Sensing, Perception and Actuation

Kinect. Sensor fusion (HA3)

Ilia Sevostianov

November 14, 2019

Abstract

In this homework assignment I was asked to work with Kinect to obtain depth map or to get the 3D point cloud and obtain the center of the chosen item. Also, I needed to implement sensor fusion for the another task in orded to get the trajectory of my running (GPS+smth else).

Tasks

1. **Task 1**: To get started your homework 03, it is needed to place a 3d object(e.g., cube, cylinder, etc) in an appropriate way with respect to Kinect. Afterwords, Kinect-2 should be used to get the depth map. Associate depth map with RGB information in order to isolate the object, which is to be extracted from the ground plane and other foreground objects if there are. You may have to use RANSAC or some other algorithms to extract the object and find the center point of the object in the 3D space or image plane relative to the Kinect.

2. **Task 2**: Take your smart phone and run for about 100 meters with a constant speed approximately. Your task is to estimate the trajectory where you ran with your phone. To estimate the trajectory, you are asked to incorporate sensor measurements that are available in your mobile phone (e.g., accelerometer, gyroscope, GPS sensor, etc). You should use **multidimensional Kalman filter** with sensor fusion to solve this task. In the report clearly explain all the assumptions you made.

Submit

Please upload the single **zip file** which includes your **source code**, **report** (valid for both tasks), **dataset you collected in task 2** and **video which shows the depth map acquisition from the Kinect** while including what you did and why you did it in the report. You may upload your video into some file server and put the reference in the report.

Task 1: Kinect

At first, I needed to connect Kinect to my computer and set up it in order to be able to work with it. I used this source for it: library for Kinect2

Secondly, I got the point cloud using this source: libfreenect2pclgrabber. I needed it because I had Ubuntu but not Windows.

You can see here the video which shows the depth map acquisition from the Kinect.

And finally, I wrote the program which allowed me to obtain the center of the chosen item. The idea is that we have the color of item. And we can simultaneously find the planes in the gotten 3D point cloud and check the points' color there to compare with the color of the item. So, we can take points from the planes and store them in arrays and after that get the mean of them, allowing to obtain the center of the object. The approximate algorithm you can see further.

The result you can see on the figure 1.

Algorithm 1 Finding center of the item on the Point Cloud from Kinect

procedure FIND THE OBJECT'S CENTER
    read the ***remainPtCloud*** from file.
    Initialize ***Cloud*** = ***remainPtCloud***
    Set the maximum point-to-point distance for plane fitting (cm) ***maxDistance***
    Set the Color Detection Error ***cerr***
    Set the subject's color
    Set intervals for given color to be detected
    $mean_x 0 = 0$
    $mean_y 0 = 0$
    $mean_z 0 = 0$
    $p = 0$ -number of subject planes
    for each integer i in m **do**
        Set some rules to avoid empty area to fit
        Set the area where to fit the plane ***roi***
        Find points in ***roi*** - ***sampleIndices***
        Calculate the size of ***sampleIndices*** and make rule to avoid inadequate array
        Run $PlaneDetectionfunction$
        Calculate the number of points in the plane ***k***
        $n = 0$- number of points in subject plane
        $plus = 0$- the detector of the subject plane
        $Ax = []$
        $Ay = []$
        $Az = []$
        for each integer j in $k(1)$ **do**
            if the point has the color similar to the subject's one, then:
            $n = n + 1$
            find point in the plane that has the same color as the object
            Add coordinates of this point to arrays Ax, Ay, Az
            $plus = 1$
        end for
        if $plus == 1$ and $n > 65$ (avoiding invalid areas):
        Calculate maxs and mins of arrays A and find means
        $p = p + 1$
        Update coordintates of the center_0
    end for
    Plot the result
end procedure

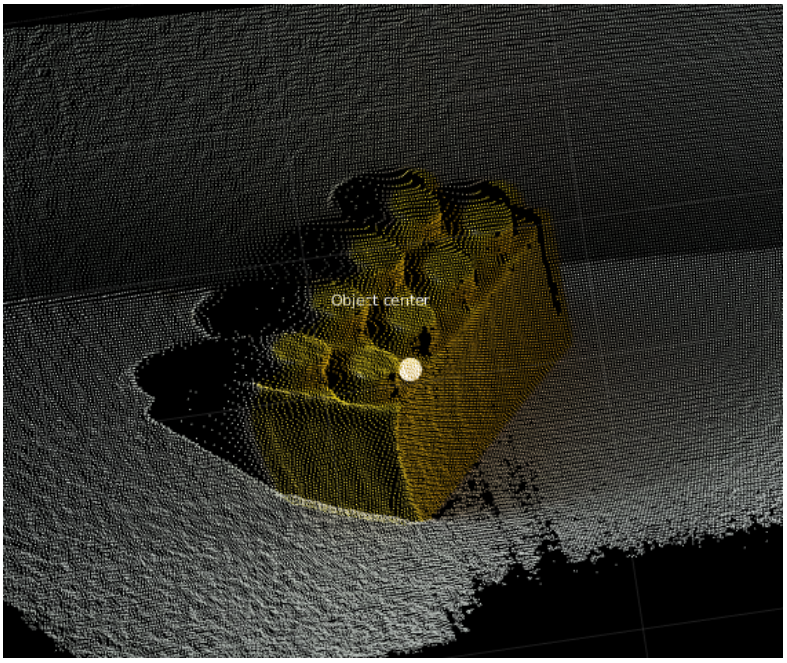

Figure 1: The object's center

As I concuded, Kinect gives not so accurate point cloud. So, even after removing invalid points I had some noize in obtaining the object's center.

