Position Estimation Task – Lilach Biton

1) 1. First I started with Only target dataset when Ego is not moving, and ego coordinate system is equal to Ego_0 through all frames.

Pre-processing

To understand the motion between each frame in the real world and in image plane, I processed the data and plot the following graphs, using the InitialProcessing function at the attached LilachBitonTask.

First, I created a 3D plot of real-world points (at ego_0 coordinates) from all the frames and I added the normal vector for each plane.

Since, it is given that all the points given of target vehicle sit on a single plane, it is possible to create the normal vector for each plane using 3 points from each frame.

Since the vehicle is a rigid body, I assumed that the normal vector has information about the target vehicle orientation.

I calculated the normal vector using this formula:

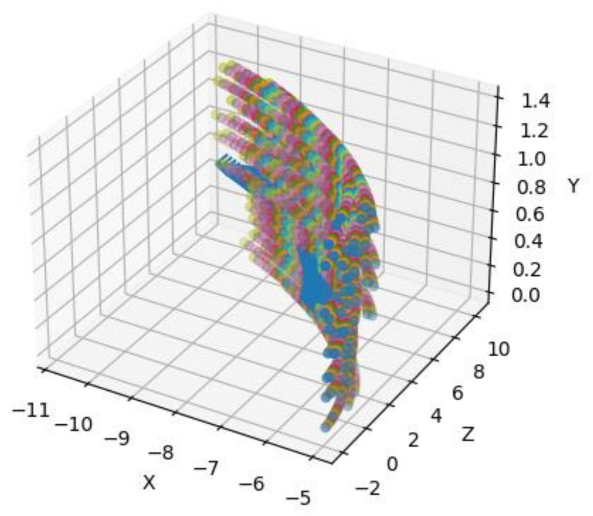
$$\vec{n} = \overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3} \rightarrow \hat{n} = \frac{\vec{n}}{|\vec{n}|}$$

When,

$$\overrightarrow{P_1P_2} = \overrightarrow{P_2} - \overrightarrow{P_1}\,, \qquad \overrightarrow{P_1P_3} = \overrightarrow{P_3} - \overrightarrow{P_1}$$

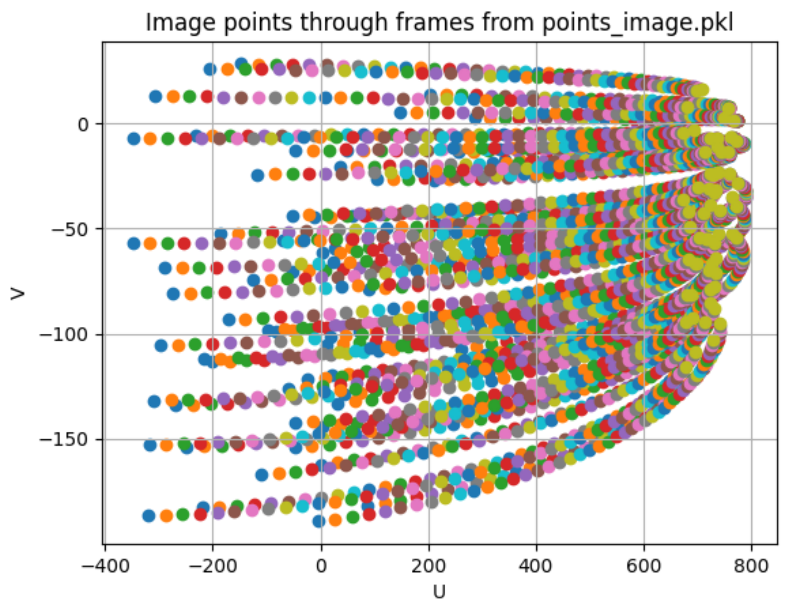
In addition to that I calculated the C.G. point for each frame in the real word points, and I assume that this point represents the center of the target vehicle, and I'm relate to this point when I'll calculate the motion.

