Report

Sensing, perception and actuation

Homework assignment №3

Student: Oleg Rodionov

Innopolis. 2019

GitHub Link

You can download the all stuff and see the video on this link.

https://github.com/rodosha98/SPA-Homework3.git

In task 1 I've used MatLab. In task 2 - Python (Jupiter notebook).

Task 01. Kinect

The task is to define the centre of a 3D object. My 3D object is a red box, you can see it on the figure 1.

Figure 1: Red Box

I solve this problem in MatLab, using special toolbox to work with point cloud.

Step1. Obtaining the point cloud and preprocessing

I used standard function pcread( ) to read the ply file. You can see the whole cloud picture on the Figure 2.

Figure 2: Point cloud

Also that is needed to pre-process the cloud, remove all NaN values and some noise with help of special functions.

Step2. Object isolation.

I've used image colour picker to obtain RGB values of my red box and after that I have written an algorithm, which allows to filter the whole cloud and get points, which RGB values match to my red box. The RGB domain is:

Red: 90 - 170

Green: 0 - 70

Blue: 0 - 70

The isolated box is on figure 3.

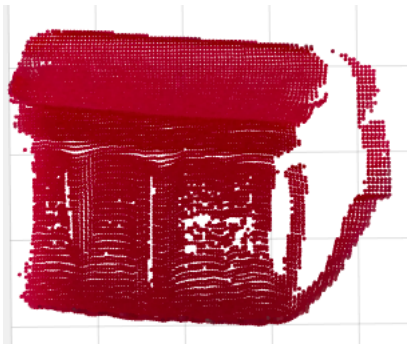

Figure 3: Box cloud

Step3. Front side

To isolate the front side I launched pcfitplane() function, which extract points from Cloud, using RANSAC and given threshold. Points fall into the cloud if the distance from them to the plane does not exceed the threshold. The front side has at the high density of clouds , that is why the function choose and fit plane on that points. But also that side isn't a plane, it has a special shape(because the box has concavity). The extracted cloud you can see on Figure 4.

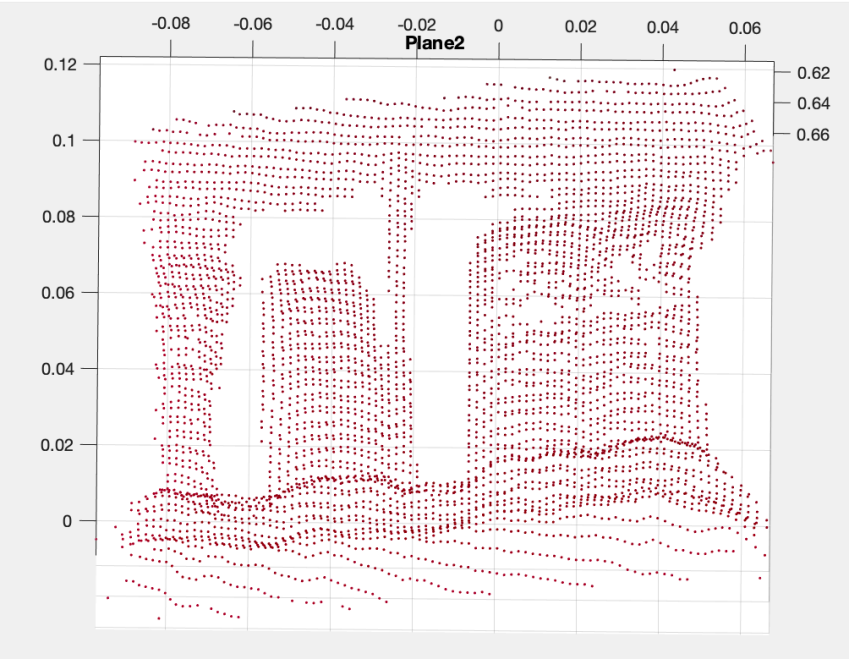

Figure 4: Front side

Step4. Center Search Algorithm

To define the centre coordinates that's needed to sum all coordinates of each point separately for x, y and z, and then just divide the sum by number of points in the cloud. Іэму also defined edge points with max x, y and z values.

