# Sensor Fusion Final Project

Eugeniu Vezeteu, Ye Zhicheng, Xiong Weijiang
Aalto University - Electronic Engineering
ESPOO, Finland
vezeteu.vezeteu@aalto.fi,zhicheng.ye@aalto.fi,weijiang.xiong@aalto.fi

## ABSTRACT

*We are team number 12*
*This is the final year project for the Basics of Sensor fusion course, at Aalto University*
*In this project, we experimented with IMU, Camera and odometry sensors. We performed the camera and IMU calibration, after which we used sensor measurements to estimate the 2D car position and orientation based on detected QR codes. In the last part of the report, we developed filtering algorithms to perform the car tracking.*

## INTRODUCTION

The aim of this project is to develop an algorithm for tracking an autonomous robot by using a set of sensors. The robot, a Diddy-Borg rover-type robot, is programmed to follow a black line inside a closed area surrounded by walls. The robot is equipped with an inertial measurement unit (IMU), which is a combination of accelerometer, gyroscope, and magnetometer. In addition to the IMU, the robot is also equipped with an infra red detector, a motor controller, and a camera module.
In the first part (Sensing model) we present the IMU, camera calibration and motor control. The second part is dealing the localization. The tracking with IMU with/without camera, is presented in part 3.

## 1 PART - SENSOR MODELING

### Task 1 - Static IMU experiment

*Task 1a.* On this task, we plotted the data from given .csv file. The data was recorded in a static case. In Figure 1 we present the linear acceleration on all 3 axis. We conclude that the car is not moving. Also, the rotation (roll and pitch) angles are zero. We plotted the roll and pitch angles in Figure 2 and a conversion to polar coordinate in Figure 3. In the static case the accelerometer measures gravitational acceleration only.

The data that comes from gyroscope and magnetometer is plotted in Figures 4 and 5. Although the readings are quite noisy, we still observe that the robot does not have significant movement.

*Task 1b.* Data obtained from the gyroscopes often show a small offset in the average signal output, even when there is no movement. Therefore, bias can be estimated by the average sensor's output. We computed gyroscope bias by:

```
x = np.array(self.degree_xyz.iloc[:, 0]).mean()
y = np.array(self.degree_xyz.iloc[:, 1]).mean()
z = np.array(self.degree_xyz.iloc[:, 2]).mean()

self.bias = [x, y,z]
```

Result is $bias = [0.085, 0.222, -0.064]$

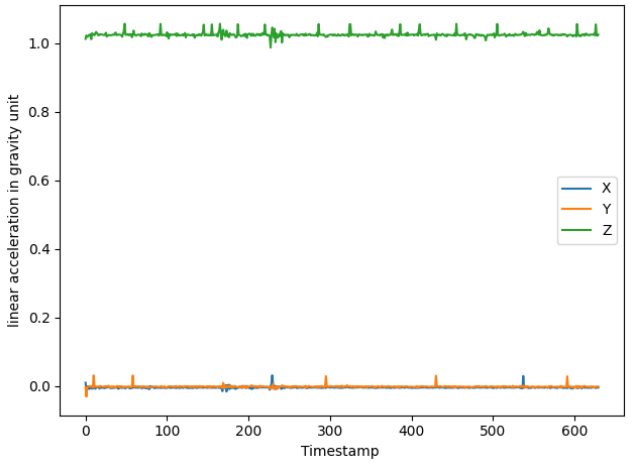


**Figure 1: Linear acceleration in gravity unit**

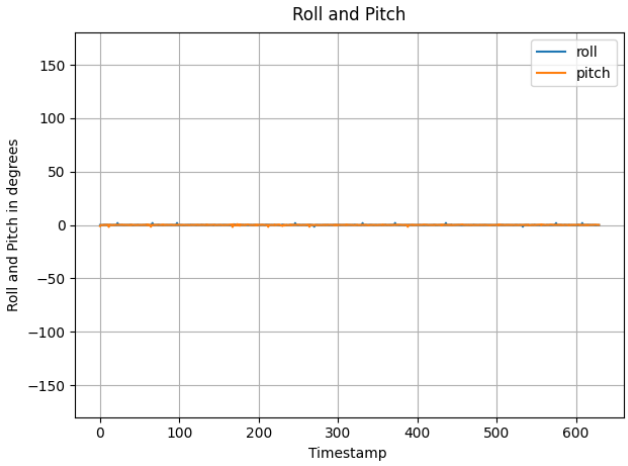


**Figure 2: Roll and Pitch angles from accelerometer, in degree**

*Task 1c.* Regarding the variance matrix of the IMU measurement noise, we assumed that all the measured variables were independent with each other. Therefore, We computed the variance matrix of the IMU using the data and *np.cov* function:

```
R = np.diag(np.diag(np.cov(imu[:, 1:12].T)))
```

The diagonal elements in the variance matrix are:
$[6.905e-06, 1.132e-05, 4.158e-05, 6.827e-02, 5.873e-02, 3.742e-02, 1.299e-01, 3.740e-02, 2.125e-05, 1.060e-05, 1.579e-05]$.

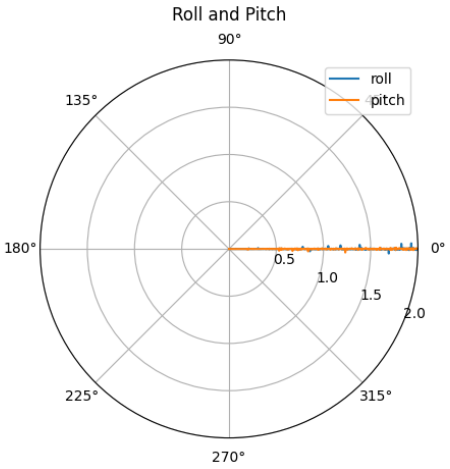


**Figure 3: Roll and Pitch angles plot in polar coordinates**

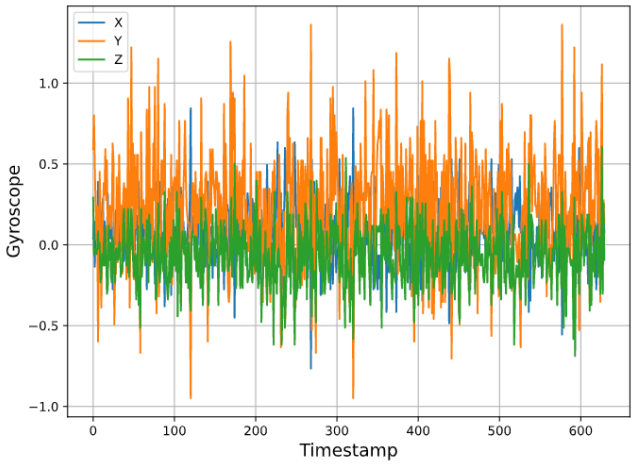


**Figure 4: Gyroscope, angular velocity x,y,z in degree/s**

However, if we visualize the whole covariance matrix, we would see that the gyro readings are actually correlated (in Figure 6).

