pixel domain to real world domain, this idea was used to enable the IMU to be initialized with real world values, allowing for a more understandable feedback from the experimental data.

The algorithm then with the post Kalman computational values of X,Y and θ from the robots would then plot on the frame where at that time instance the robots thought it was. However, due to the delay in the algorithm processing time, this was not always the case, the results of this can be seen in the attached videos.

3.4 Distributed Communication

MQTT is a lightweight IoT protocol, which is implemented on top of the TCP/IP layer. It was implemented for messaging over low bandwidth, high-latency or unreliable networks, with minimal code size needed. The communication protocol works with clients subscribing and publishing to different topics held on a broker (server). Multiple clients can subscribe/publish to the same/multiple topics. It has minimal packet overhead, compared to protocols like HTTP, and clients are easy to implement.

Some consideration need to be taken into account when using this communication protocol. MQTT can be used in both synchronous/asynchronous behaviour. The experiment was setup using asynchronous behaviour. This meant that the communication was being handled on a separate thread to the main program and the values were updated as and when they were received, this was decided upon to reduce the amount of latency measurements and updates.

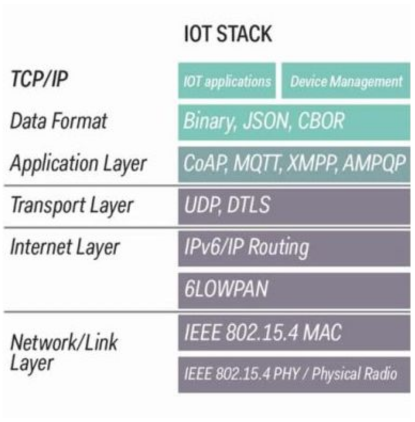

Figure 3: MQTT OSI Model Stack [2]

3.4.1 Latency measurements

In an attempt to understand what the overall latency of the communication across the system was, some experiments were conducted to work out the length of time taken from a message leaving the robots until it was received by the broker. A number of different experiments were conducted, these measurement were attained by connecting a real time clock to the robots and to the central node. These clocks were not synchronized, and the below information is based on the absolute time elapsed since the timer started until the timer is stopped. This was analyzed by starting a timer in the code before the intended function/feature was ran and then directly after the function finished. The real time clocks used were interfaced to the hardware via I2C, which is not exactly the fastest protocol in the world, however given the small data packet being sent across the bus, this latency has been taken as negligible in the below tests. The below table show averaged measurements based on 50 tests.

Test	**Time**
Robot to Broker	122 ms
Camera Algorithm	443 ms
Kalman Processing (dt)	28 ms
End to End	593 ms

Table 1: Latency measurements

3.5 RSSI

One of the localization methods, that has been implemented in the system, uses Radio Signal Strength Indication (RSSI), which is mostly selected as the sensor localization method, especially in indoor environments.

Having placed the three WiFi nodes on the corners of the terrain, and using the WiFi modules on the top of the robots, the estimation of the location of each has been done by converting the signal strength to distance.

According to this paper [1], this useful equation occurs:

$$RSSI = -(10 * n)log10(d) - A \tag{5}$$

The RSSI is the radio signal strength indicator in dBm, n is the signal propagation constant or path-loss exponent, d is the relative distance between the communicating nodes, and A is a reference received signal strength in dBm (the RSSI value measured when the separation distance between the receiver and the transmitter is one meter).

From the equation (5) this formula occurs easily:

$$d = 10^{(A-RSSI)/(10*n)} \tag{6}$$

In this way, a distance approach from the WiFi node that sends a signal has been extracted.

In order to acquire the value A, one of the WiFi modules that are used on the robots, has been placed in one meter distance from one WiFi node. The mean value of 300 RSSI values has been taked and the result was this:

$$A = -50.57dB \tag{7}$$

The n in the system is equal to 2 as the terrain is a free space without obstacles.

Summing up, this method ends up with this distance formula in respect to meters:

$$d = 10^{(A-RSSI)/20} \tag{8}$$

3.6 System Structure

The figure 4 describes the relationships in the system. There are two types of connections.
Directional: The robots receiving the RSSI values from the WiFi nodes.
BiDirectional: The camera is sending position data to robots but also receiving filtered position data from the robots.