The centre is the blue point on the the figure 5. This corresponds to the selected cloud, and to a real object with some large error

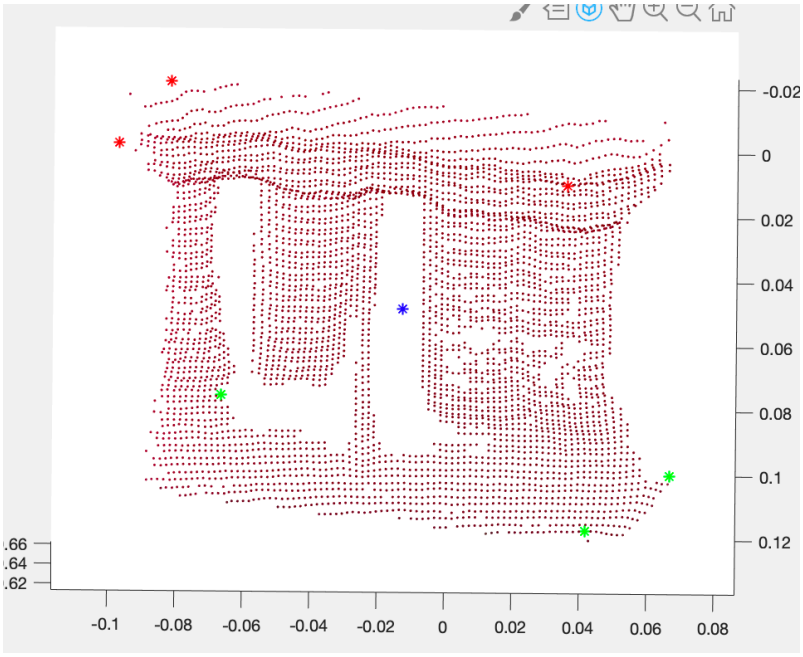

Figure 5: Front side centre cloud

Step5. Top side

In this step all actions repeat, after first fitting we also obtain the remaining cloud, which is given on the figure 6.

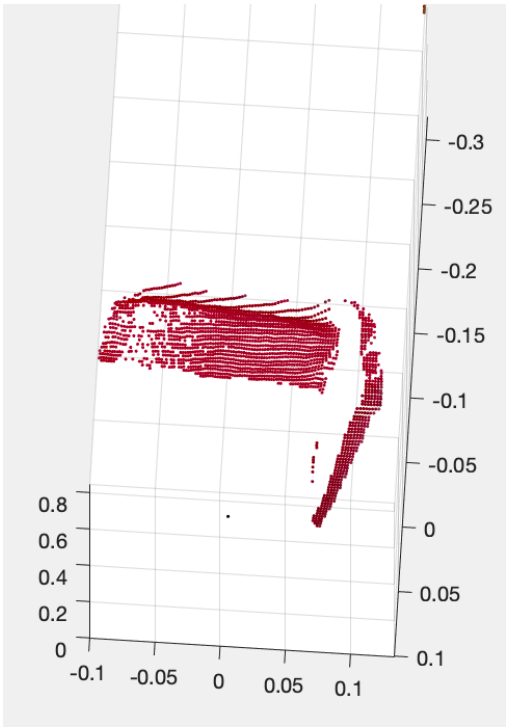

Figure 6: Remaining cloud

Apply the fit method again. It picks out the top plane of the box. The plane is depicted on the figure 7.