Task 2: Sensor Fusion

At first, I dug into the theory in order to undrestand Multidimensional Kalman Filter better. So, It was very helpful to see the lab and this two links: link 1, link 2.

Theory

We have already implemented one dimensional Kalman Filter. But now we have the mean and the variance inside a matrix on which all the operations are performed. And we have sensor fusion, so, this results in a Kalman filter with the following state variables (**System state X**).

At the beginning we will have to initialize with an initial state.

$$x = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} \tag{1}$$

where

- x and y - position (coordinates) of the mobile phone;
- $\dot{x}$ and $\dot{y}$ - velocity of the mobile phone

I could take x and y from gps data. Also, I decided to take velocities $\dot{x}$ and $\dot{y}$ equal to zero. Because I started recoring data from sensors when I had zero velocity.

Also, an uncertainty must be given for the initial state - **covariance matrix P**. This matrix is most likely to be changed during the filter passes. It's changed in both the predict and correction steps. I assumed that my sensors are very accurate, so initially I put zeros here.

The core of the filter, however, is the following definition, which we should set up with great understanding of the physical context. In our case, the dynamics in matrix notation (**Dynamics matrix A**) is as follows:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where

- dt - calculatation step

This states "where" the state vector moves from one calculation step to the next within.

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t \tag{3}$$

As the movement of phone may also be disturbed, this is where the **process noise co-variance matrix Q** is introduced. This matrix tells us about the filter , and how the system state can "jump" from one step to the next. Imagine the vehicle that drives autonomously. The matrix is a co-variance matrix containing the following elements:

$$\mathbf{Q} = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\dot{x}} & \sigma_{x\dot{y}} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{y\dot{x}} & \sigma_{y\dot{y}} \\ \sigma_{\dot{x}x} & \sigma_{\dot{x}y} & \sigma_{\dot{x}}^2 & \sigma_{\dot{x}\dot{y}} \\ \sigma_{\dot{y}x} & \sigma_{\dot{y}y} & \sigma_{\dot{y}\dot{x}} & \sigma_{\dot{y}}^2 \end{bmatrix} \tag{4}$$

An external control variables is possible via the **control matrix B**. The **matrix u** will contain the robotic input of the system which could be the instantaneous acceleration from IMU.

The filter must also be told what is measured and how it relates to the state vector. So, let's introduce **measuring matrix H**. We measure position in x and y.

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{5}$$

A measurement uncerainty must also be stated here. This measurement uncertainty indicates how much one trusts the measured values of the sensors.

$$\mathbf{R} = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \tag{6}$$

Additionally, we have control by using acceleration, because we know the law.

$$\mathbf{B} = \begin{bmatrix} \frac{dt^2}{2} & 0 \\ 0 & \frac{dt^2}{2} \\ dt & 0 \\ 0 & dt \end{bmatrix} \tag{7}$$

Approximations of randomness:

- I tried to run approximately with the constant speed.
- I can estimate the accuracy of determining the GPS position. And I've established this value as 25 meters.
- Noise of the model. To get better results, I was needed to add small noise in my model, otherwise the trajectory will be very noisy.

Kalman algorithm:

- Prediction part
  - State vector prediction

$$\mathbf{X}_{kp} = \mathbf{A}\mathbf{X}_{k-1} + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k \tag{8}$$

$\mathbf{w}_k$ - noise of the model, it's represented as vector:

$$\mathbf{w}_k = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{\dot{x}} \\ \sigma_{\dot{y}} \end{bmatrix} \tag{9}$$

- State covariation matrix prediction

$$\mathbf{P}_{kp} = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A} + \mathbf{Q}_k \tag{10}$$

- Correction part
  - Kalman Gain

$$\mathbf{K} = \frac{\mathbf{P}_{kp}\mathbf{H}^T}{\mathbf{H}\mathbf{P}_{kp}\mathbf{H}^T + \mathbf{R}} \tag{11}$$

  - Measurements

$$\mathbf{Y}_k = \mathbf{C}\mathbf{Y}_{km} + \mathbf{Z}_m \tag{12}$$

    It's the data from GPS. C matrix shows, which parameters from state vector we are measuring.
  - Update State Vector

$$\mathbf{X}_k = \mathbf{X}_{kp} + \mathbf{K}[\mathbf{Y} - \mathbf{H}\mathbf{X}_{kp}] \tag{13}$$

