SIS Home Assignment 5 Report

Konstantin Danilov

November 14, 2017

1 Data collection

1.1 Review

Partner name: Ruslan Rezin

Data was collected from the following sources:

1. Control data - input data for left and right motors (in tacho per second)
2. X and Y from camera, located over the terrain
3. Angle from gyroscope, located on robot
4. Angle from compass from the phone, fixed on the robot
5. Distance to the wall from the sonar

Robot is shown on 1. Green cap is intended to be recognized by the simple script to track x and y coordinates of the robot.

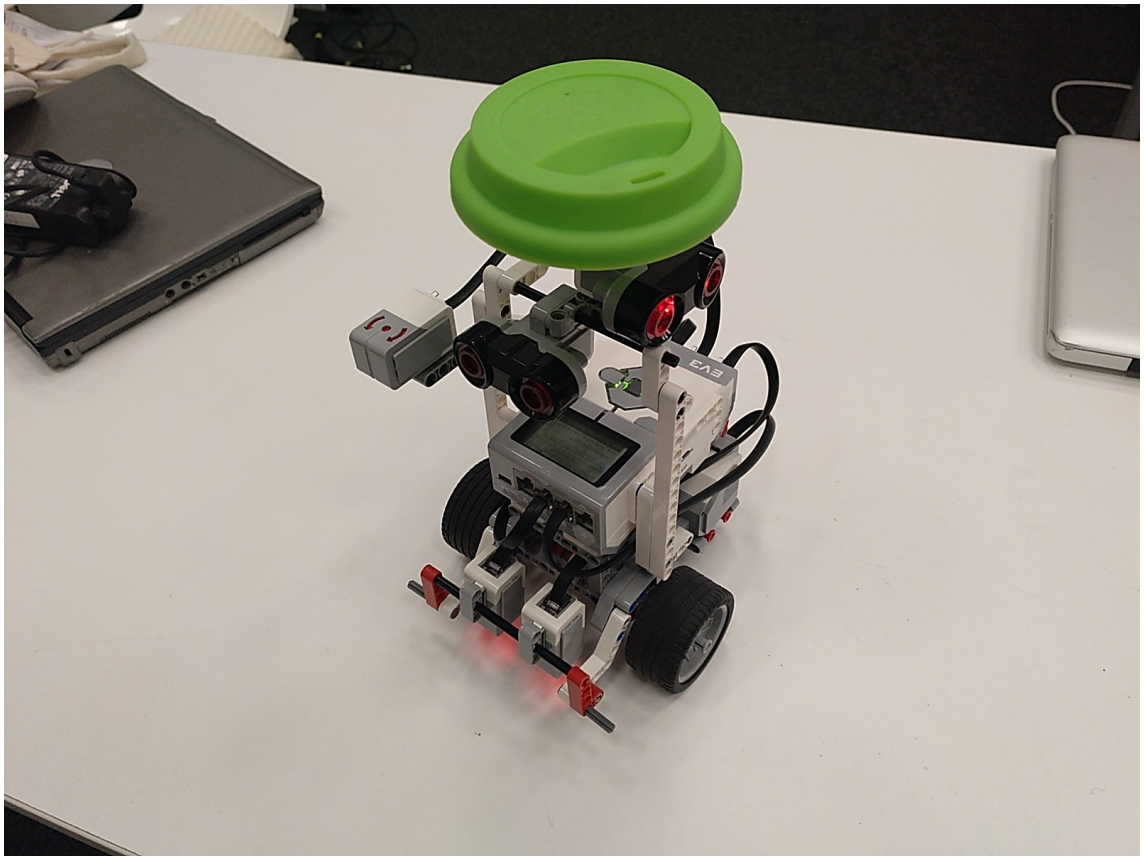

Figure 1: Robot

1.2 Trajectory program and robot's measurements

Project from the previous assignment was used to make robot following some particular trajectory - this trajectory was build using electrical tape (3). Code was extended in order to make robot recording sensors and control output data. Control output data is represented by rotation speeds of left and right motor in tacho per second. These values directly goes to the ev3dev API, controlling motors.

Code is located here.

1.3 Sonar limitations

As it can be observed on 1, there are two sonars. Initially, the idea was to use two sonars, located perpendicular and gyroscope. Such construction could provide us (X, Y, Theta) coordinates if the following requirements are satisfied:

1. There are walls, located opposite to robot's sonars.
2. Trajectory has no wide angles.

However, this solution does not work because of sonar limitations. According to the documentation, it has range up to 250 cm. In our case it provided very bad measurements (2), if distance was higher than 50 cm.

Therefore it was decided to use measurements from only one sonar, which is located on Y-axis (which matches wheels axis).