## Task 2 -IMU calibration

*Task 2a.* On this task we have to plot the data from IMU when the car is moving. The visualization of the data for this task is presented in Figures 7, 8, 9, 10 and 11.

The plots for the task 2 are slightly different from the results in task 1, this is because on the first task we had the static case, while on the second for IMU calibration, the car was rotated and moved. If we compare the linear accelerations in the task 1 (Figure 1) and linear accelerations from the task 2 (Figure 7), we observe the moving in direction of Z,X and Y axis. The same behaviour can be observed from Figure 9, we see the change in pitch angle around 1700 timestamp, and then the same for roll angle. The gain $k_i$ and bias $bi$ was computed using the following formulas:

$$b_i = \frac{a_u + a_d}{2}$$

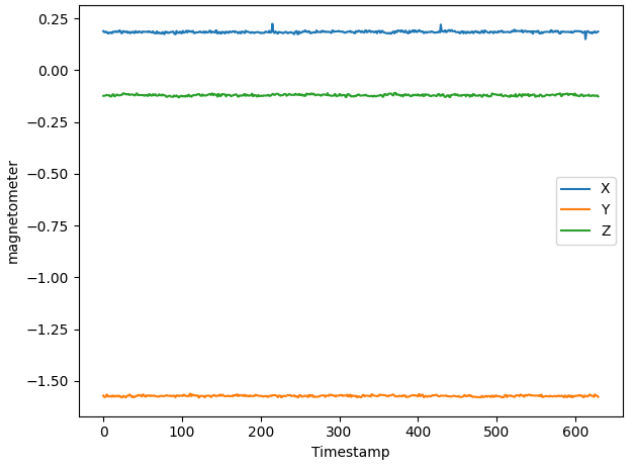


**Figure 5: Magnetometer field strength in x,y,z**

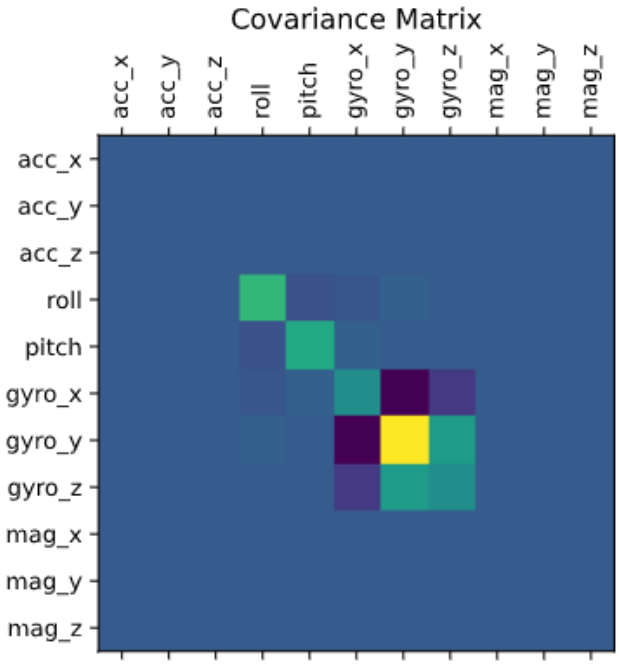


**Figure 6: Covariance matrix of the readings**

$$k_i = \frac{a_u - a_d}{2}$$

Note that readings of the accerometer are given in unit of gravitational acceleration, and our gain and bias are calculated based on the same unit. The actual value of acceleration should be the reading times the gravitational acceleration $g$.

With the plotted trend in Figure 6, we first select the values that are larger than 0.8 (about 80% of the steady value), and then discard the heading and trailing 5% to leave out those unstable readings. The actual codes for this part are