Real World points and normal to each plane from points_world.pkl

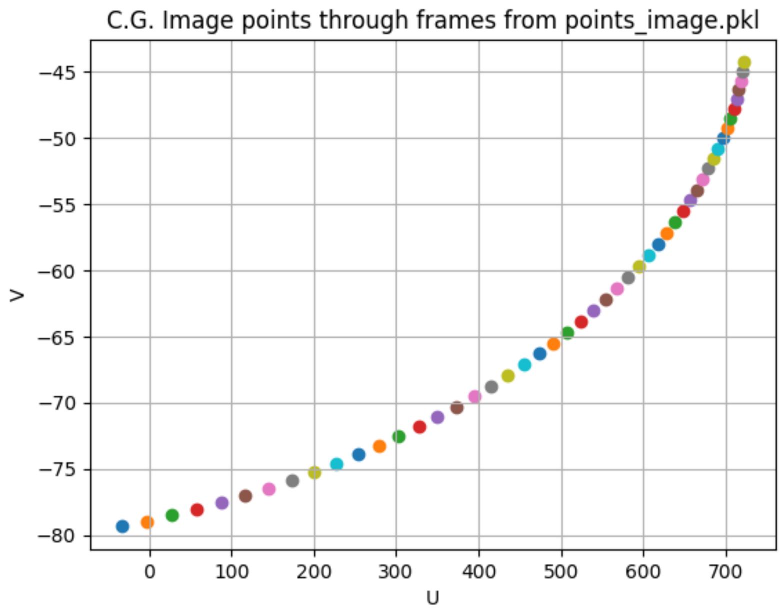


It is possible to see that the target vehicle is moving on X-Z plane only which is correlate to the Flat world assumption (dy = 0).

The target vehicle is become farther away from ego_0 and turning to the left and changing is heading through frames.
The heading of the vehicle is defined by the yaw angle (angle around Y axis) only because of the Flat world assumption.
After I understood the real-world motion, I observed the motion in the image plane through frames:



First frame points are blue color and last frame points are ochre color.
It is possible to see that distance between points are getting smaller through frames since the target vehicle is become farther away from ego.
To understand more about the movement in the image plane I calculated C.G. point for every frame and plot it through frames:



It is possible to see that the C.G. points are moving up and to the right side of image plane, and the distance between each point is getting smaller.
The points are moving to the right side of the image because the camera is directed to the left side of the vehicle and when the target vehicle is become farther away from ego it seems he is moving to the right, the points are moving to the upper side of the image because the target vehicle is become farther away from ego, and the image catch more of the close road.
The distance between points is getting smaller because the distance between target vehicle and ego vehicle is getting bigger.
<u>Solution</u>
Since it is given that the camera is undistorted there is strong connection between points in real world and points in image plane as I described. So, in order to transfer all points from image plane to real world (in camera coordinates) I used the Pinhole camera model which connect points from image plane to points in real world using camera focal length which is the value on the main diagonal of given camera matrix K:

$$\begin{pmatrix} u \\ v \end{pmatrix} = -\frac{f}{z}\cdot\begin{pmatrix} x \\ y \end{pmatrix}$$

To fully solve this transformation, it is needed to know the distance between camera to the point in real world, <u>or</u> another information about real world point to cancel one degree of freedom.
Since I already know that dy=0 through all frames and all datasets, and since I can use first frame of world_points.pkl it is possible to fully solve this equation:

$$\begin{pmatrix} u \\ v \end{pmatrix} = -\frac{f}{z}\cdot\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{cases} u = -\dfrac{f}{z}\cdot x \\ v = -\dfrac{f}{z}\cdot y \end{cases} \rightarrow$$

$$\begin{cases} u_1 + du = u_2 = -\dfrac{f}{z_1 + dz}\cdot x_1 + dx \\ v_1 + dv = v_2 = -\dfrac{f}{z_1 + dz}\cdot y_1 + \cancel{dy} \end{cases} \rightarrow$$

$$\begin{cases} z_1 + dz = z_2 = -\dfrac{f\cdot y_1}{v_2} \\ x_1 + dx = x_2 = -\dfrac{u_2\cdot z_2}{f} \rightarrow \\ y_2 = y_1 \end{cases}$$

Where 1 index represent point from first frame and 2 index represent point from second frame, and x_i, y_i, z_i are real world point in camera coordinate.
Since I can use the first frame of world_points.pkl dataset which contain real world points in ego_0 coordinates all I need to do to solve this

equation is to transfer these points from ego_0 coordinate to camera coordinates using the given transformation matrices that way:

$$\begin{bmatrix} P_{c_1} \\ 1 \end{bmatrix} = (T_{c_1 e_1} \cdot T_{e_1 e_0}) \begin{bmatrix} P_{e_0} \\ 1 \end{bmatrix}$$

After I have all real-world points from the first frame in camera coordinates system I can solve this equation using matching points from image plane and find the real world points from the second frame in camera coordinate system.

