Robot Navigation Project 2 Report

ID N13492085

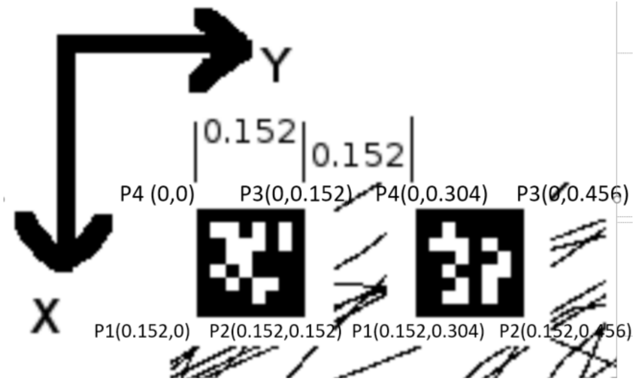
Jeremy Lu

Part 1 Pose Estimation

In this part we are estimate the pose of the robot including XYZ location and orientation. We are given all camera frame captured during its flight, for this part we are going to use following information:

- April tag's ID that are detected in each frame
- Location of four corners of the April tags
- Parameters including calibration camera matrix
- Layout of the April tag on the ground

First, we want to calculate the corner's position in world frame, given the layout, we can know the coordinate of corner of each April tag using distance and size of the tags.



We can code the coordinate of April tags in to a GETCORNER function in matlab, the function will take in the ID of the April tag and return corresponding 4 corners of that April tag which will be used as location in world frame.

The transformation between point in camera frame a world is:

$$p_c \sim H p_w$$

Here we know Pc and Pw is calculated using the getCorner function described above, we want to know H(a 3x3 matrix), we can write the above equation,

$$\begin{pmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i' \end{pmatrix} \underset{\uparrow}{\boxed{h}} = 0$$

We need all of our point into the above matrix, so our matrix will be a 2n * 9, and h is the transformation matrix (9*1), we solve this equation using svd,

USV' = A

And we take 9th column of V as h.

Then we want to decompose the h into rotation matrix and translation,

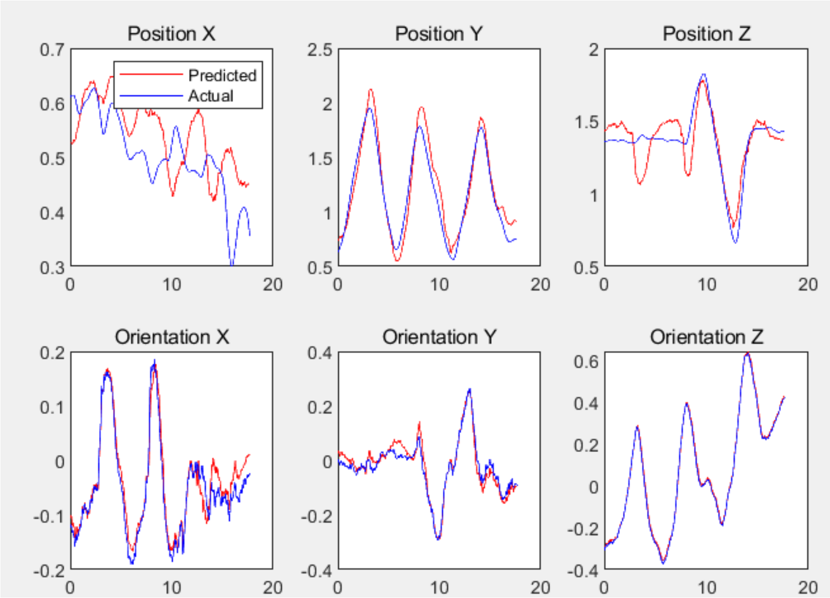
$$\begin{pmatrix} \hat{R}_1 & \hat{R}_2 & \hat{T} \end{pmatrix} = \begin{pmatrix} \hat{r}_{11} & \hat{r}_{12} & \hat{t}_1 \\ \hat{r}_{21} & \hat{r}_{22} & \hat{t}_2 \\ \hat{r}_{31} & \hat{r}_{32} & \hat{t}_3 \end{pmatrix} = \underbrace{\begin{pmatrix} f & 0 & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}}_{K^{-1}H}$$