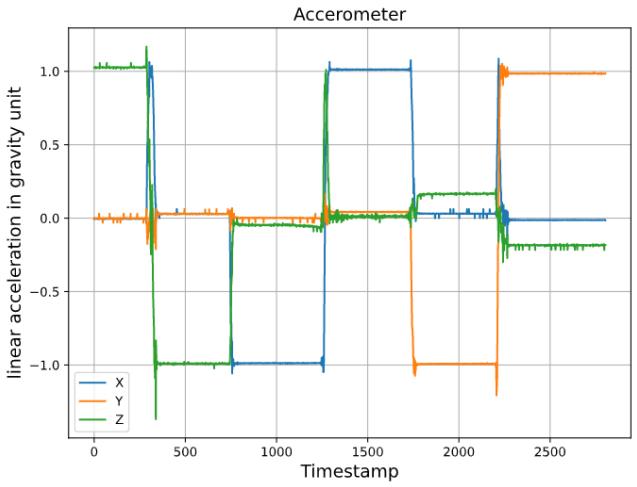


**Figure 7: Linear acceleration in gravity unit, car moving case**

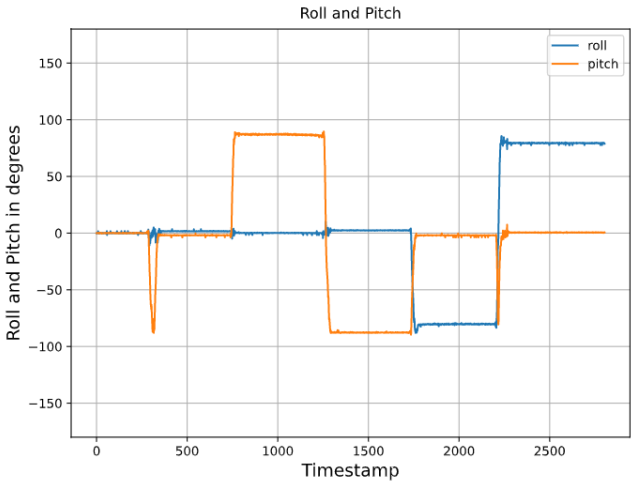


**Figure 9: Roll and Pitch angles from accelerometer, in degree, car moving case**

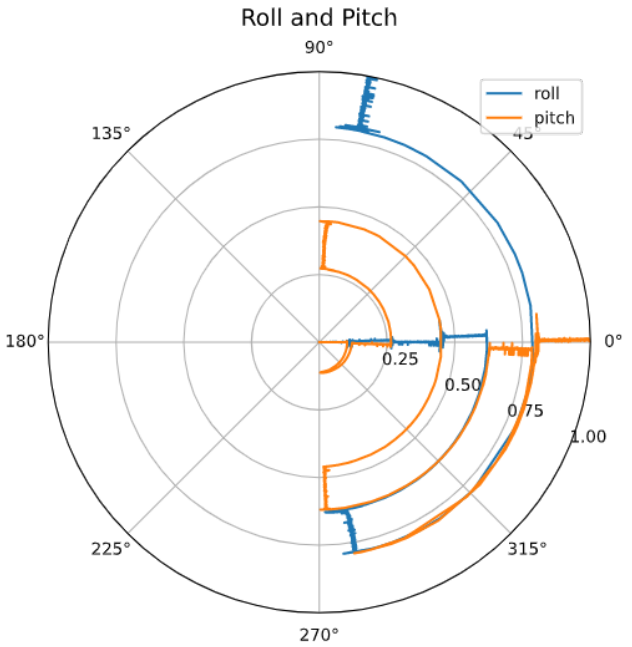


**Figure 8: Roll and Pitch angles plot in polar coordinates, car moving case**

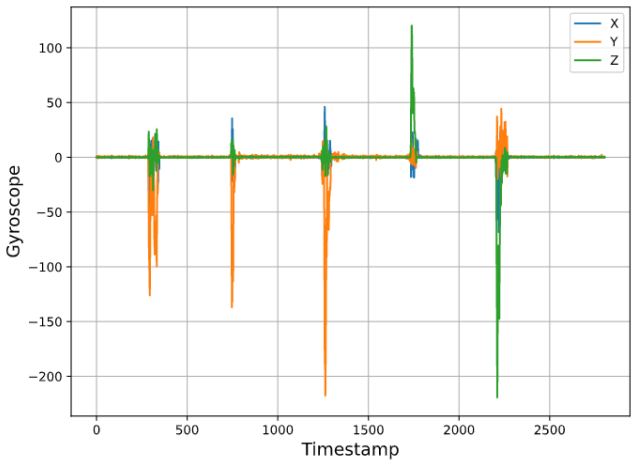


**Figure 10: Gyroscope, angular velocity x,y,z in degree/s, car moving case**

```
def truncate(signal, prop = 0.05):
    trunc_len = round(prop*signal.size)
    return signal[trunc_len:-trunc_len]

def get_gain_bias(signal):
    # ki * [-1, 1] + bi = [au, ad]
    # => ki = (au - ad)/2, bi = (au + ad)/2
    up_signal = signal[signal >  0.8]
    dn_signal = signal[signal < -0.8]
    au = np.mean(truncate(up_signal))
    ad = np.mean(truncate(dn_signal))
    ki = (au - ad)/2
    bi = (au + ad)/2
    return ki, bi
```

The calculated gain for x, y, z axis are $[0.999, 0.989, 1.009]$, and the corresponding biases are $[0.011, -0.004, 0.018]$. Considering the x and y measurements are actually the negative of the true value, the gain values does not change, but the bias of x and y should take an opposite sign so the corrected biases are $[-0.011, 0.004, 0.018]$.

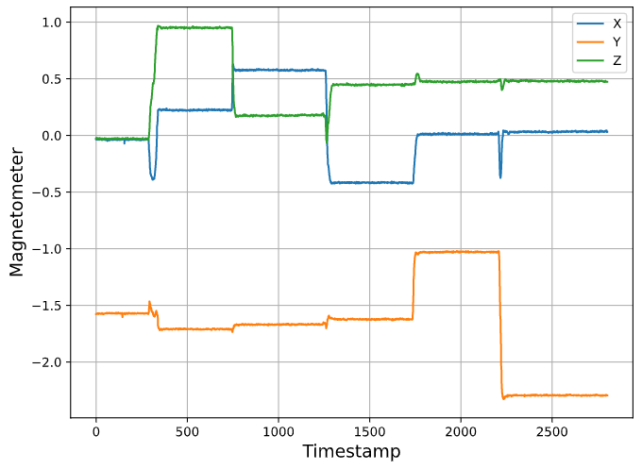


**Figure 11: Magnetometer field strength in x,y,z, car moving case**

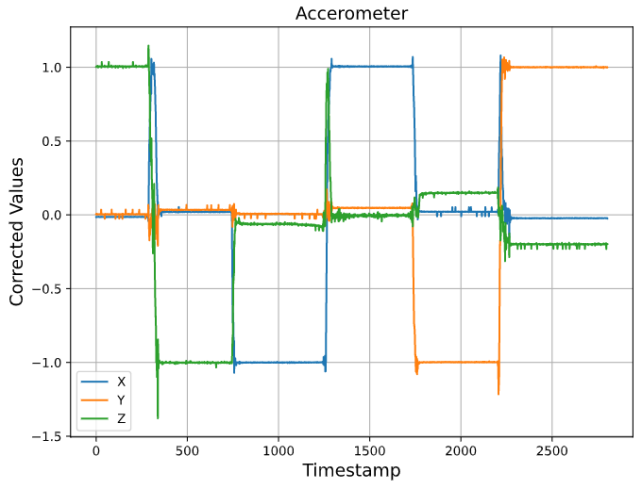


**Figure 12: Corrected accerometer readings**

*Task 2b.* The acceleration measurement is the scaled and biased result of the true value, in our case we use:

$$\begin{bmatrix} y1 \\ y2 \\ y3 \end{bmatrix} = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{bmatrix} * \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} + \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

For simplification, we define $G = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{bmatrix}$ and the formula above could be expressed with an affine model $y = Gx + b$. This model could be transformed to a linear model $\tilde{y} = y - b = Gx$, with which we could use least mean square method to estimate the true value of $x$.

Define cost $J = \sum_i (\tilde{y}_i - Gx)^2 = (\tilde{y} - Gx)^T * ((\tilde{y} - Gx))$

$$\frac{\partial J}{\partial x} = \tilde{y}^T\tilde{y} - 2x^T G^T\tilde{y} + x^T G^T Gx = 0$$

$$x_{LS} = (G^T G)^{-1} G^T \tilde{y}.$$

In our case, the matrix $G$ is a diagonal matrix, and therefore the final expression could be simplified into three independent formulas:

$$a_x = (y_1 - b_1)/k_1$$
$$a_y = (y_2 - b_2)/k_2$$
$$a_z = (y_3 - b_3)/k_3$$

With these formulas, we can now compute the corrected readings of the accerometer (shown in Figure 12, still in gravitational unit).