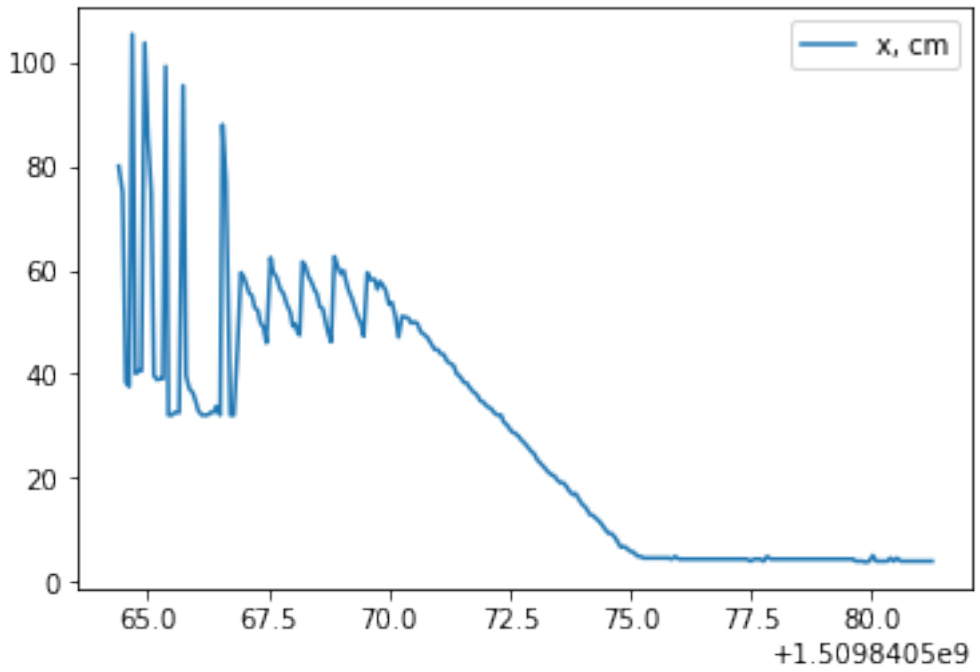

Figure 2: Bad sonar measurements example

1.4 Camera robot detection

iPhone was used to record a video with robot, following the trajectory. Then it was passed to simple script, performing color-based filtration (green pixels) and circle hough transform. This script produced log with raw data - (milliseconds from video start, circle center x in pixels, circle center y in pixels).

Next script accepts:

1. Field origin point coordinates (top left point) in px
2. Field size (width and height) in px

3. Field size (width and height) in cm

4. Movement start time (as unix timestamp, according to robot's time), recognized manually

Script produces CSV file with absolute time, x and y in centimeters.

Script is located in Separate repository.

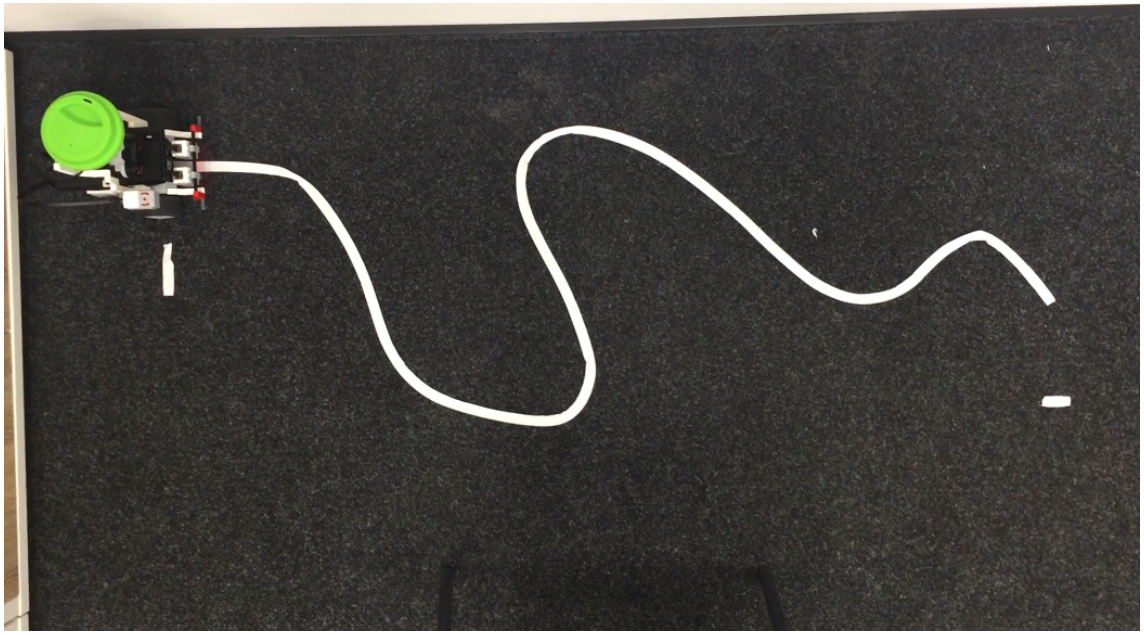

Figure 3: Robot trajectory from the camera

1.5 Compass issues

Unfortunately, data from phone's compass incorrect: it had low changes. This was related with the fact, that phone was located on EV3 control block and, probably, measured corrupted magnetic field data. That is why this data is not used.

1.6 Data merge

Since it is required to make a synchronization between data from different sources (phone, camera and robot), the following approach was developed.

At first, all data should have absolute time, which should be synchronized on all devices. Then single source is selected (with the lowest frequency) and its timeline is iterated. For each time value from "main timeline" system tries to find data with time, nearest to the given.

2 Data processing

2.1 Robot model

2.1.1 Transition function

Using classic differential drive robot kinematics equations, the following model was implemented:

$$
\begin{gathered}
state = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \; control = \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix} \\
observations = \begin{bmatrix} x_{camera} \\ y_{camera} \\ \theta_{gyroscope} \\ d_{sonar} \end{bmatrix} \\
v_t = \frac{R}{2} \times (\omega_{l,t} + \omega_{r,t}) \\
\omega_t^{'} = \frac{R}{2d} \times (\omega_{l,t} - \omega_{r,t}) \\
x_{t+1} = x_t + \Delta t \times v_{t+1} \times cos(\theta_t + \Delta t \times \omega_{t+1}^{'}) \\
y_{t+1} = y_t + \Delta t \times v_{t+1} \times sin(\theta_t + \Delta t \times \omega_{t+1}^{'}) \\
\theta_{t+1} = \theta_t + \Delta t \times \omega_{t+1}^{'}
\end{gathered}
\tag{1}
$$

Where d, R - half wheel distance and wheel radius.

Empiric model standard deviations are the following: $\sigma_x^2 = 225, \sigma_y^2 = 225, \sigma_\theta^2 = 225$. Initial deviations: $\sigma_x^2 = 25, \sigma_y^2 = 25, \sigma_\theta^2 = 4$. For the simplicity it is assumed, that all parameters are independent and covariance matrix is a diagonal matrix with stds on the main diagonal.