The I can transfer these points back to ego_0 coordinates system using the opposite transform:

$$\begin{bmatrix} P_{e_0} \\ 1 \end{bmatrix} = (T_{e_2 e_0}^{-1} \cdot T_{c_2 e_2}^{-1}) \begin{bmatrix} P_{c_2} \\ 1 \end{bmatrix}$$

After I have all points from the second frame in real world at ego_0 coordinate system I'm calculating the C.G. point which represent the center of target vehicle location and easily calculate the movement by subtraction of real-world C.G. from the first frame:

$$[dx, dy, dz] = [x_2, y_2, z_2] - [x_1, y_1, z_1]$$

Now, in order to find the change in heading orientation it is possible to calculate the normal vector for each plane created in each frame, (it is given that all of the given points of target vehicle sit on a single plane), like shown before, and calculate the change of the normal direction which is equal to the change of the target vehicle orientation.

So, to find this change of orientation I calculated the angle between the two normal vectors using dot product:

$$\overrightarrow{n_1} \cdot \overrightarrow{n_2} = |n_1||n_2| \cos(d\theta) \rightarrow d\theta = \cos^{-1}(\widehat{n_1} \cdot \widehat{n_2})$$

This way I can find the movement and change of heading of target vehicle.

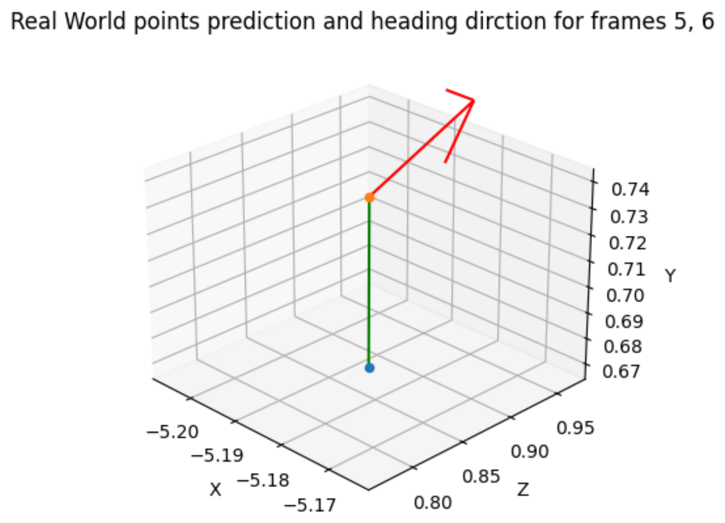
At the main function in the attached LilachBitonTask.py script it is possible to choose frame number, which transferred to Section1 function, and the motion estimation is calculated using the MotionEstTwoFrame function, from the imported PosestTask class I created, which takes image points from two consecutive frames and outputs the vehicle's movement. In order to find the error from this estimation I created the RealMotion function which calculate the Ground Truth position of the vehicle using points_world.pkl dataset and the heading change using T_{tt_0} given transformation, by calculating the transformation matrix between each frame and converting it to rotation vector, when the angle around Y axis is the heading change. Because of the Flat world assumption it's the same angle as of ego_0 coordinate system, I also calculated this angle using the normal method I showed with points_world.pkl dataset to confirm this way to be right.