## Task 3-Camera module calibration

*Task 3a.* On this task we read the measured distance (in cm) and the height (in pixels) from the given .csv file. In order to get the true distance between the QR code and the camera we had to add the wooden bar distance(5cm) and the IR detector(1.7cm) measured distance.

```
distance_cm = np.array(Cam.iloc[:,0:1])
.squeeze(-1)
```

```
distance_cm = distance_cm+self.d+self.wooden_bar
```

To plot one over the height against the recorded distance, we computed the gradient and bias, by solving the following problem.

$$distance = \frac{1}{height} * x$$

We computed matrix $A = \begin{bmatrix} \frac{1}{height} & 1 \\ . & 1 \\ . & 1 \\ . & 1 \end{bmatrix}$ We define $distance = A * x$, where $x = [gradient, bias]$ Solving for x, we have:

```
A_matrix = np.vstack([1 / height_px,
        np.ones(len(height_px))]).T

gradient, bias =
np.linalg.lstsq(A_matrix, distance_cm)[0]
```

The gradient is 6213.07, and bias is 3.25
The result is presented in Figure 13

*Task 3b.* We have the gradient equal to multiplication if the QR code and the actual focal length in pixels. Therefore, $focal\_length = gradient/h_0 = 540.26$

## Task 4 - Motor control

On this task, we have to estimate the robot speed. In the provided .csv file we have 2 columns (position and time). The car was moving from position 0 to 280 cm, and at each 40 cm the required amount of time to reach that point was recorded. Let's note the position $p$, and time $t$, at each point we have $\Delta p$ and $\Delta t$. We know that velocity is the derivative of the position, $v = \dot{p} = \frac{d}{dt}p = \frac{\Delta p}{\Delta t}$. With the travel time for each 40 cm distance, we could calculate the robot's average speed in every time interval. For simplicity, we assume the robot reaches its mean speed at the mid point of the time interval. Therefore the position and estimated velocity of the car is presented

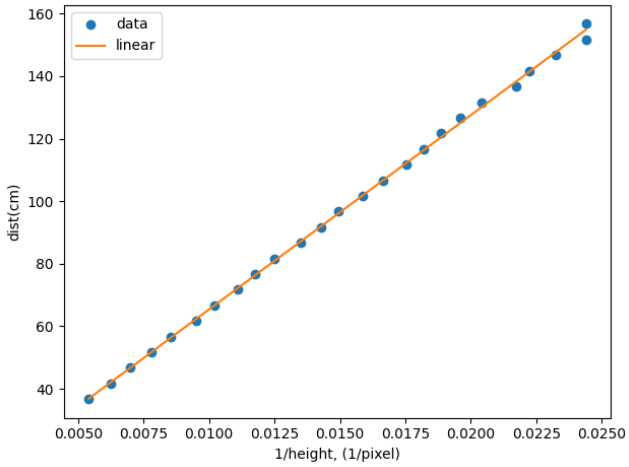


**Figure 13: Task3**

in Figure 14. From the plot, we see that the robot accelerates in the first few seconds, and then operates at a speed around 5m/s.

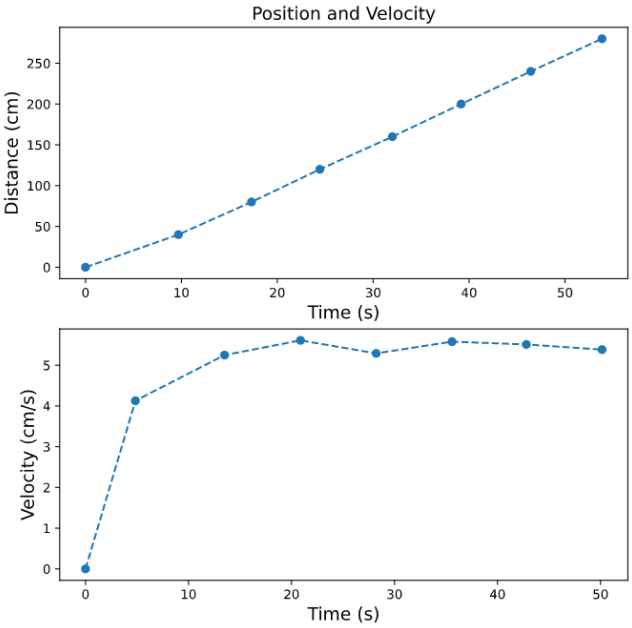


**Figure 14: Position and Velocity of the car**

## 2 PART - LOCALIZATION

### Task 5a

For this task, we try to localize the robot with its observation on the QR codes. We know the global location $\left(s_i^x, s_i^y\right)$ and actual height $h_0$ of each QR code, and we've estimated the focal length $f$ during camera calibration. With these, we could estimate the robot's position $(p_i^x, p_i^y)$ and its heading angle $\psi$ based on the geometric relationship in Figure 15. From the triangles, we have

$$|\phi_i| = \psi - arctan\left(\frac{(s_i^y - p_i^y)}{(s_i^x - p_i^x)}\right).$$

Since the $\phi_i$ in the figure is a clock-wise angle, it should be negative. Therefore the observation angle of the $i$th QR code is:

$$\phi_i = arctan\left(\frac{(s_i^y - p_i^y)}{(s_i^x - p_i^x)}\right) - \psi.$$

Meanwhile, the distance from the camera to center of QR code is

$$d_i = \sqrt{(s_i^x - p_i^x)^2 + (s_i^y - p_i^y)^2}.$$

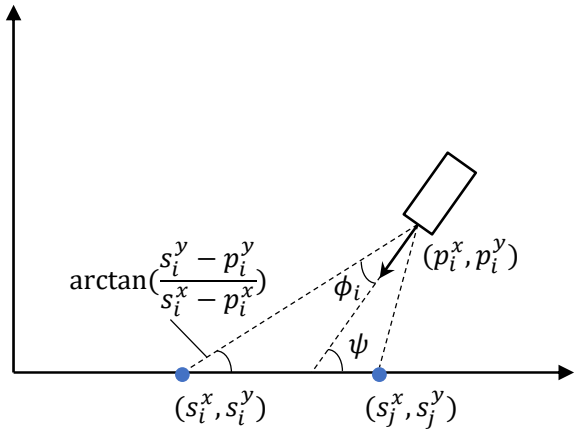


**Figure 15: Geometric relationship**

Based on the pin hole camera model in the project guide, if we know the pixel height $h_i$ and pixel position $(C_{x,i}, C_{y,i})$ of a QR code, we know

$$d_i = \frac{h_0 f}{h_i}$$

$$\phi_i = \arctan\left(\frac{C_{x,i}}{f}\right)$$

With those equations, we can get a function $\mathbf{y} = \mathbf{g}(x)$ that connects the robot's position and heading with the QR code's height and center location:

$$\begin{bmatrix} h_i \\ C_{x,i} \end{bmatrix} = \begin{bmatrix} \frac{h_0 f}{\sqrt{(s_i^x - p_i^x)^2 + (s_i^y - p_i^y)^2}} \\ f \tan\left(\arctan\left(\left(s_i^y - p_i^y\right) / \left(s_i^x - p_i^x\right)\right) - \psi\right) \end{bmatrix}$$

To estimate the robot's position and heading, we need at least two QR codes, which could be understood both in geometry and algebra. With one QR code, we could know its' distance to the robot $d_i$ and the observation angle $\Phi_i$ Geometrically, we could draw a half circle with radius $d_i$ that centers on this QR code. The robot must lie on this half circle. We could draw another half circle with another QR code, and the robot's position is the intersection of those two half circles. With the position, we could use either of the two observation angle to determine the robot's heading. From an

algebraic perspective, we need to solve for 3 unknowns and each QR code gives us 2 independent equations, so we need at least 2 QR codes.