4 Experiments and results

4.1 1 robot using RSSI and IMU processed through Kalman filter

The first localization experiment was done on the Kalman filter of RSSI (used only for position measurements) and IMU, using a single stationary robot and trying to locate it within the frame (node A, node B, node C, visible in Figure 5). The results are not satisfying, given the

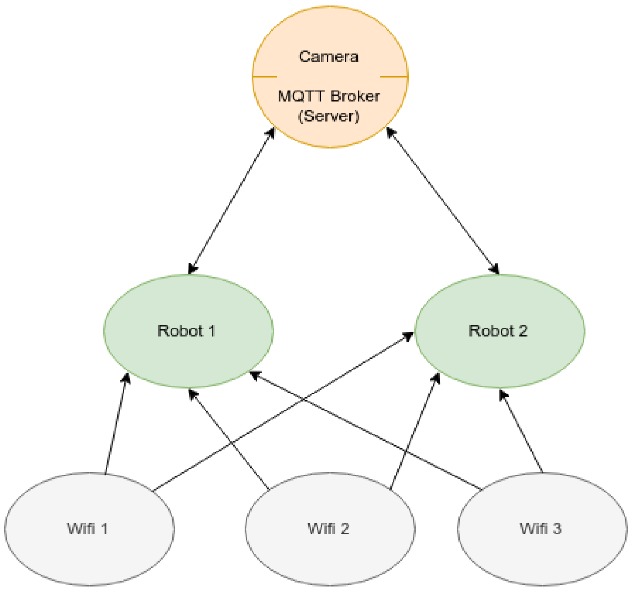

Figure 4: Logic model of the system.

poor accuracy of the measurement and the huge variances of the sensors. In fact it is possible to notice for Figure 6 how the measurements have a range of about 7m, which therefore is absolutely insufficient for the proper localization within the frame of 2 x 2,5m. Due to this fact, it has been avoided to go further on the experiments with the RSSI (moving the robot or putting 2 robots on the frame).
An interesting thing to be noticed, however, is the drift due to the integration on accelerometer and gyroscope data (visible in Figure 6 on the inclination of the lines). It is clear how for the accelerometer the variation is about 0.2-0.3 meters every 1000 iterations, while for the gyroscope the variation is considerably smaller (more or less 0.02 radiants (about 1 degree) every 1000 iterations) due to both the fact that the gyroscope has itself a much smaller variance than the accelerometer, both to the fact that the accelerometer data are integrated 2 times to obtain a distance measurement, while those of the gyroscope only once.
It must be said that in this case the initial orientation with respect to the fixed frame is not present, given that it would be necessary to have at least 2 quite precise measurements of 2 different positions from RSSI in order to calculate, with the proper trigonometric functions, the initial orientation of the robot.

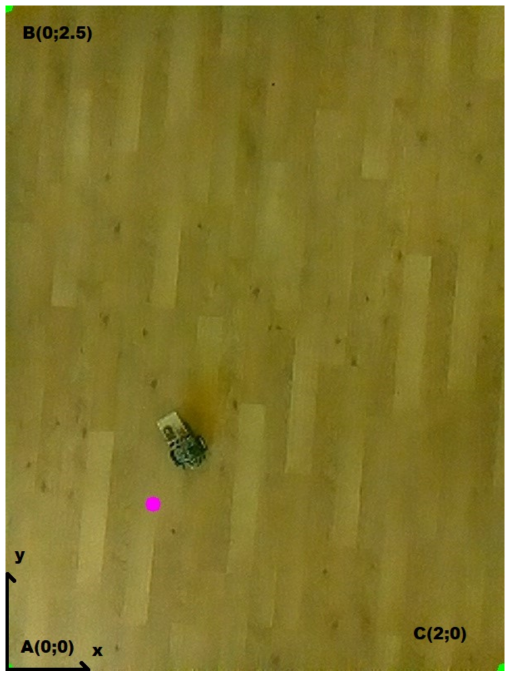

Figure 5: Picture taken by the camera of the RSSI setup.The purple dot represents where the kalman filter thinks the robot is (frequently the dot is outside the frame).