2.1.2 Observations function

For observations the following function was implemented:

$$
\begin{gathered}
x_{camera} = x \\
y_{camera} = y \\
\theta_{gyroscope} = \theta \\
d_{sonar} = g(y, \theta) \\
g(y, \theta) = \frac{y}{cos(\theta)} + d_{wall}
\end{gathered}
\tag{2}
$$

Where d_{wall} - Distance to the wall from the origin point. The following empirical standard deviations for observations were used: $\sigma_{x_{camera}}^2 = 100, \sigma_{y_{camera}}^2 = 100, \sigma_{\theta_{gyroscope}}^2 = 4, \sigma_{d_{sonar}}^2 = 4$

However, $\sigma_{d_{sonar}}^2$ is changed, depending on sensor's value and angle. This is done in order to protect system from incorrect measurements, when angle is too high and distance to the wall can not be measured correctly. In this case $\sigma_{d_{sonar}}^2$ is set to 100000.

For the simplicity it is assumed, that all parameters are independent and covariance matrix is a diagonal matrix with stds on the main diagonal.

2.2 Filters

2.2.1 Used libraries

All filters are implemented by myself, all of them use numpy. EKF uses numdifftools library for Jacobian calculation. Particle filter uses resample methods from filterpy and scipy to work with multivariate normal distribution.

2.2.2 Kalman filter

Since $f(x, u)$ and $h(x)$ are non-linear functions, it is not possible to use simple Kalman filter, therefore it was not implemented.

2.2.3 Extended Kalman filter

The following result was gathered from EKF. On x, y and angle plots Y axis is in centimetrs, X - is in observations number. On XY-plot all axis are in centimeters.