## Task 5b

In this task we need to estimate the position and orientation of the robot using the nonlinear (weighted) least squares. We defined the measurement model in following way:

$$d_i = \sqrt{(s_i^x - p_i^x)^2 + (s_i^y - p_i^y)^2}$$

$$\phi_i = arctan(\frac{(s_i^y - p_i^y)}{(s_i^x - p_i^x)}) - \psi$$

where $[p^x, p^y, \psi]$ is the position and orientation of the robot, $d_i$ and $\phi_i$ is the distance and angle to the detected qr code.

$$\underbrace{\begin{bmatrix} d_i \\ \phi_i \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} \sqrt{(s_i^x - p_i^x)^2 + (s_i^y - p_i^y)^2} \\ arctan(\frac{(s_i^y - p_i^y)}{(s_i^x - p_i^x)}) - \psi \end{bmatrix}}_{g(x)}$$

Using the measurement model $y = g(x)$, we can compute the least squares cost in following way:

$$J_{LS}(x) = \sum_{n=1}^{N} (y_n - g_n(x))^2$$

The gradient is computed as:

$$\frac{\partial J_{LS}(x)}{\partial x} = \frac{\partial}{\partial x} \sum_{n=1}^{N} (y_n - g_n(x))^2 = \sum_{n=1}^{N} -2 * (y_n - g_n(x)) * \underbrace{\frac{\partial g_n(x)}{\partial x}}_{Jacobian\_of\_g_n(x)}$$

We note the Jacobian as:

$$G_x = \frac{\partial g_n(x)}{\partial x}$$

In this case the generalization to weighted least squares will be:

$$\frac{\partial J_{WLS}(x)}{\partial x} = \frac{\partial}{\partial x}(y - g(x))^T R^{-1}(y - g(x)) = -2G_x^T R^{-1}(y - g(x))$$

The gradient $G_x$ is computed in following way:

$$G_x = \begin{bmatrix} \frac{\partial g_1(x)}{\partial x1} & \dots & \frac{\partial g_N(x)}{\partial x1} \\ \dots & \dots & \dots \\ \frac{\partial g_1(x)}{\partial xk} & \dots & \frac{\partial g_N(x)}{\partial xk} \end{bmatrix}$$

In our case Jacobian is:

$$G_x = \begin{bmatrix} \frac{\partial d_i(x)}{\partial p_i^x} & \frac{\partial \phi_i(x)}{\partial p_i^x} \\ \frac{\partial d_i(x)}{\partial p_i^y} & \frac{\partial \phi_i(x)}{\partial p_i^y} \\ \frac{\partial d_i(x)}{\partial \psi} & \frac{\partial \phi_i(x)}{\partial \psi} \end{bmatrix}$$

To compute $\frac{\partial g_2(x)}{\partial p^x}$, we used:

$$\frac{d}{dx} arctan(\frac{y}{x}) = \frac{1}{1 + (\frac{y}{x})^2} * (\frac{d}{dx} * \frac{y}{x})$$

$$\frac{d}{dx} arctan(\frac{y}{x}) = \frac{1}{1 + (\frac{y}{x})^2} * \frac{y}{x^2} \frac{dy}{dx}$$

$$\frac{d}{dx} arctan(\frac{y}{x}) = -\frac{y}{x^2 + y^2} \frac{dy}{dx}$$

To compute $\frac{\partial g_2(x)}{\partial p^y}$, we differentiate using the chain rule, which states that: $\frac{d}{dy}[f(g(y))]$ is $f'(g(y))g'(y)$ where $f(y) = arctan(y)$ and $g(y) = \frac{y}{x}$. We set the $u = \frac{y}{x}$ and then:

$$\frac{d}{du}[arctan(u)] \frac{d}{dy}[\frac{y}{x}]$$

The derivative of $arctan(u)$ with respect to $u$ is $\frac{1}{1+u^2}$, result that

$$\frac{d}{du}[arctan(u)] \frac{d}{dy}[\frac{y}{x}] = \frac{1}{1 + u^2} \frac{d}{dy}[\frac{y}{x}]$$

By replacing $u$ with $\frac{y}{x}$ we have $\frac{1}{1+(\frac{y}{x})^2} \frac{d}{dy}[\frac{y}{x}]$. Therefore,

$$\frac{d}{dy} \arctan(\frac{y}{x}) = \frac{1}{x * (1 + (\frac{y}{x})^2} = \frac{1}{x + \frac{y^2}{x}}$$

The Jacobian $G_x$ is computed as:

$$G_x = \begin{bmatrix} \frac{\partial d_i(x)}{\partial p_i^x} & \frac{\partial \phi_i(x)}{\partial p_i^x} \\ \frac{\partial d_i(x)}{\partial p_i^y} & \frac{\partial \phi_i(x)}{\partial p_i^y} \\ \frac{\partial d_i(x)}{\partial \psi} & \frac{\partial \phi_i(x)}{\partial \psi} \end{bmatrix} = \begin{bmatrix} \frac{-(s_i^x - p_i^x)}{\sqrt{(s_i^x - p_i^x)^2 + (s_i^y - p_i^y)^2}} & -\frac{(s_i^y - p_i^y)}{(s_i^x - p_i^x)^2 + (s_i^y - p_i^y)^2} \\ \frac{-(s_i^y - p_i^y)}{\sqrt{(s_i^x - p_i^x)^2 + (s_i^y - p_i^y)^2}} & \frac{(s_i^x - p_i^x)}{(s_i^x - p_i^x)^2 + (s_i^y - p_i^y)^2} \\ 0 & -1 \end{bmatrix}$$

We used Gauss-Newton method presented in exercise sessions for lsq with line search for $\gamma$ estimation. The rezult of localization is prezented in Figure 16.