  - Update the covariation state matrix

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{P}_{kp} \tag{14}$$

Solution

Let's see our data from sensors:

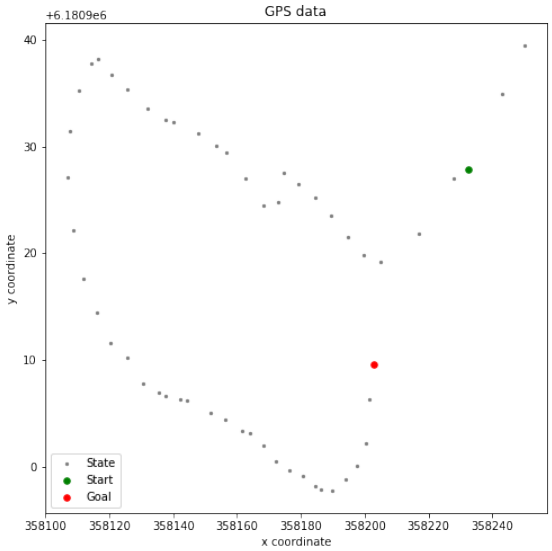

Figure 2: GPS data

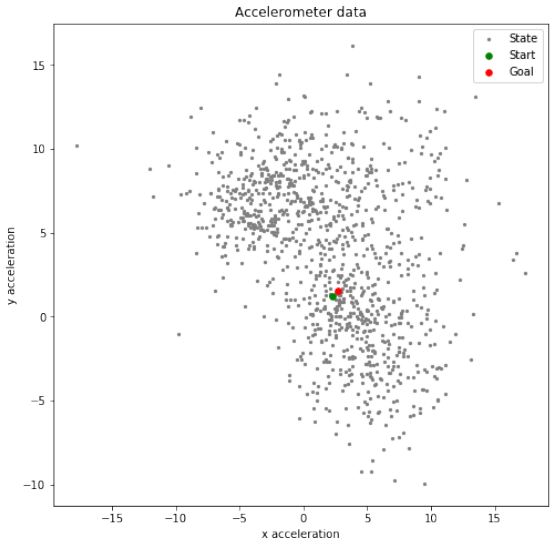

Figure 3: Acceleration data

I run approximately 100 meters around the alley in front of the 3rd dormitory of IU. And the data from GPS is pretty the same as my path.

Also, I needed to interpolate GPS data because of the smaller amount of points than in acceleration data. You can see it in figure 4

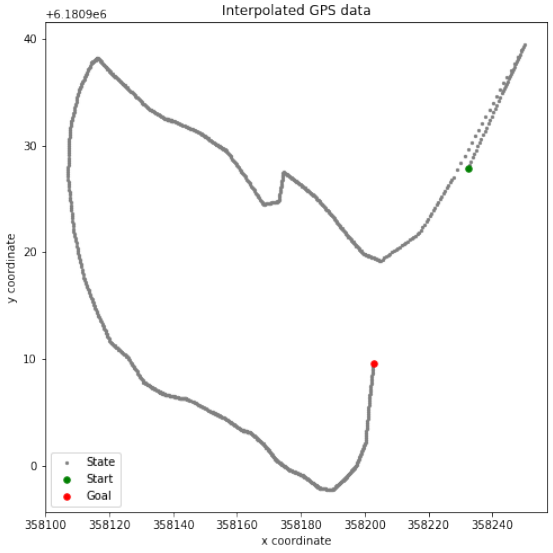

Figure 4: Interpolated GPS data

Results after sensor fusion by Kalman algorithm you can see in figure 5.

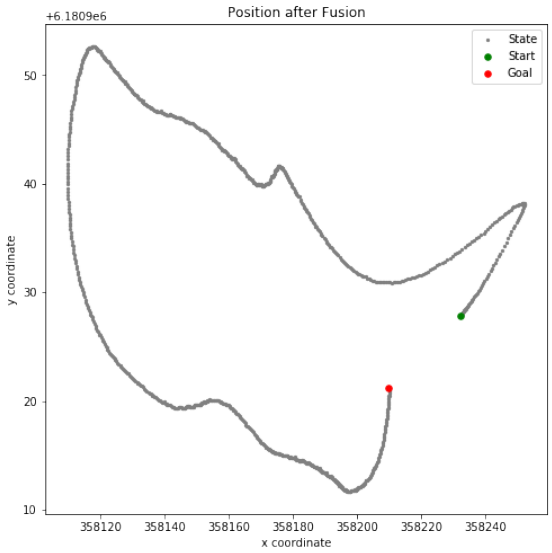

Figure 5: Fusion GPS data

The bigger noise, the noiser the trajectory will be. Here we can see only small changes, because we didn't get so inaccurate measurements from GPS. And Kalman gain was coming to 1, what means that we trust out measurements.

My code you can find here.