<u>Results</u>

After I run over all frames, I found out that the maximal errors are:

$$Max\ \Delta\theta = 9.1 \cdot 10^{-9}[rad], Max\ norm(dx, dy, dz) = 6.4 \cdot 10^{-15}$$

It is possible to see the movement for two consecutive frames at 3Dplot, for example for frames 5,6:

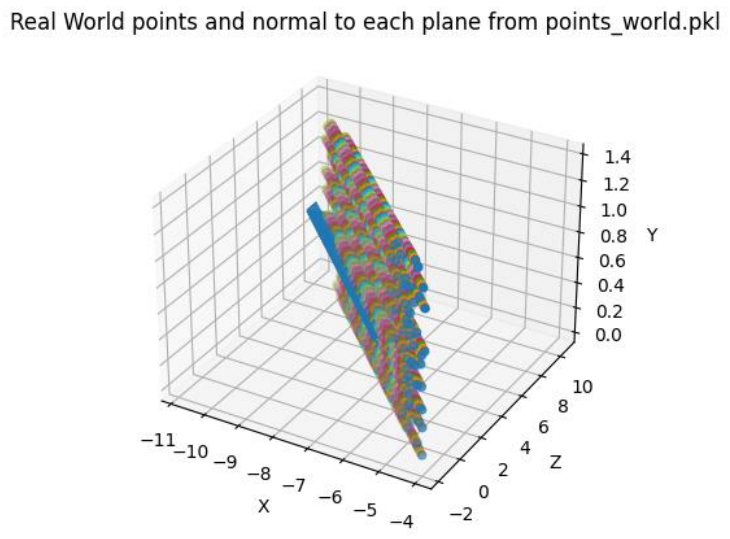


The blue point is the first frame real world C.G. from dataset, the orange point is the second frame estimated C.G., the green line is the movement (dx,dy,dz), and the red arrow is the heading direction.

2. I did the same process for the Target and Ego dataset, where both vehicles are driving at a fixed heading.

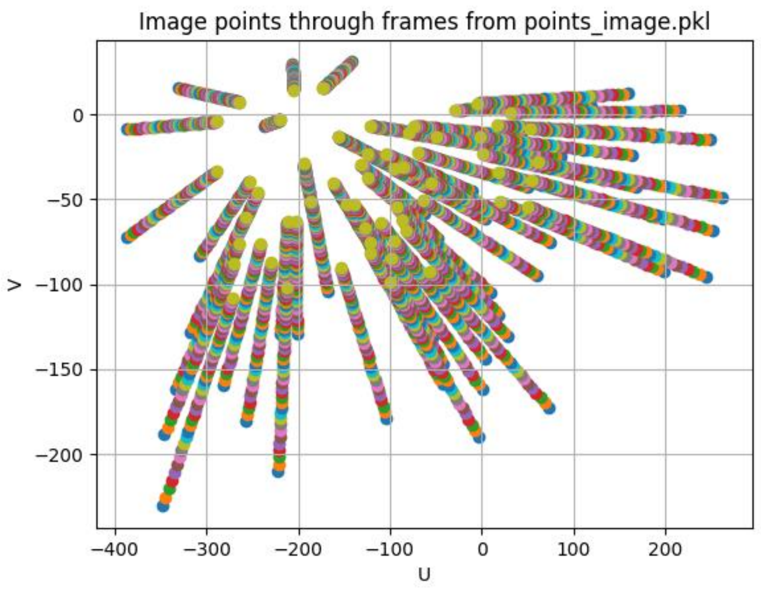
Pre-processing

3D plot of real-world points (at ego_0 coordinates) from all the frames and normal vector for each plane:

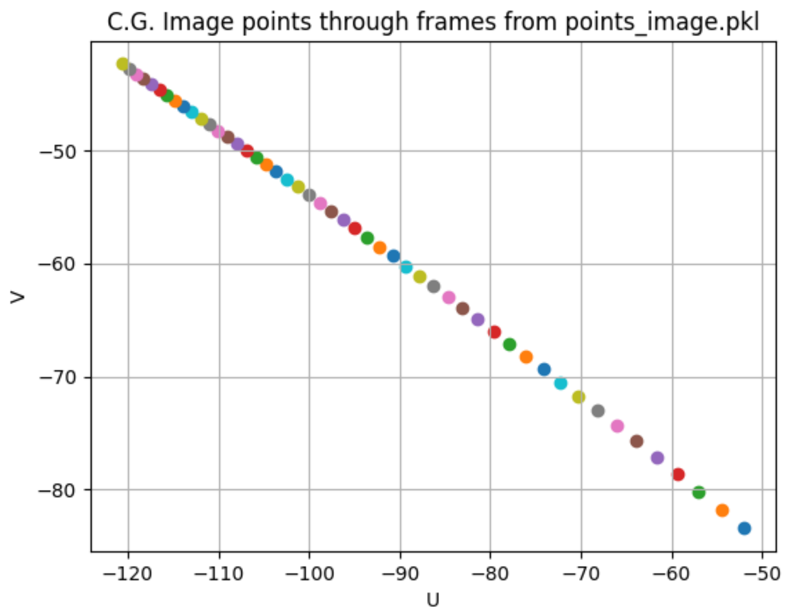


It is possible to see that the target vehicle is moving on X-Z plane only which is correlate to the Flat world assumption (dy = 0).

The target vehicle is become farther away from ego_0 and moving straight forward without changing the heading through frames.
After I understood the real world motion, I observed the motion in the image plane through frames:



First frame points are blue color and last frame points are ochre color.
It is possible to see that the distance between points is getting smaller through frames since the target vehicle is become farther away from ego.
To understand more the movement in the image plane I calculated C.G. point for every frame and plot it through frames:



It is possible to see that the C.G. point is moving up and left of image plane, and distance between each point is getting smaller.
The points are moving to the left of the image because the camera is directed to the left side of the vehicle but this time both vehicles are

driving at a fixed heading, the points are moving to the upper side of the image become farther away from ego, and the image catch more of the close road. The distance between points is getting smaller because the distance between target vehicle and ego vehicle is getting bigger.

Solution

Since I already used T_{ee_0} transformation matrix at each frame, the relative movement already considered in this process.

Results

After I run over all frames, I found out that the maximal errors are:

$$Max\ \Delta\theta = 0.0[rad], Max\ norm(dx, dy, dz) = 6.9 \cdot 10^{-15}$$

2) **Solution**

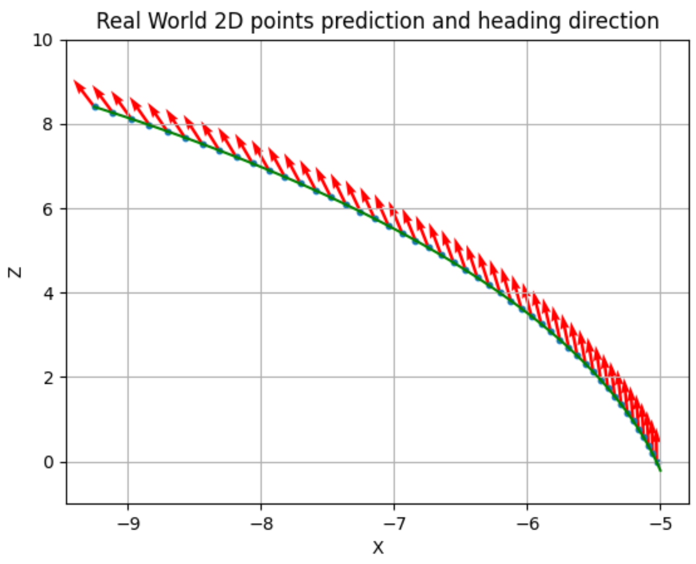
To extend the program to estimate motion over the whole datasets, using points_world.pkl every 5 frames, I created the MotionEstFull which run over all frames and estimate the motion between each two frames using the MotionEstTwoFrame function, when every 5 frames the first real world points that given are from points_world.pkl, and on the other frames I'm transferring the previous estimated points in real world. Every 5 frames I calculated the norm of the prediction error from the estimated real-world points to the real-world points from points_world.pkl, and since the errors are from order of $O(10^{-14})$ I left it that way, but if the prediction error was larger, I could use Kalman filter in order to prove the estimation.