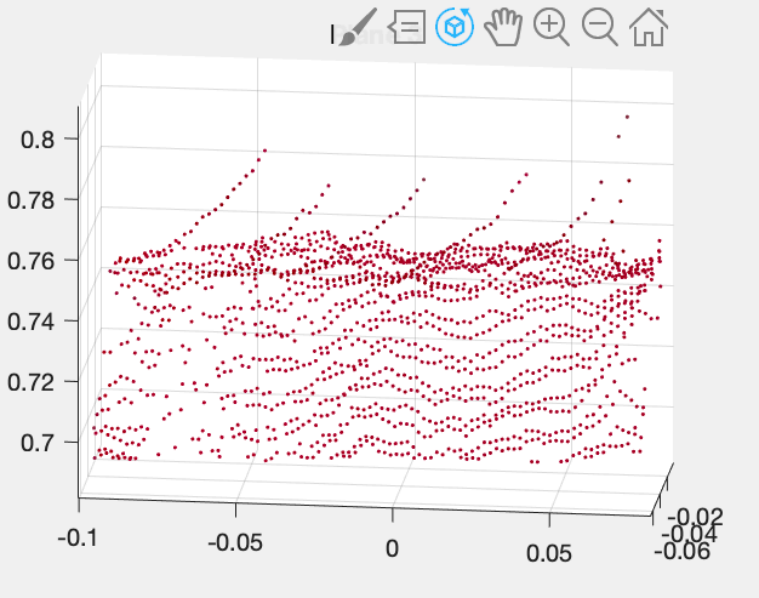

Figure 7: Plane 3

Center 2:

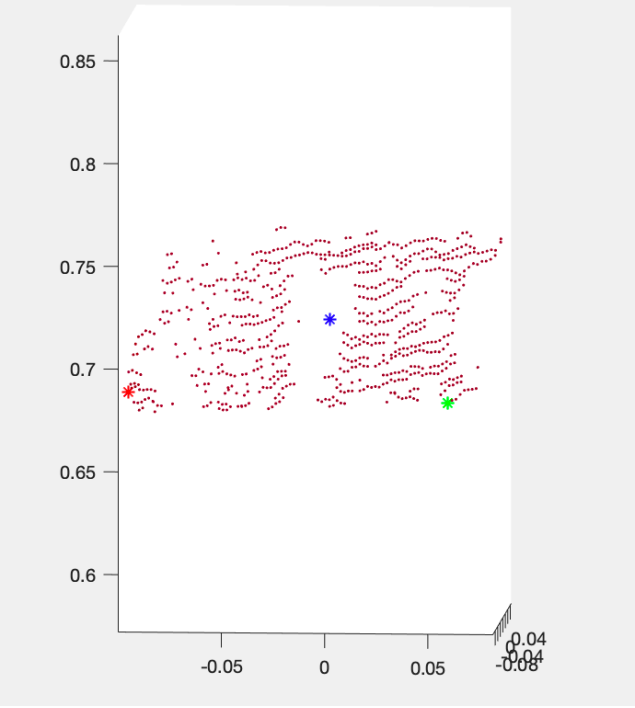

Figure 8: Center 2

Step6. Center of 3D Box

We can find a normal to the top plane, taking 3 points from this plane and find coefficients of the plane (A, B, C, D), where A, B, C - normal.

Next, we can also roughly estimate the height of the front side, subtracting y coordinate from max and min points. We can estimate it more accurate, if we know the real sizes of the box(it's not a cube).

So, if we find a product of normal to the plane and half of the height and then sum it with centre 2, we will define the coordinates of the centre of the whole box. You can see it in the Figure 9.

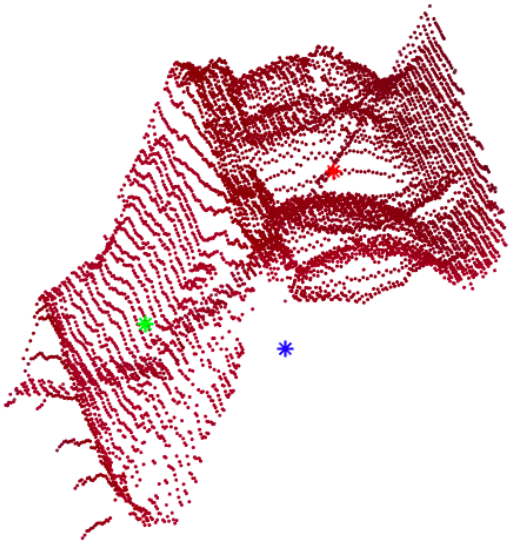

Figure 9: Center of the Box(Blue)

The definition of the centre that way isn't precise.