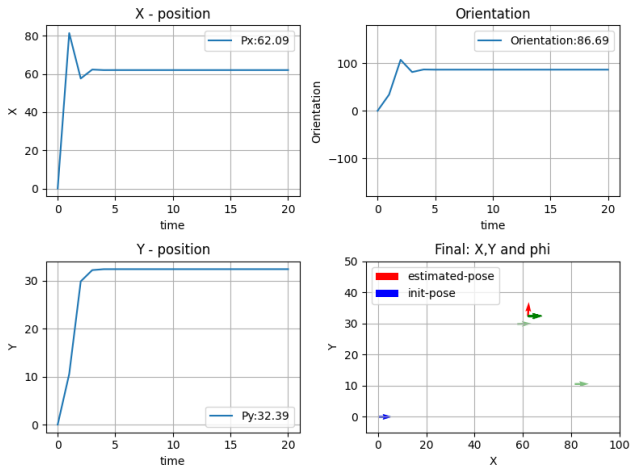


**Figure 16: Localization**

We run the lsq algorithm for 20 iterations, and we notice from Figure 16 that it converges in first 5 iterations. We started with the initial guess: $init\_x = [0, 0, 0]$, the true pose of the robot is: $x\_true = [61.0, 33.9, 90.0]$, and the algorithm returned: $estimated\_pose = [62.08, 32.39, 86.68]$. In Figure 16 bottom right, we plotted with blue the initial pose of the robot, with green are the intermediary results and with red the final estimated pose.

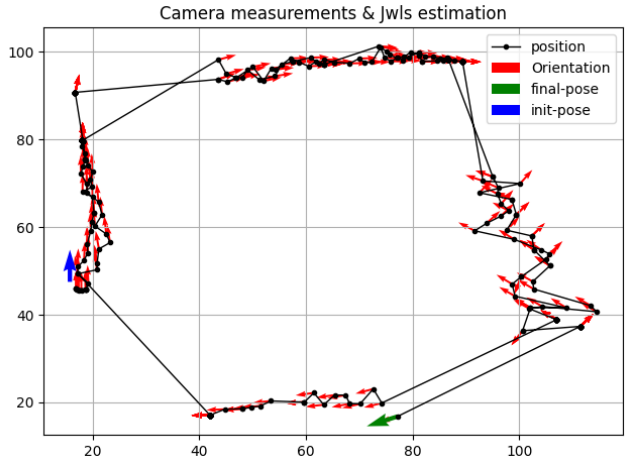


**Figure 17: Tracking with camera measurements**

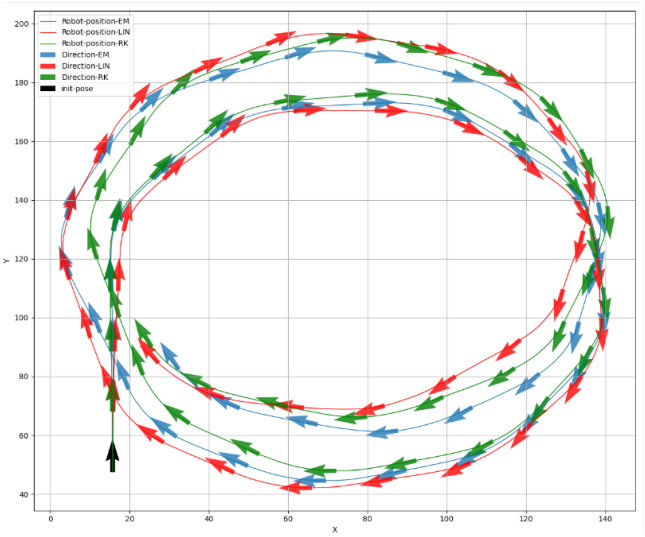


**Figure 18: Dead-reckoning**

## 3 PART - TRACKING

To track the robot, for the beginning we started with the same method described above in the Location section. We used weighted least squares to estimate the position and orientation of the robot, but in this case we computed the estimation each time-stamp. The result of tracking using only camera measurements without any filtering algorithm is presented in Figure 17. We plotted with black dots the position of the car and with red arrows the orientation. We got 4 clusters of points which correspond to each wall in the real world.
We notice that the estimation is very noisy and we need to proceed with further methods to accurately track the car.

### Task 6a

For this part, we chose the Quasi-Constant Turn model. Since the velocity could be obtained directly from the motor, we could further simplify the model into the following case:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\phi}\end{bmatrix} = \begin{bmatrix}v * cos(x)\\ v * sin(x)\\ w_{gyro} + w\end{bmatrix}$$

The input $u$ consists of car linear velocity and the turning rate, which comes from gyroscope acceleration on the Z axis.

$$u = [u, w_{gyro}]$$

Our motion model function is:

$$\dot{x} = f(x, u) + B(x) * w$$

where $f = \begin{bmatrix}v * cos(x)\\ v * sin(x)\\ w_{gyro}\end{bmatrix}$, $B = I_3 = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$ and $w$ is the input noise. We approximate the $\dot{x} = f(x_{t-1}) + A_x(x_t - x_{t-1}) + B * u$ , where $A$ is the Jacobian matrix of the dynamic function defined above.

$$A = \begin{bmatrix}0 & 0 & -v * sin(\phi)\\ 0 & 0 & v * cos(\phi)\\ 0 & 0 & 0\end{bmatrix}$$

We discretized the $x$ using:

$$F = I + \frac{1}{2} * A * dt + A^2 * dt$$

and

$$x_{t+1} = x_t + F * f(x_t, u_t) + F * w_t$$

Besides the presented method, we also implemented the Euler-Maruyama discretization method.

$$x_t = x_{t-1} + \int_{t-1}^{t} f(x_t)dt + \int_{t-1}^{t} B_w * w_t * dt$$

### Task 6b

On this task we did dead-reckoning method to track the car. We used car velocity computed in the first part of the project, and IMU gyroscope measurement on Z axis. We calibrated the gyroscope data, by subtracting the gyroscope bias computed before. Besides the linearization and E-M methods presented above, we also used runge kutta 4th order method to estimate the car position. The dead-reckoning result is presented in Figure 18. We started from the known ground truth position: $[15.7, 47.5, np.deg2rad(90)]$. The car made 2 circles, but, as we see in Figure 18, the farther we are from the starting point, the more uncertain is the current position.

### Task 6c

Since we don't have the ground truth position of the car, we decided to take the average position of the presented methods (E-M, Linearization and RK4), result is presented in Figure 19.