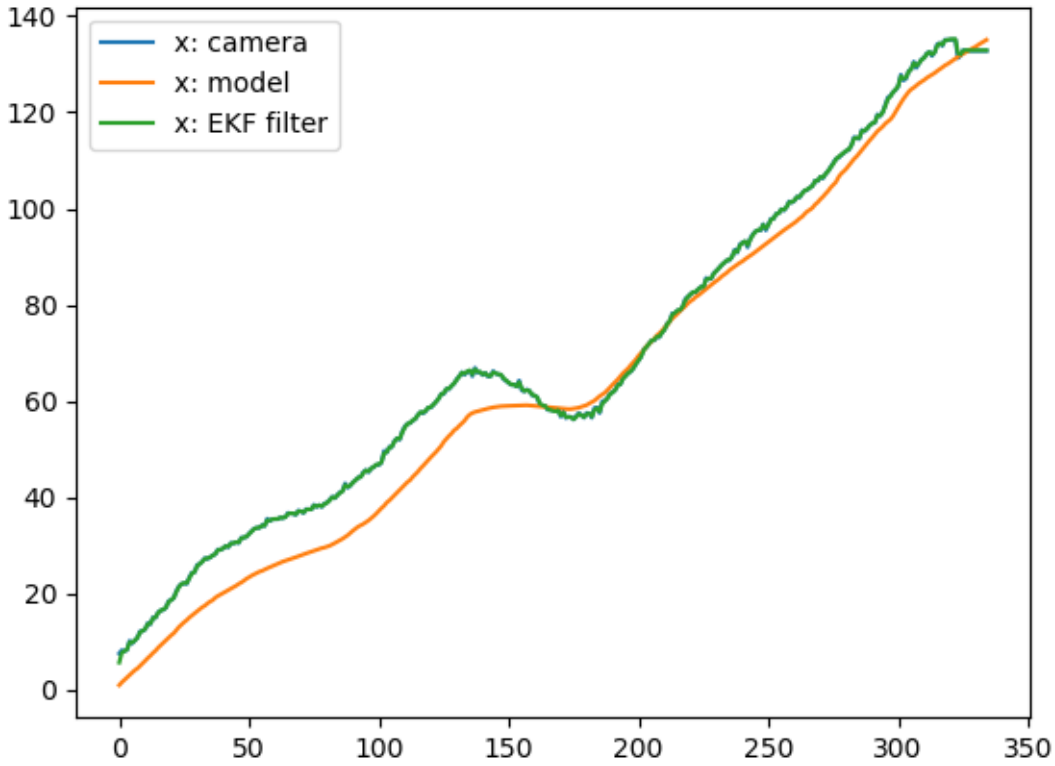

Figure 4: EKF: x values

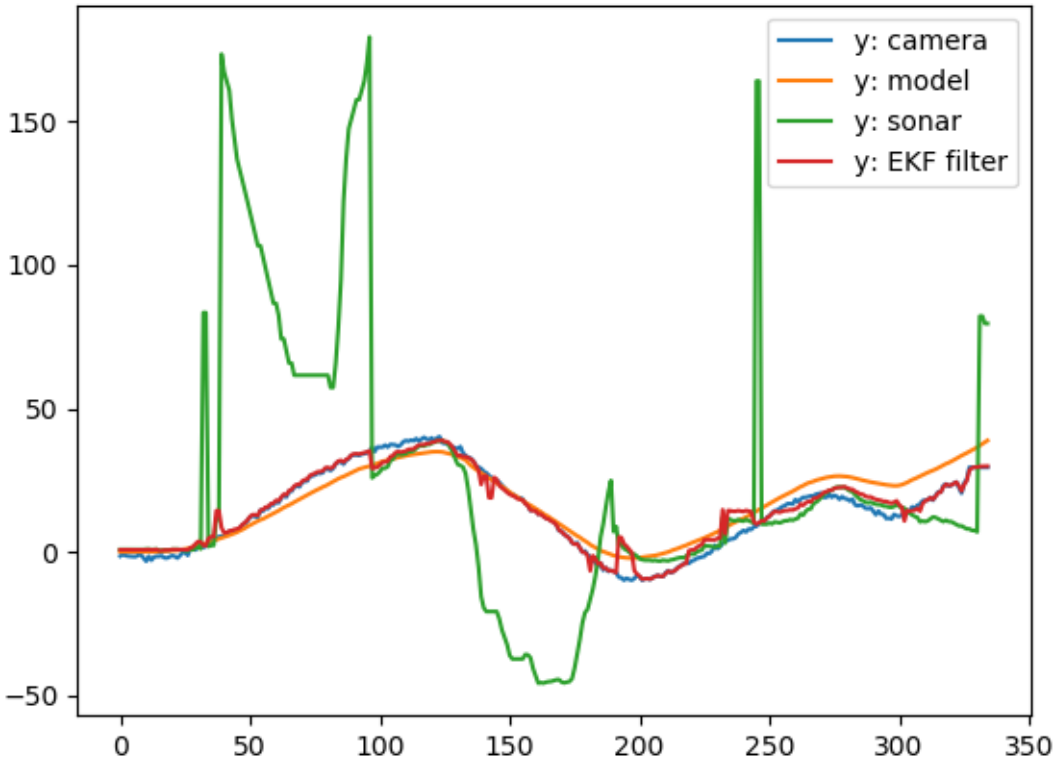

Figure 5: EKF: y values

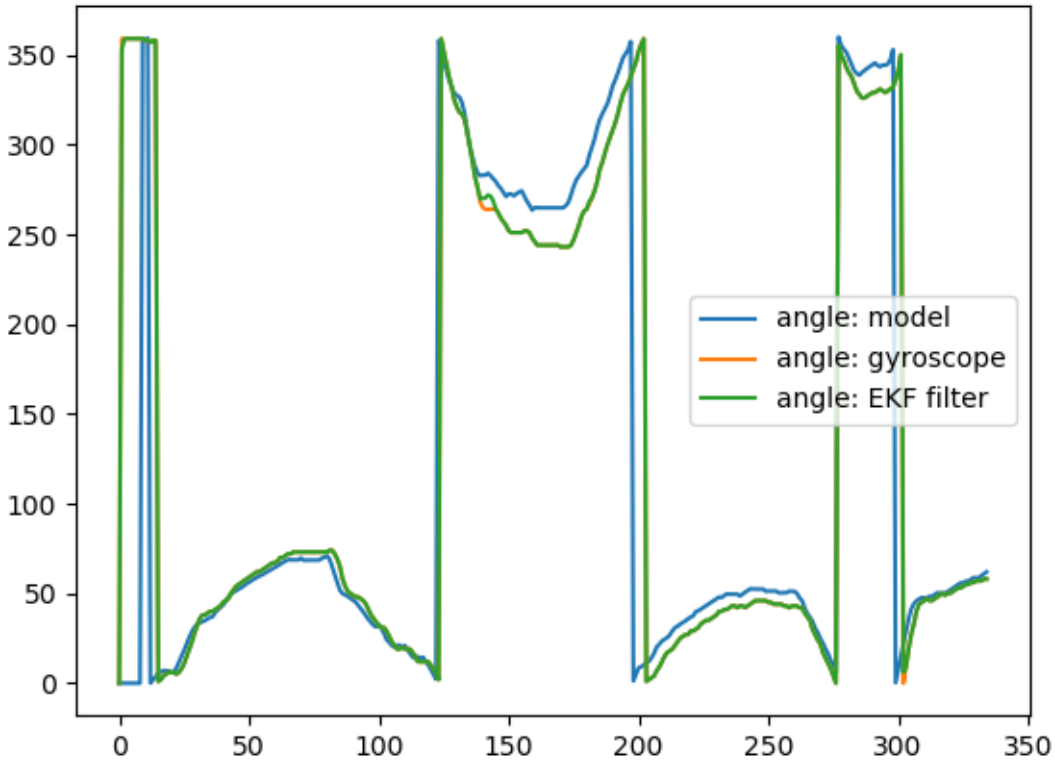

Figure 6: EKF: theta values

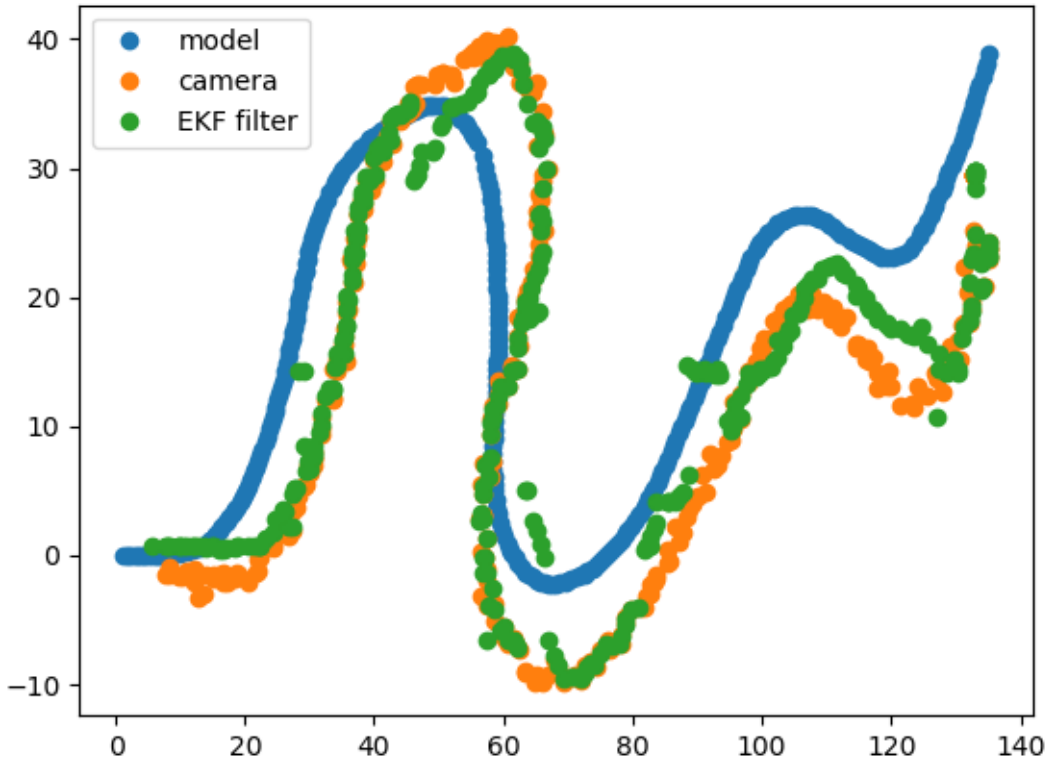

Figure 7: EKF: xy values

As it can be observed, the filter prefers values from sensors, not from model. After experiments with covariance matrices I think, that it is related with the fact that model and observations diverges from each other (but typical case for Kalman filtrations is when observations fluctuate around the model).