Task 02. Kalman Filter

Stage 1. Data obtaining and preprocessing

In this task there are required to estimate real trajectory using sensor fusion. I used data from accelerometer and GPS, and only horizontal plane.

Initially, I convert the distance from latitude and longitude to world frame x, y. That is needed to use UTM convention and UTM python library. According to this convention, the entire globe is divided into zones and all distances are counted from absolute zero. So, the Innopolis area is located in the zone 39-U. The path from GPS you can see on the figure 10. And data from accelerometers on figure 11.

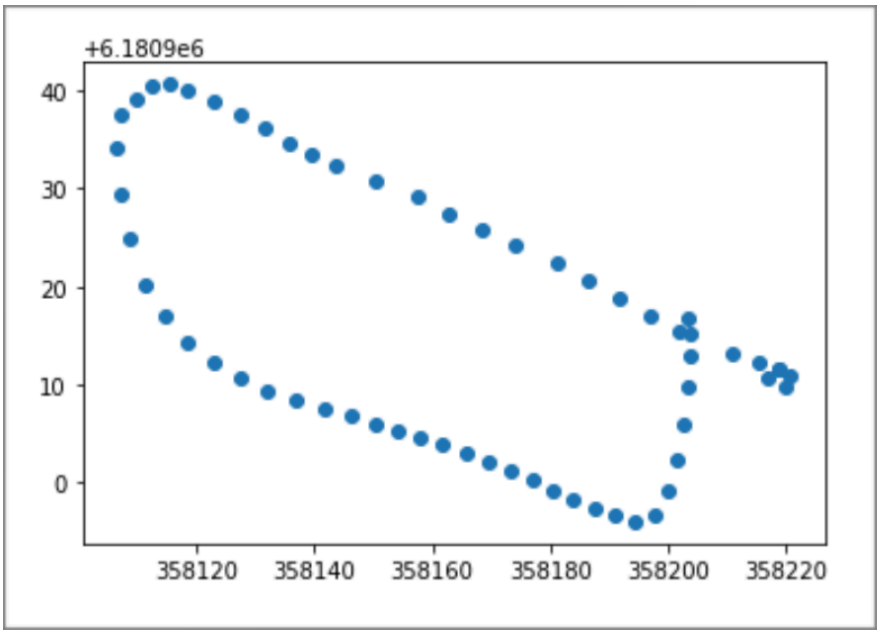

Figure 10: GPS data

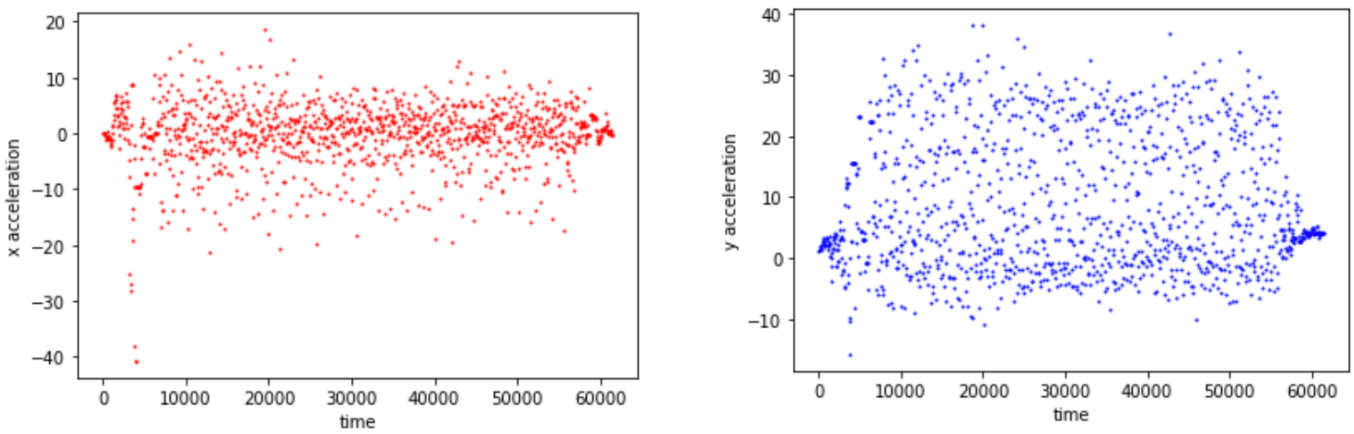

Figure 11: Acceleration data: x acceleration - left, y acceleration - right

Stage 2. Data Fusion

We have 2 datasets, but measurements were taken at different points in time and the dataset sizes do not match. Because of it, I've used interpolation approach to resize GPS dataset. It's a simple linear interpolation method, that connects 2 points and take average between them. The result of interpolation you can see on the figure 12

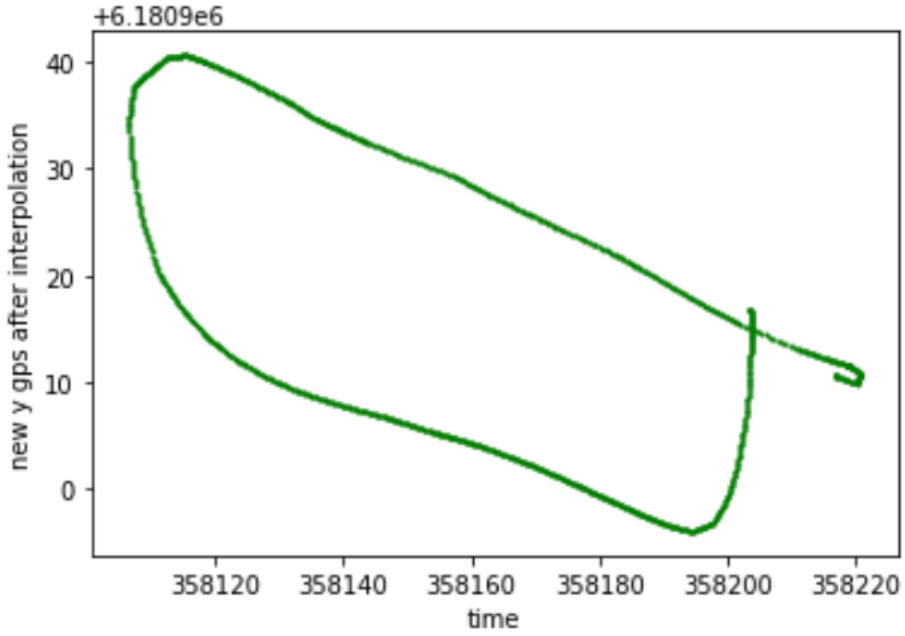

Figure 12: GPS measurements after the interpolation

The time is represented in millisecond and this must be taken into account

Stage 3. State space vector and filter structure

State vector is:

$$X = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