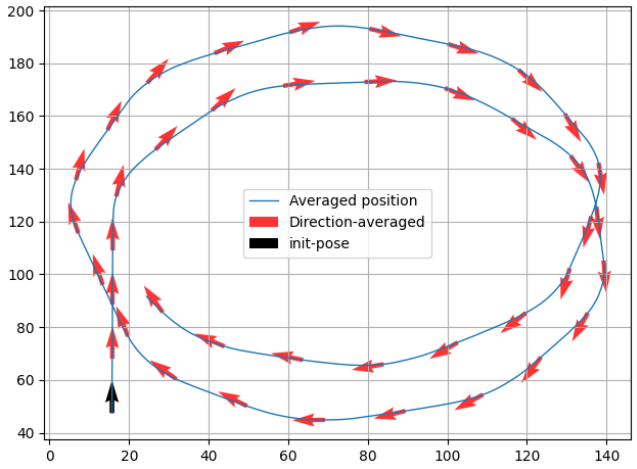


**Figure 19: Dead-reckoning final averaged estimation**

## Task 7

On this task we had to solve the tracking problems using both, IMU and Camera measurements. The first challenge was to align the sensors time-stamps, also at each Camera time-stamp, there are more than just one detected qr code. Our solution to get the multiple detected qr codes from camera measurements was to group camera data by timestamp column, and then for each timestamp get the qr index and later get its global positions as landmarks. To align the measurements with the IMU and motor, for each camera time index, we got the closest record from IMU and motor files, using *idxmin*() function from *pandas*.
We decided to use EKF as a first filter approach for tracking problem. The robot has 3 states, $x, y$ position and orientation $\phi$:

$$x = \begin{bmatrix} x_t \\ y_t \\ \phi_t \end{bmatrix}$$

The inputs are the car linear velocity $v$ and turning rate $w_{gyro}$.

$$u_t = \begin{bmatrix} v_t \\ w_t \end{bmatrix}$$

The robot is using camera sensor to detect the qr codes, whose global positions are known. The measurement model is defined as the distance and angle to the detected qr code.

$$z_t = \begin{bmatrix} d_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} \sqrt{(s_x - x_t)^2 + (s_y - y_t)^2} \\ atan2(\frac{s_y - y_t}{s_x - x_t}) - \phi_t \end{bmatrix}$$

We defined the motion model as:

$$x_{t+1} = F * x_t + B * u_t$$

where:

$$F = I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} cos(\phi) * DT & 0 \\ sin(\phi) * DT & 0 \\ 0 & 1 * DT \end{bmatrix}$$

We linearizeed the motion model by computing the Jacobian matrix $J$, where $J$ is:

$$J = \begin{bmatrix} \frac{dx}{dx} & \frac{dx}{dy} & \frac{dx}{d\phi} \\ \frac{dy}{dx} & \frac{dy}{dy} & \frac{dy}{d\phi} \\ \frac{d\phi}{dx} & \frac{d\phi}{dy} & \frac{d\phi}{d\phi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -DT * sin(\phi) \\ 0 & 1 & DT * cos(\phi) \\ 0 & 0 & 1 * DT \end{bmatrix}$$

The jacobian of the measurement model is:

$$G = \begin{bmatrix} \frac{-(s_x - x)}{\sqrt{(s_x - x_t)^2 + (s_y - y_t)^2}} & \frac{-(s_y - y)}{\sqrt{(s_x - x_t)^2 + (s_y - y_t)^2}} & 0 \\ 0 & 1 & 0 \\ \frac{s_y - y}{\sqrt{(s_x - x_t)^2 + (s_y - y_t)^2}} & \frac{-(s_x - x)}{\sqrt{(s_x - x_t)^2 + (s_y - y_t)^2}} & -1 * DT \end{bmatrix}$$

The EKF algorithm presented below, was adapted from [1].

**Algorithm 6.2** Extended Kalman Filter

1: Initialize $\hat{\mathbf{x}}_{0|0} = \mathbf{m}_0$, $\mathbf{P}_{0|0} = \mathbf{P}_0$
2: **for** $n = 1, 2, \ldots$ **do**
3: Prediction (time update):

$$\hat{\mathbf{x}}_{n|n-1} = \mathbf{f}(\hat{\mathbf{x}}_{n-1|n-1})$$
$$\mathbf{P}_{n|n-1} = \mathbf{F}_x(\hat{\mathbf{x}}_{n-1|n-1})\mathbf{P}_{n-1|n-1}\mathbf{F}_x^\top(\hat{\mathbf{x}}_{n-1|n-1}) + \mathbf{Q}_n$$

4: Measurement update:

$$\mathbf{K}_n = \mathbf{P}_{n|n-1}\mathbf{G}_x^\top(\hat{\mathbf{x}}_{n|n-1})(\mathbf{G}_x(\hat{\mathbf{x}}_{n|n-1})\mathbf{P}_{n|n-1}\mathbf{G}_x^\top(\hat{\mathbf{x}}_{n|n-1}) + \mathbf{R}_n)^{-1}$$
$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + \mathbf{K}_n(\mathbf{y}_n - \mathbf{g}(\hat{\mathbf{x}}_{n|n-1}))$$
$$\mathbf{P}_{n|n} = \mathbf{P}_{n|n-1} - \mathbf{K}_n(\mathbf{G}_x(\hat{\mathbf{x}}_{n|n-1})\mathbf{P}_{n|n-1}\mathbf{G}_x^\top(\hat{\mathbf{x}}_{n|n-1}) + \mathbf{R}_n)\mathbf{K}_n^\top$$

5: **end for**

We used the prediction covariance noise as:

$$Q = \begin{bmatrix} 0.1^2 & 0 & 0 \\ 0 & 0.1^2 & 0 \\ 0 & 0 & 5^2 \end{bmatrix}$$

The observation noise covariance matrix R is:

$$R = \begin{bmatrix} 1^2 & 0 \\ 0 & 1^2 \end{bmatrix}$$

The values for input and measurement noise were found by trial and error.

$$input\_noise = \begin{bmatrix} 0.4^2 & 0 \\ 0 & 5^2 \end{bmatrix}, measurement\_noise = \begin{bmatrix} 0.3^2 & 0 \\ 0 & 0.3^2 \end{bmatrix}$$

The EKF result is presented in Figure 20. We plotted with blue squares the detected qr codes, and with blue arrow the initial pose of the car. According to EKF result, the car followed almost 2 circles.

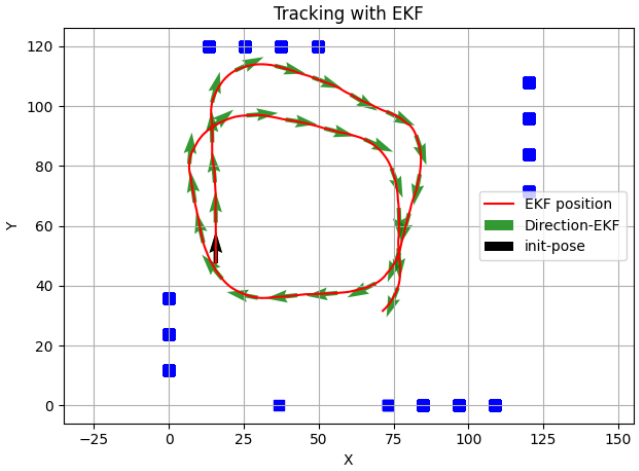


**Figure 20: EKF estimation**

The second approach for tracking was Particle Filter. We used the same motion and measurement models described in the EKF section. We created 5000 particles, uniformly on the grid, each particle is a possible car pose. The result of particle initialization is presented in Figure 21.