H was just calculated, and K is given in parameter, the rotation matrix is given as below:

$$\begin{pmatrix} \hat{R}_1 & \hat{R}_2 & \hat{R}_1 \times \hat{R}_2 \end{pmatrix} = USV^T$$

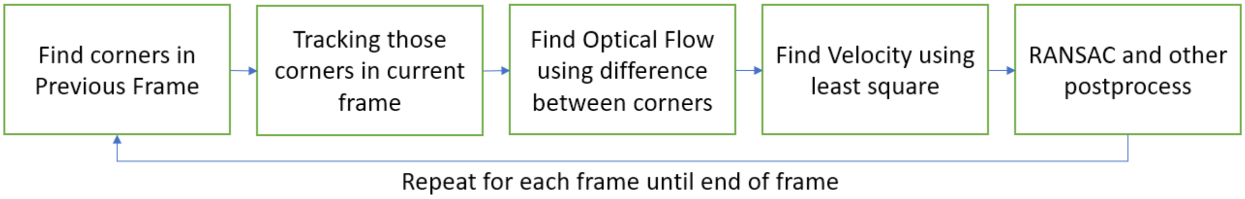
$$R = U \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(UV^T) \end{pmatrix} V^T$$

The translation needs to be normalized so wen divide T_hat by norm of R1_hat. And we want to run through all of the frames to get pos at each moment, here is the result:



We can see that prediction is like real data except position X has relatively large drift

PART 2.1 Velocity Estimation


Find corners in Previous Frame
Tracking those corners in current frame
Find Optical Flow using difference between corners
Find Velocity using least square
RANSAC and other postprocess
Repeat for each frame until end of frame

Above is the general process, we use matlab built in function to find corners in a frame and using point tracker to see where those points go.

After we get points locations and corresponding velocity, we want to know given position and velocity what linear and angular velocity will cause such position and optical flow, we have relation:

$$\dot{\mathbf{p}} = \frac{1}{Z}A(\mathbf{p})\mathbf{V} + B(\mathbf{p})\boldsymbol{\Omega} = \left(\tfrac{1}{Z}A(\mathbf{p}) \quad B(\mathbf{p})\right)\begin{pmatrix}\mathbf{V}\\ \boldsymbol{\Omega}\end{pmatrix}$$

It is unlikely that we will find V and Omega that perfectly match the above equation, we will use least square method to have a best estimate,

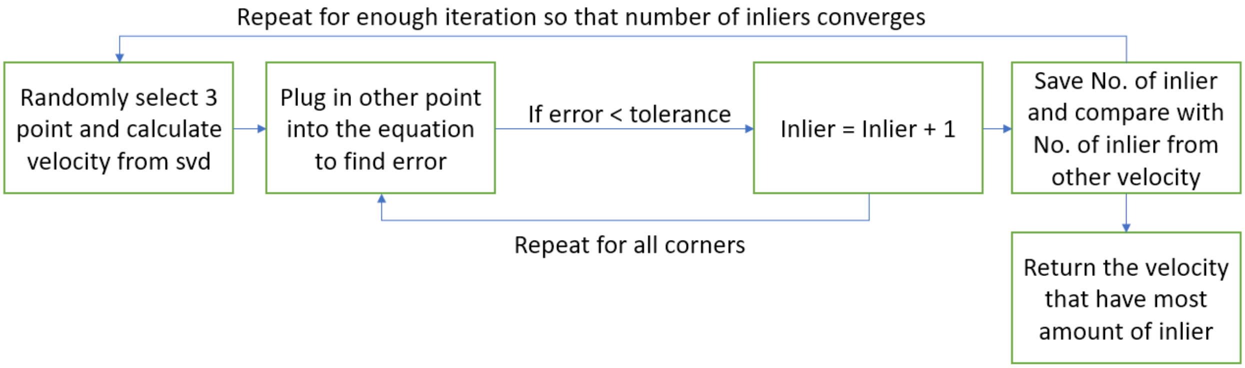
$$\mathbf{V}^*, \boldsymbol{\Omega}^* = \arg\min_{\mathbf{V},\boldsymbol{\Omega}} \sum_{i=1}^{n} \left\| \left(\tfrac{1}{Z_i}A(\mathbf{p}_i) \quad B(\mathbf{p}_i)\right)\begin{pmatrix}\mathbf{V}\\ \boldsymbol{\Omega}\end{pmatrix} - \dot{\mathbf{p}}_i \right\|^2$$