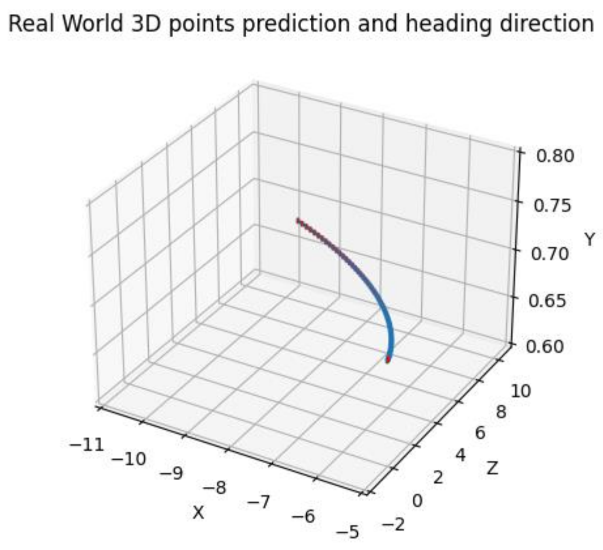
Results

1. After I run this function on the Target only dataset, I found out that the maximal errors are:

$$Max\ \Delta\theta = 9.2 \cdot 10^{-9}[rad], Max\ norm(dx, dy, dz) = 6.4 \cdot 10^{-15}$$

And the estimated real-world 3D and 2D route:



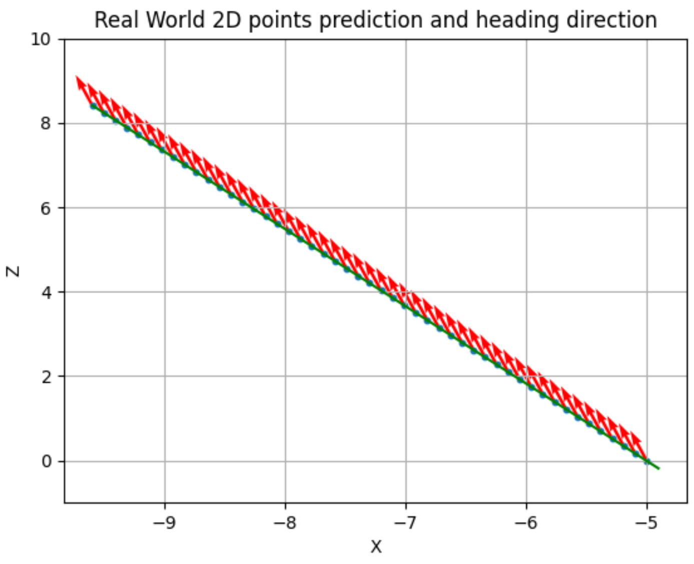


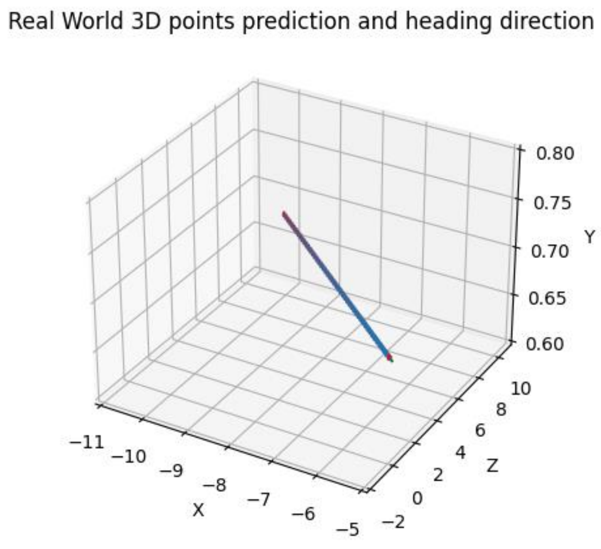
The blue points are the estimated C.G. real world points, the green lines are the movements (dx,dy,dz) and the red arrow is the heading direction.

2. After I run this function on the Target and Ego dataset, I found out that the maximal errors are:

$$Max\ \Delta\theta = 0.0[rad], Max\ norm(dx, dy, dz) = 6.4 \cdot 10^{-15}$$

And the estimated real-world 3D and 2D route:


Real World 2D points prediction and heading direction
Z
X


Real World 3D points prediction and heading direction
Y
Z
X