We will estimate and upgrade position (x, y) and velocities, because it depends from acceleration and we know initial conditions.

Dynamics matrix A:

$$A = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Control we will implement by using acceleration, cause we know the law.

Control matrix B:

$$B = \begin{bmatrix} \frac{dt^2}{2} & 0 \\ 0 & \frac{dt^2}{2} \\ dt & 0 \\ 0 & dt \end{bmatrix}$$

It has shape (4 x 2). Control acceleration matrix u has the following view:

$$u = \begin{bmatrix} a_x \\ a_y \end{bmatrix}$$

It contains data from accelerometer.

Stage 4. Guessing and approximation of randomness

In Kalman filter we have model and measurement noises. And we can make some simplifications:

1) We tried to run approximately with the constant speed and data from accelerometer proves it. So, we can trust this data and find distributions from them.
2) We can estimate the accuracy of determining the GPS position(I've established this value as 10 metres, what is accurate enough for GPS system.
3) Noise of the model. To get better results, we are needed to add small noise in our model or the trajectory will be very noisy.

Stage 5. Kalman filter algorithm

First of all we need to define initial state vector X0 and state covariance vector P0. Kalman filter algorithm is iterative and it and it converges at the end of the cycle.The initial vector corresponds to the initial position from gps and the initial velocity is zero. I took the initial covariance matrix with zero elements.