Outliers in the trajectory are caused by the fact that y coordinate, calculated using sonar has lower standard deviation (only when angle is small enough). I believe that these measurements are more correct than others and that is why there are taken into account. To make trajectory smoother sonar's std can be increased.

2.2.4 Unscented Kalman filter

There were a lot of issues, related to UKF implementation. They are caused by the fact that sigma-points generation algorithm uses square root of covariance matrices, therefore we have to maintain positive-defined property of the matrix. For example, this property can be broken by computational error. Some hacks were applied to deal with this problem. The following result was gathered from UKF.

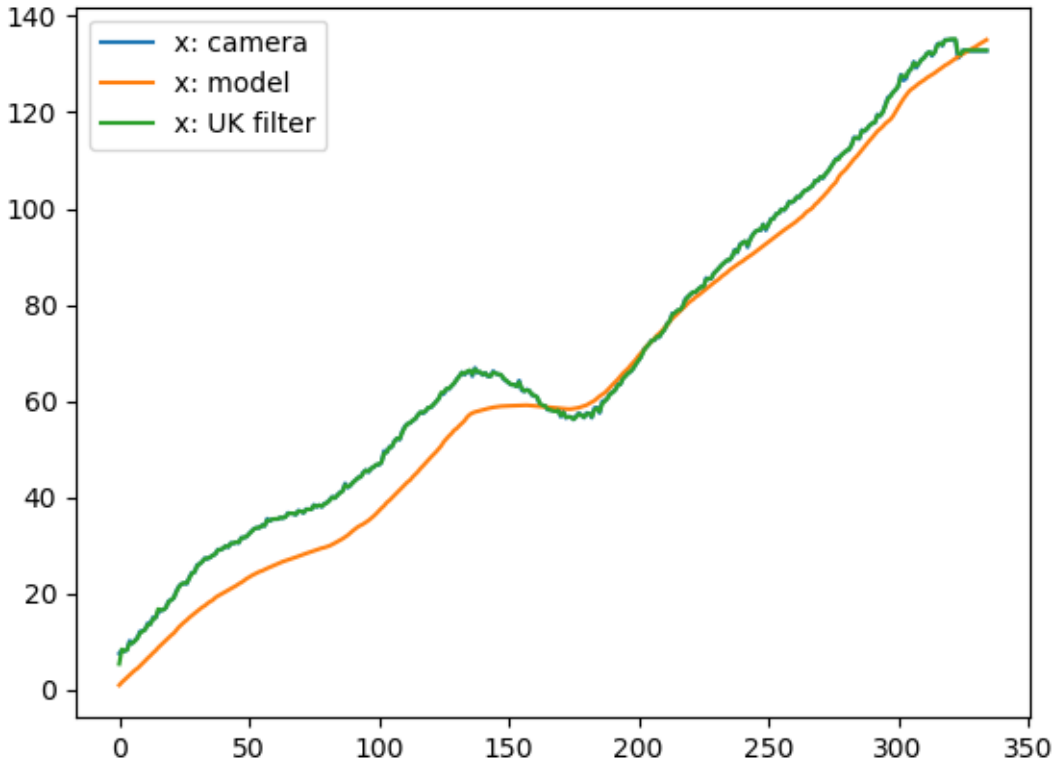

Figure 8: UKF: x values

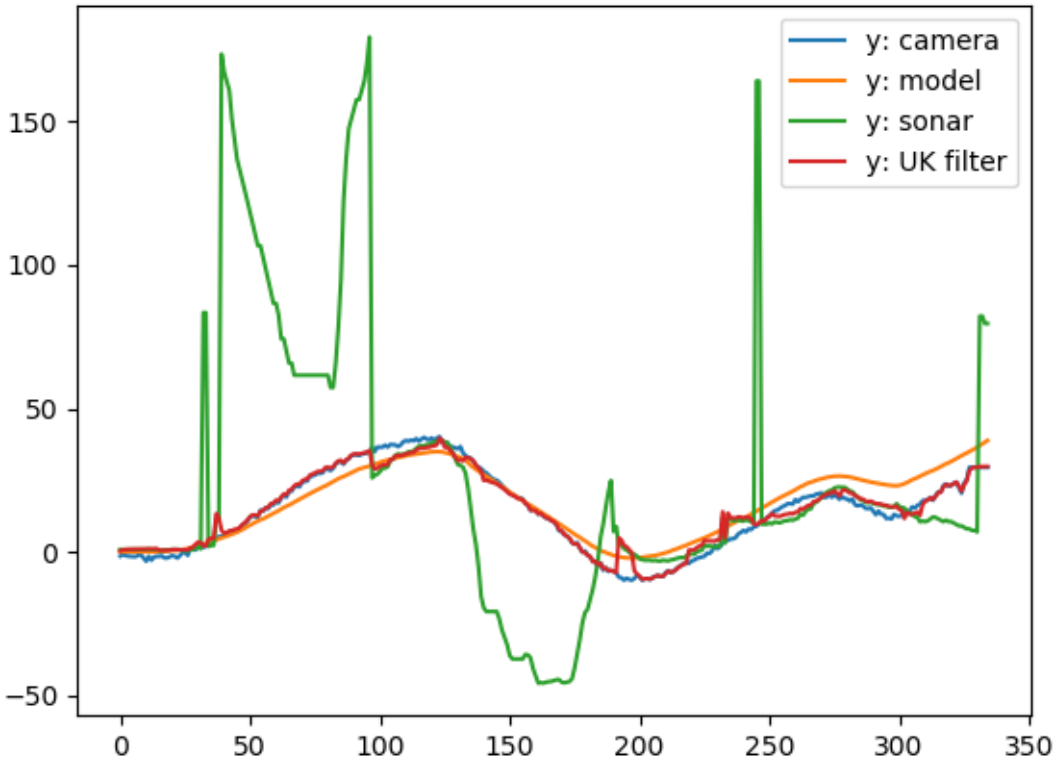

Figure 9: UKF: y values

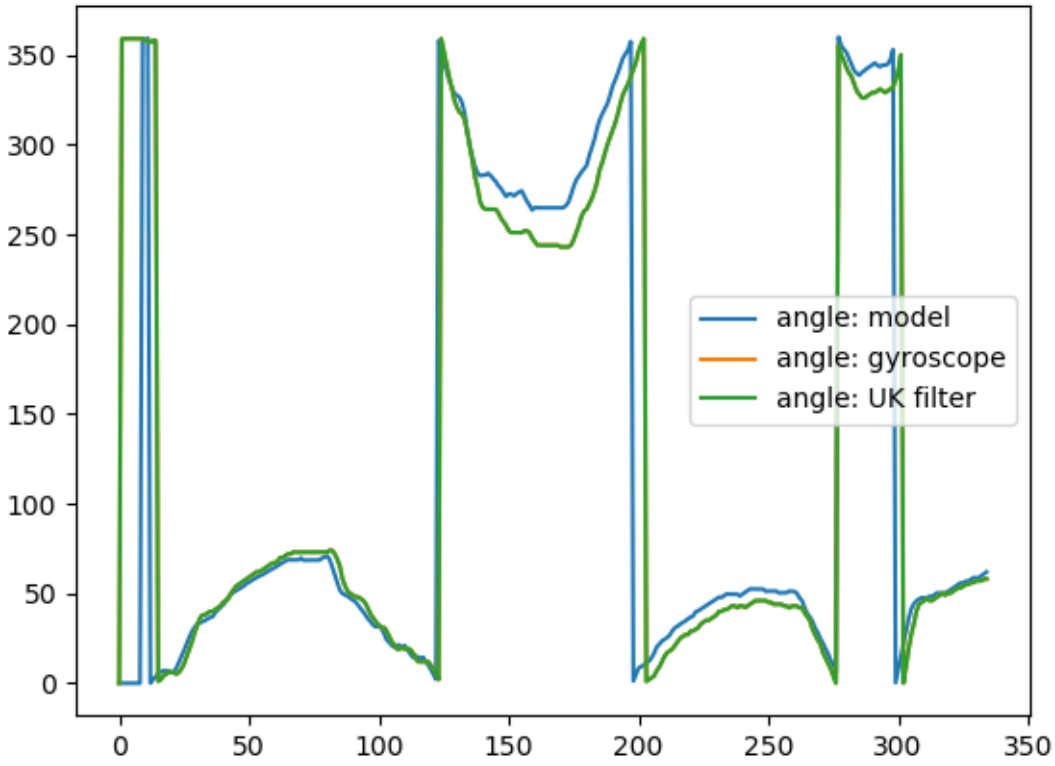

Figure 10: UKF: theta values

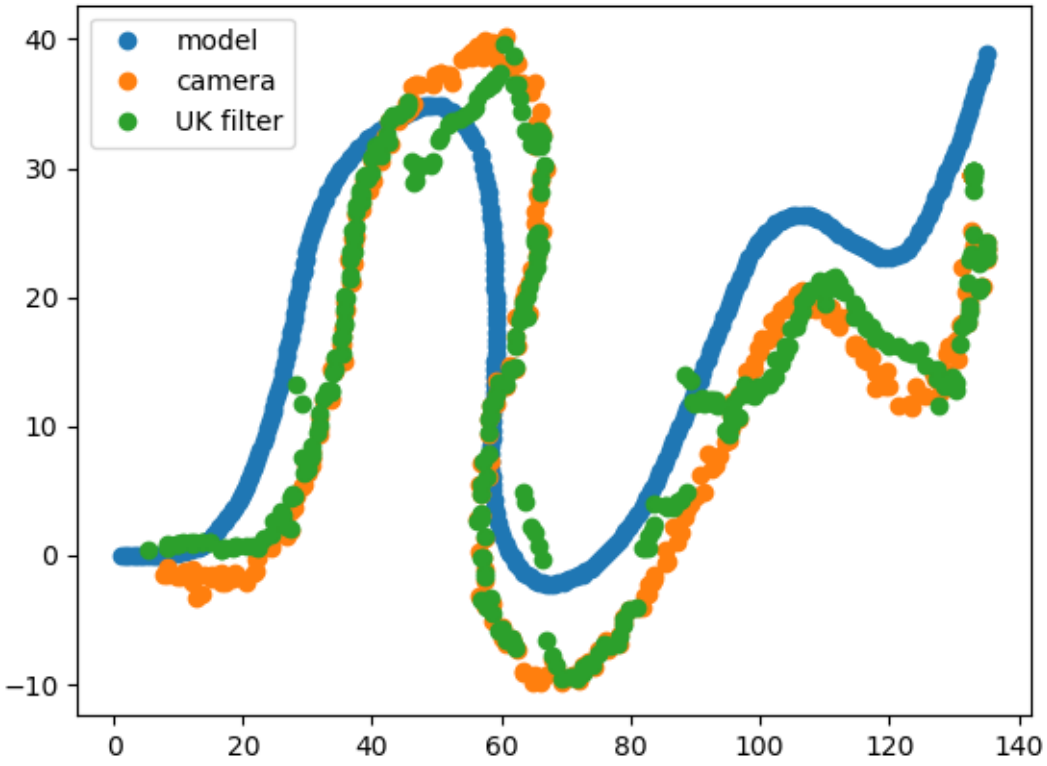

Figure 11: UKF: xy values

As it can be observed, it is almost the same as from EKF. However, as it can be observed from 9, filtered value become smoother.