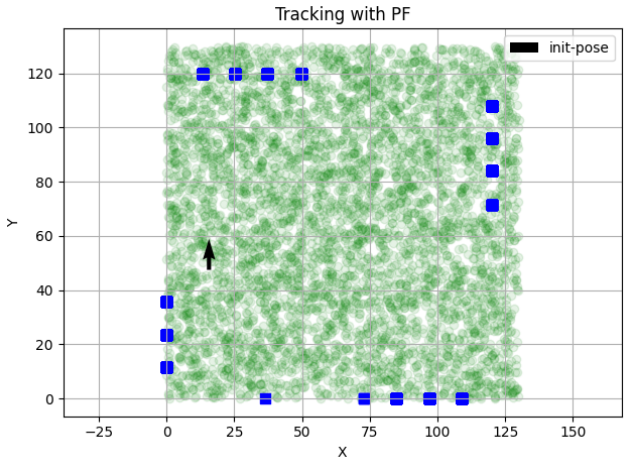


**Figure 21: Particle Filter initialization**

After several steps the particles converged to true position. The result of the final pose estimation with Particle Filter is presented in Figure 22. Having the measurement model defined as:

$$y = g(x) + r$$

in PF we compute the Gaussian likelihood: $p(y|x) = N(y; g(x), R)$, [2]
We estimate the pose by computing the mean of the particles using the weights.

$$x_{n|n} = \frac{1}{J}\sum_{j=1}^{J} w_n^j * x_n^j$$

then, the covariance matrix P is:

$$P_{n|n} = \sum_{j=1}^{J} w_n^j (x_n^j - x_{n|n})(x_n^j - x_{n|n})^T$$

**Algorithm 1** Bootstrap Particle Filter (Gaussian Noises)

1: Initialize: $\mathbf{x}_0^j \sim \mathcal{N}(\mathbf{m}_0, \mathbf{P}_0)$ $(j = 1, \ldots, J)$
2: **for** $n = 1, 2, \ldots$ **do**
3:  **for** $j = 1, 2, \ldots, J$ **do**
4:  Sample: $\mathbf{q}_n^j \sim \mathcal{N}(0, \mathbf{Q})$
5:  Propagate the state: $\mathbf{x}_n^j = \mathbf{f}(\mathbf{x}_{n-1}^j) + \mathbf{q}_n^j$
6:  Calculate the weights: $\tilde{w}_n^j = \mathcal{N}(\mathbf{y}_n; \mathbf{g}(\mathbf{x}_n^j), \mathbf{R}_n)$
7:  **end for**
8:  Normalize the importance weights $(j = 1, \ldots, J)$

$$w_n^j = \frac{\tilde{w}_n^j}{\sum_{i=1}^{J} \tilde{w}_n^i}$$

9:  Calculate the mean $\hat{\mathbf{x}}_{n|n}$ and covariance $\mathbf{P}_{n|n}$
10:  Resample such that $\Pr\{\tilde{\mathbf{x}}_n^i = \mathbf{x}_n^j\} = w_n^j$
11: **end for**

We implemented the Particle Filter algorithm presented in the lecture slides.

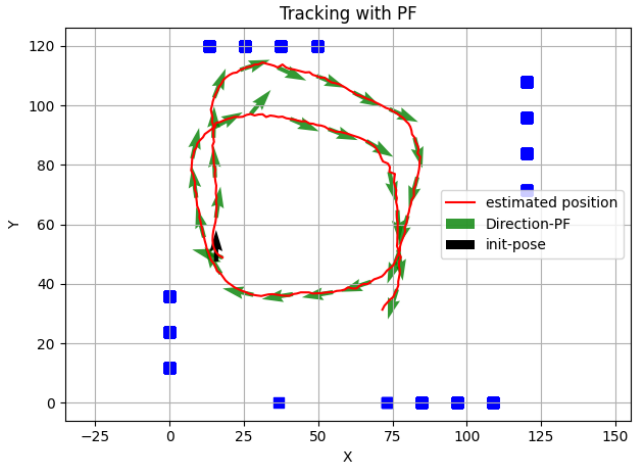


**Figure 22: Particle Filter final pose estimation**

We got similar results with PF and EKF, see Figure 20 and 22, however the orientation and position of the car is a bit more smooth with EKF, here we still need to tune the parameters to PF (change the noises, increase number of particles, experiment with different resampling methods, etc).

## CONCLUSION

In this project we performed the following tasks:

- IMU and Camera calibration
- Pose estimation using the regularized weighted least squares
- Derivation of a motion and measurement model

- Localization and tracking

We used gyroscope data for our motion model, and camera data (distance and angle to QR code) for measurements model. We used Euler-Maruyama, Linear and RK4 methods to perform the dead-reckoning.We adapted EKF method to track the car, and we are planning to compare it with Particle filter.
After all these experiments we concluded the following ideas:

- Sensors require calibration to estimate intrinsic parameters, if the calibration is not done properly, we will get many outliers and as a result a worse accuracy of the algorithms.
- Depending on the type of sensor we use, we derive our motion and measurement models.
- If there is no ground truth data provided, we can apply several algorithms to the same problem, and compare their result.

## SOURCE CODE

In this section we describe the source code solutions that you can find in the attached .py files.

- Static IMU experiment (**task1**) and IMU calibration (**task2**) solutions are presented in file *IMU_task_12.py*.
- Camera calibration (**task3**) and Motor control (**task4**) are presented in file *Camera_Motor_task_34.py*
- The notebook for task 1 to 4 is *Task1-4.ipynb*
- Static localization (**task5**) is presented in file *Localization_task56.py*, also in this file we presented the tracking (**task6**) with camera measurements.
- Dead-reckoning (**task6**) method is presented in file *DR.py*
- The file *EKF_and_PF.py* contains the solutions for tasks (**6 and 7**), tracking the car with both, EKF and PF.

## REFERENCES

[1] Roland Hostettler and Simo Särkkä. 2020. Lecture notes on Basics of Sensor Fusion. (2020).
[2] Simo Särkkä. November 27 2020. Lecture slides on Basics of Sensor Fusion. (November 27 2020).