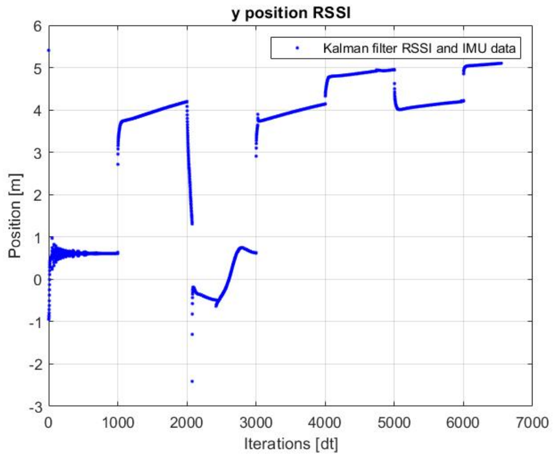

Figure 6: Example of data got by Kalman filter of RSSI and IMU while the robot was still.

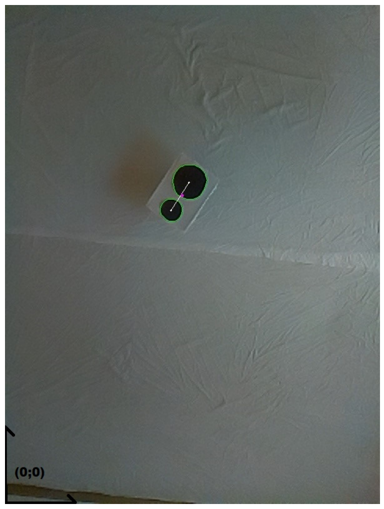

Figure 7: Picture of the shape detection got by the OpenCV algorithm. The purple dot shows where the Kalman filter thinks the robot is.

4.2 1 robot using Camera and IMU processed through Kalman filter

In the first test with the camera a single robot was moving quite randomly inside the frame (figure 7). From the measurements obtained it can be seen how, in this case, the camera measurements are predominant in the fusion, given that the shape of the curve is stepwise, a symptom of the fact that the Kalman data remains more or less unchanged in the fusion until a new camera measure is available (figure 8). In fact, reading for example the data of the position respect to the x-axis, it can be noticed that between two camera data the oscillation of the value due to the IMU is on the sixth decimal digit, due to the huge difference in terms of variance between the camera and the accelerometer (for the angle data the difference is on the fourth decimal digit).

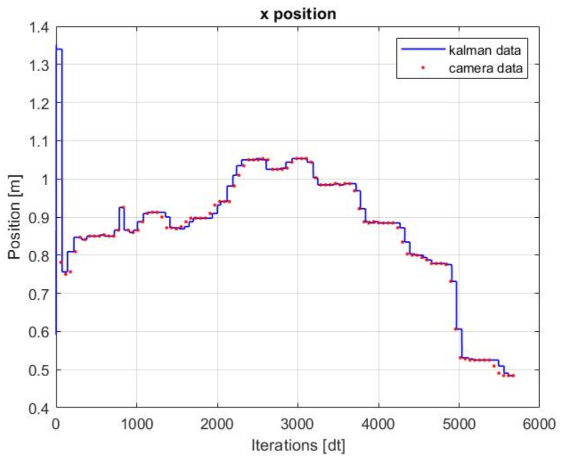

Figure 8: Example of data got by Kalman filter of Camera and IMU while the robot is moving randomly on the frame (x position).

4.3 2 robots playing the game Guard and Walker.

For the last experiments the two robots have been placed on the two opposite sides of the ter-

rain in such a way that their directions are opposed. One is the guard and its role is to turning right passing from range [0-π] to [π-2π] staying on each range for a random time between 0.5 and 2 seconds. The other robot is the walker and its goal is to move straight and cross the whole terrain area until it will reach the guard robot when the guard will not look at it (in other words, the two robots will not face each other) (figure 9). The walker is constantly checking when the received guard's angle from the server is within the safe range to walk ([0-π] that is the range in which the guard robot is looking on the opposite direction). When this condition is true, then the walker is moving straight until to reach the guard's side (figure 10).
The experiment was quite successful, as showed in graphs: the walker is moving (changing x and y position as shown in Figure 11) more or less between 1100-1300 and 2500-3200 iterations, that is more or less the range when the guard's angle is between 0 and π. Obviously the precision is not very high and there are some little delays or errors on the measurements.