2.2.5 Particle filter

Particle filter uses stratified resample algorithm, since it is one of the most popular universal algorithm.

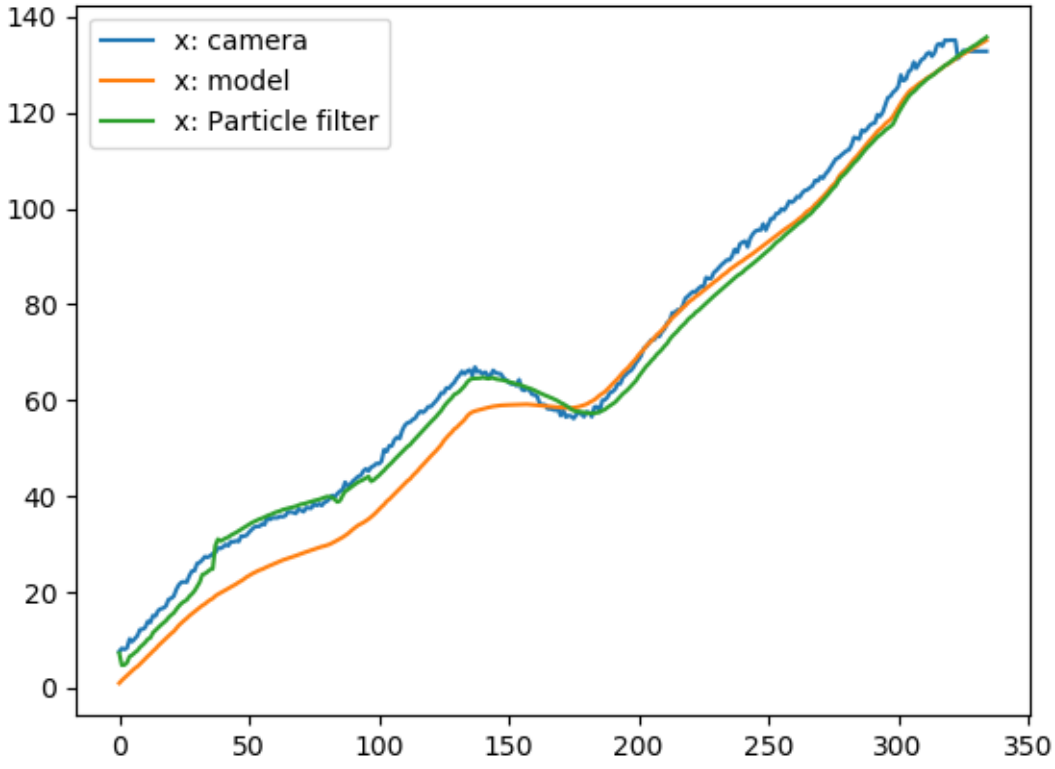

Figure 12: PF: x values

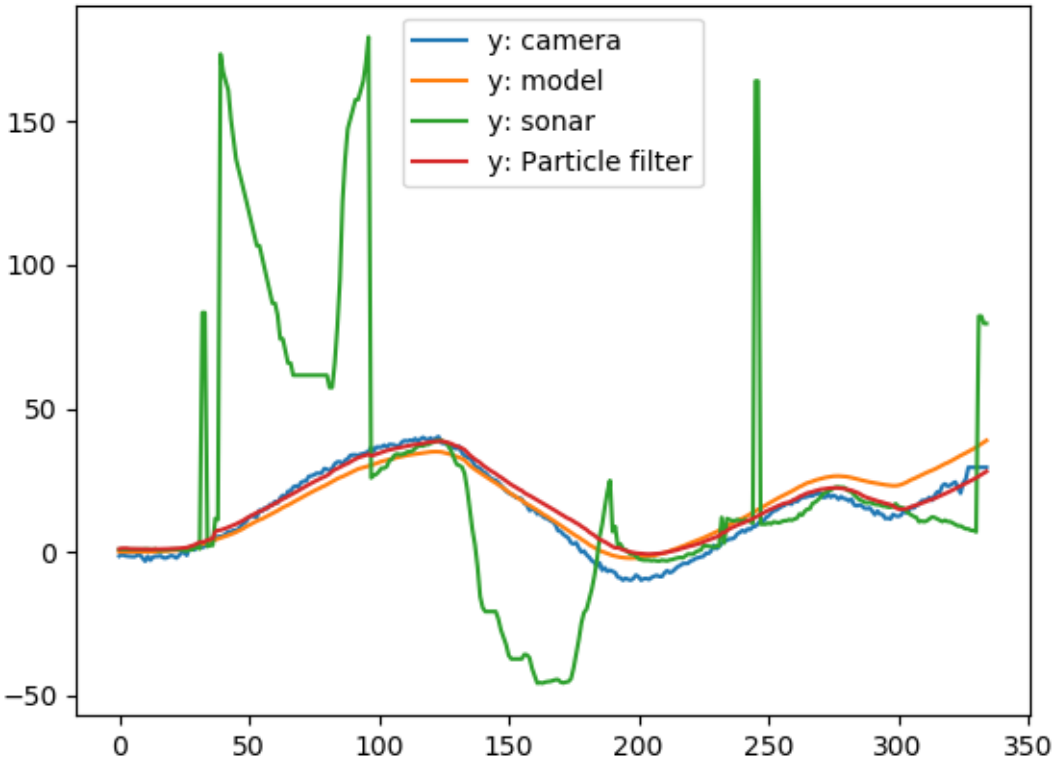

Figure 13: PF: y values

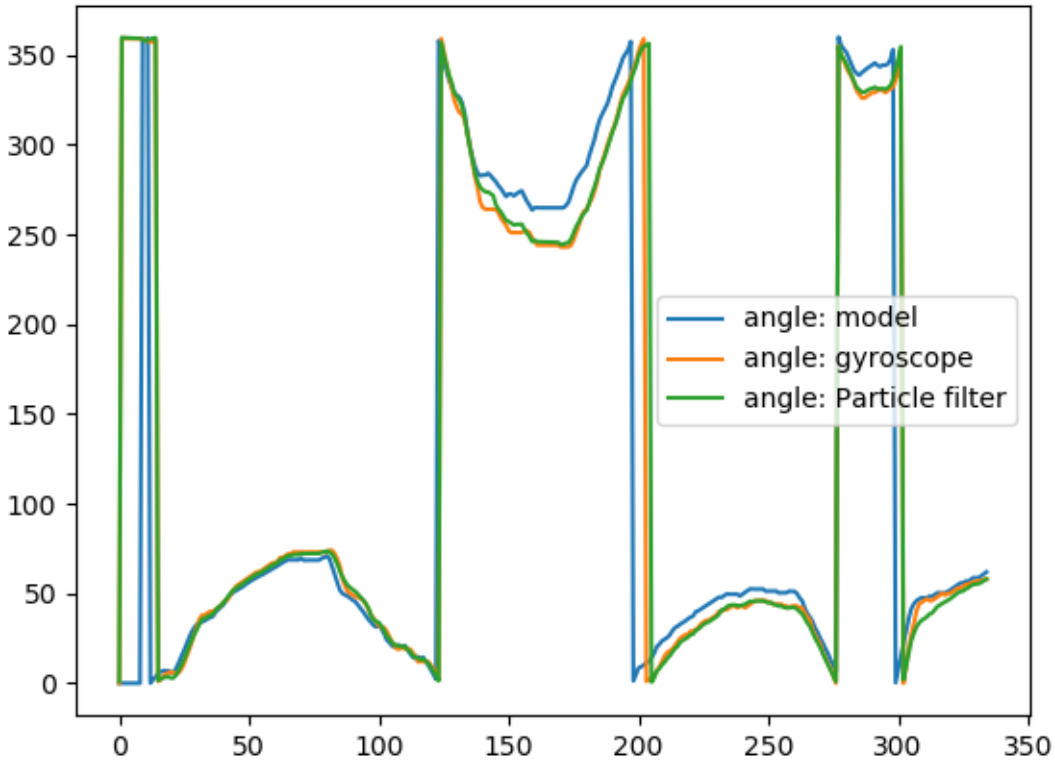

Figure 14: PF: theta values

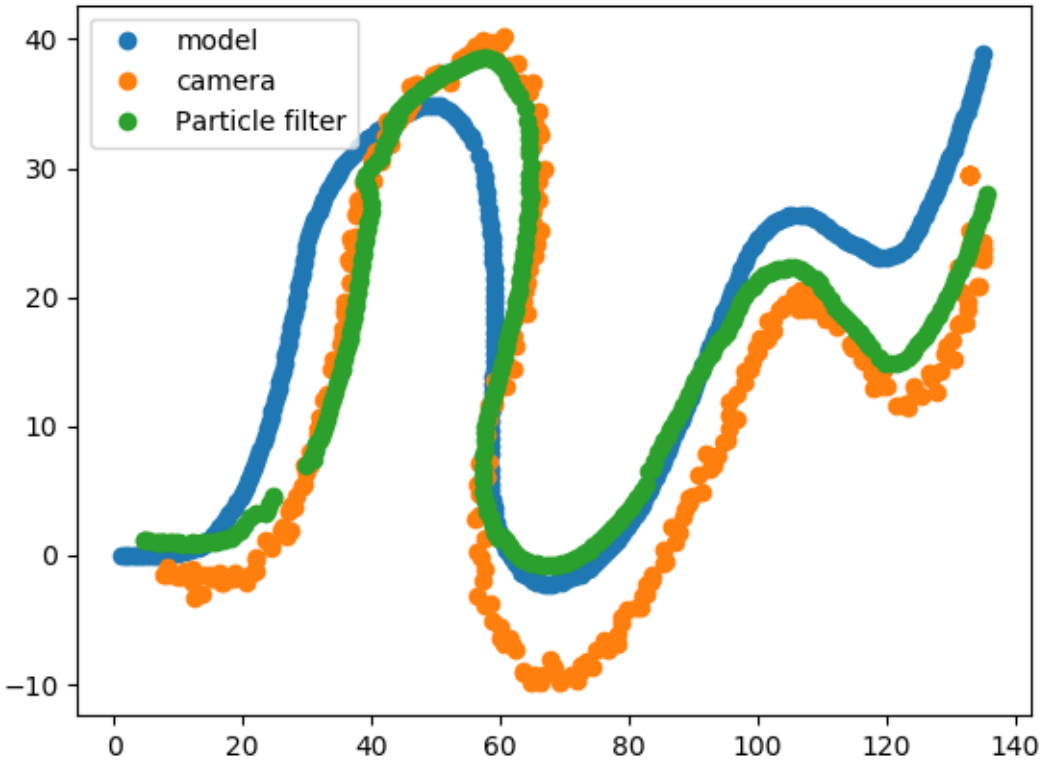

Figure 15: PF: xy values

As it can be observed, particle filter is the most robust to model-observations discrepancy and provided smooth result. However, there is discontinuity, related to sonar measurements.

3 Summary

The unknown trajectory was successfully estimated from the given data. However, it could be done without any filtration. This is my assumption, because, I believe neither model nor camera measurements are correct. That is why filters gives "strange results".

In such situation it seems to be reasonable to try some kind of averaging, which was performed by the PF.