PREDICTION PART

Step 1. State vector prediction

$$X_{kp} = A X_{k-1} + B u_k + w_k$$

w_k - noise of the model, it's represented as vector:

$$w_k = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{\dot{x}} \\ \sigma_{\dot{y}} \end{bmatrix}$$

Steps 2 and 3. State Covariation matrix prediction

$$P_{kp} = A P_{k-1} A^T + Q_k$$

Q_k - process noise covariation matrix.

$$Q = \begin{bmatrix} \sigma_x^2 & 0 & \sigma_x\sigma_{\dot{x}} & 0 \\ 0 & \sigma_y^2 & 0 & \sigma_y\sigma_{\dot{y}} \\ \sigma_x\sigma_{\dot{x}} & 0 & \sigma_{\dot{x}}^2 & 0 \\ 0 & \sigma_y\sigma_{\dot{y}} & 0 & \sigma_{\dot{y}}^2 \end{bmatrix}$$

Correlation between x and $\dot{x}$ is equal to 1. The same to y axis.

UPDATING PART

Step 4. Kalman Gain

$$K = \frac{P_{kp} H^T}{H P_{kp} H^T + R}$$

R - measurement noise correlation matrix

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

H is matrix to get the same size of vectors. K is changing during each iteration. If R is equal to 0, than means that we absolutely trust to measurements(K = 1) and vice versa.

Step 5. Measurements

$$Y_k = C Y_{km} + Z_m$$

It’s the data from GPS. C matrix shows, which parameters from state vector we are measuring.

Step 6. Updating State Vector

$$X_k = X_{kp} + K[Y - HX_{kp}]$$

Step 7. Updating the covariation state matrix

$$P_k = (I - KH)P_{kp}$$

Results

You can see the final trajectory in the figure 13:

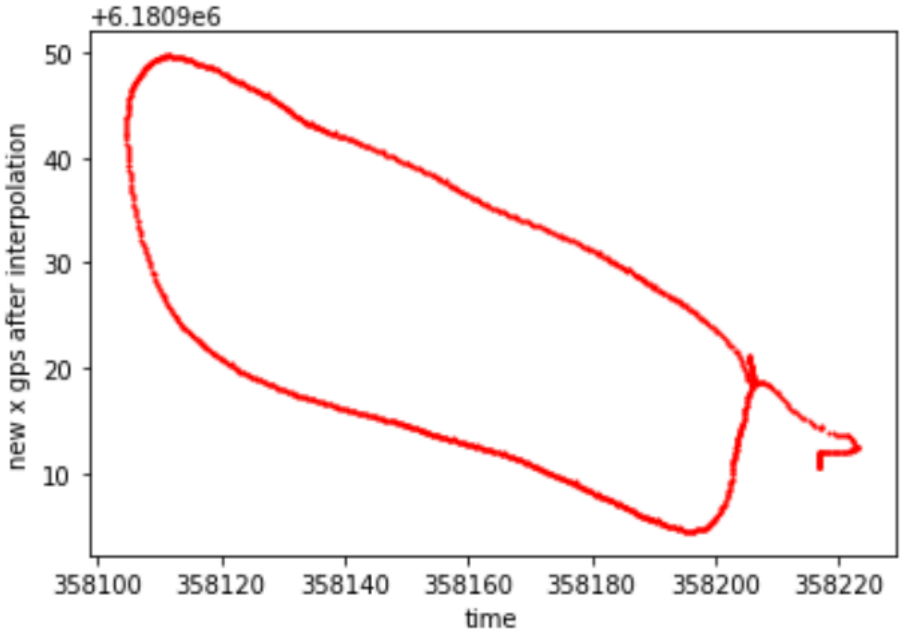

Figure 12: GPS measurements after the interpolation

If we add big noise, the trajectory will also be noisy. Here we can see only small changes, also because we didn’t get wrong measurements from GPS and Kalman gain was close to 1, what means that we are trusting to measurements more.