5 Conclusion: limitations and considerations

In conclusion, the project had some position outcomes and some things which could be done differently. First of all, using only the IMU for localization is not a very good idea, since it has been shown that its variance is too high to have a proper utilization of its data in the fusion. So, for instance, coupling the IMU with some rotary encoders for the wheels would have yielded much greater results.
It can be said that there are some limitations which caused the overall results to be not as good

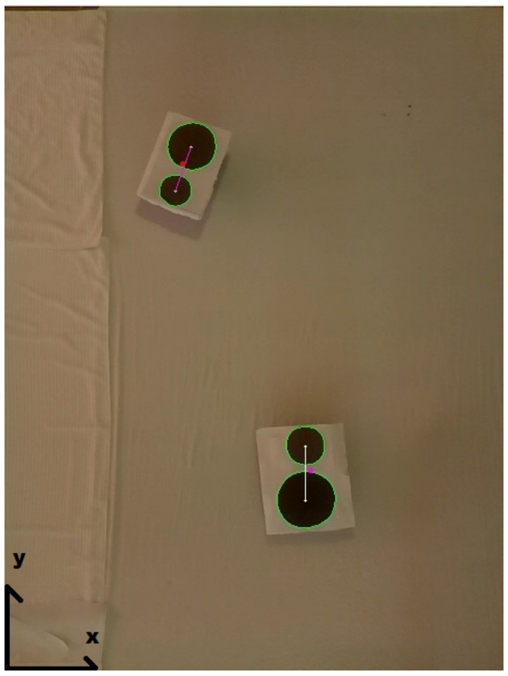

Figure 9: Picture of the camera during the experiment.

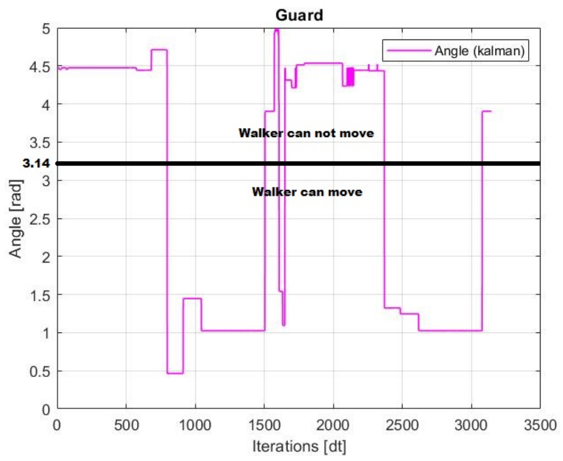

Figure 10: Variation of the orientation of the guard robot during the experiment.

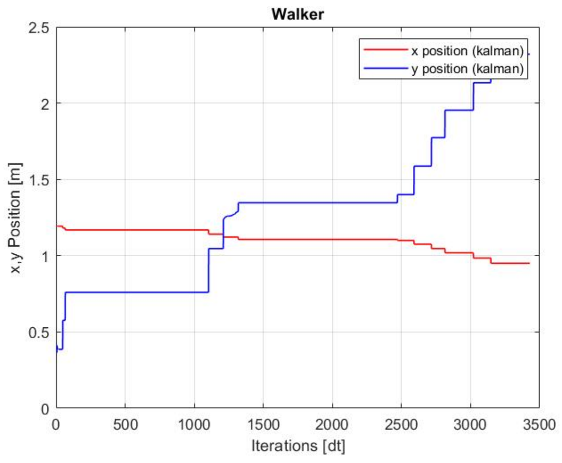

Figure 11: Variation of the position of the walker robot during the experiment.

as could have been. From a hardware point of view, the Raspberry Pi lacked the computational ability to deal with the computer vision at a real time rate, this can be seen in the videos, in contrast to this, the real world data which was attained and plotted in graphical form shows that indeed the Kalman filter was working well. The camera however, which has the capability to capture frames at 60 FPS, could not complete its loop cycle in 1/60 seconds, thus the lag is present in the videos.
Overall, the solution has proven it can work, in the future more robots could be added to perform more complex simulations, or other sensors (for example other cameras to grow the robot's work area).

References

[1] Qian Dong and Waltenegus Dargie. Evaluation of the reliability of rssi for indoor localization. *2012 International Conference on Wireless Communications in Underground and Confined Areas*, pages 1–6, 2012.

[2] Nicolas Windpassinger. What is series (1): what is the osi reference model ?, 2018.