Note that (1/Z * A, B) is a 2 by 6 matrix, in order to have a solution, we need three matrix thus three point needed to get a result, here we randomly select three points and compute the result using svd in matlab.

PART 2.2 RANSAC

We know that there will be bad point in data because of noise and uncertainty. If we choose those bad point to estimate velocity the result will be not useful. Thus, we use RANSAC to remove those outlier point in data.

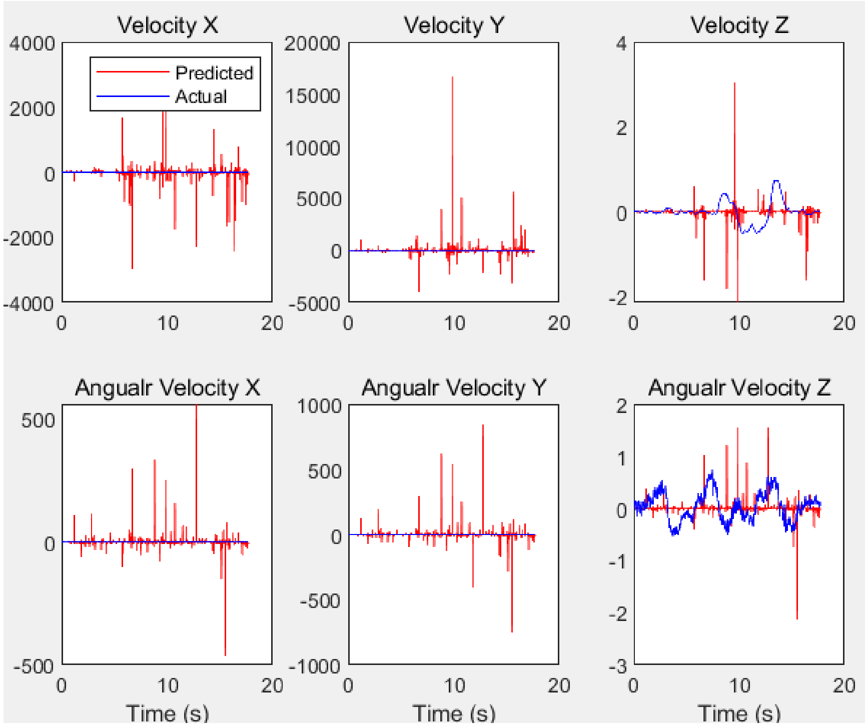
From 2.2, we calculated the velocity, we now use all other points to test if our estimated velocity accurately describes the relation between Point position and Point velocity. Of course, it is unlikely the least square becomes zero (which means no error at all), we define a tolerance e to forgive reasonable deviation, the algorithm is described as below:



Repeat for enough iteration so that number of inliers converges
Randomly select 3 point and calculate velocity from svd
Plug in other point into the equation to find error
If error < tolerance
Inlier = Inlier + 1
Save No. of inlier and compare with No. of inlier from other velocity
Repeat for all corners
Return the velocity that have most amount of inlier

Finally we plot the result from matlab:

Note that the result is not correct, I think the reason is due to the error is not correctly defined


Velocity X
Predicted
Actual
Velocity Y
Velocity Z
Angualr Velocity X
Time (s)
Angualr Velocity Y
Time (s)
Angualr Velocity Z
Time (s)