Robot Localization and Navigation:Project 2

Abhimanyu Suthar (abs9477) N10804526

April 7, 2024

1 Introduction

This report intends to serve as a self-contained description of the project implementation and the results in 2 phases. In the first phase, a pose estimator that estimates position and orientation [3] of the quadrotor by making use of April Tags[2] is implemented.
The second phase involves detecting features in the image and then finding the linear and angular velocity of the robot using mathematical models that involve image gradients.

2 Background

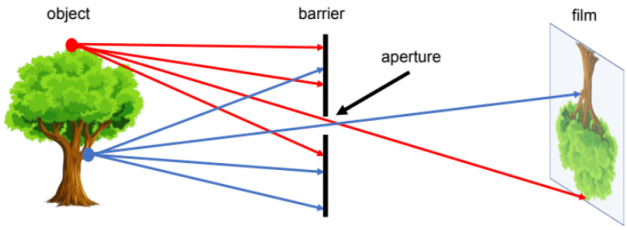

Figure 1: pinhole model

The pinhole camera model offers a conceptually elegant and computationally inexpensive framework for investigating the fundamental principles of image formation. This model serves as a critical foundation for understanding more complex camera systems and their interaction with the environment.
In order to make use we move that image to the front diagrammatically such that Xw with w subscript represents point in world frame. x,y represents points in image frame whereas XC represents points in camera frame.

$$\lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = K \begin{pmatrix} {}^{c}R_{w} & {}^{c}T_{w} \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

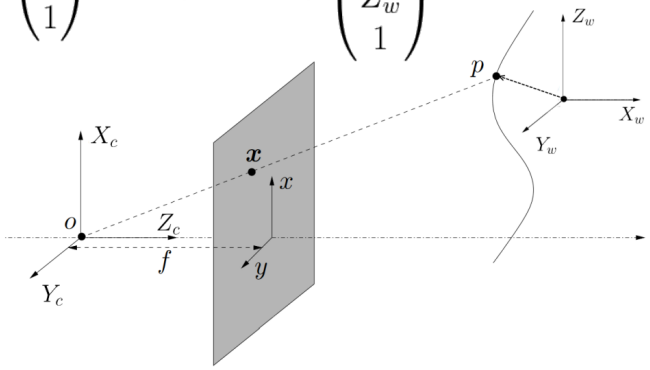

Figure 2: Projection equation

2.1 April Tags

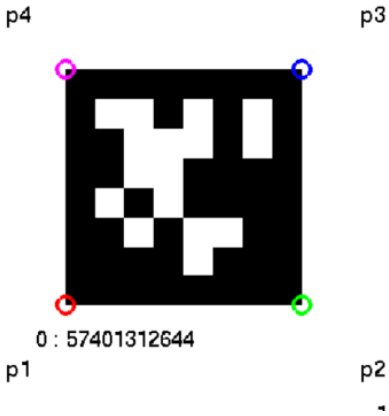

Figure 3: Corners of the April Tag

[2] April Tags are a robust and flexible visual fiducial system that are designed to be easy to recognize and distinguish from one another. It allows to use a projective transform to find the relation between the coordinates of the tag in the world frame and in the image frame(pixels). This transformation matrix can then be utilized to find the robot's pose i.e position and orientation in the world frame. Although computer vision is currently focused on finding patterns from natural surroundings, artificial markers like April Tags can play a role in localization by strategically placing them in controlled environments

3 Phase-1: Vision based Pose Estimation

This phase involved using data collected from the Nano quadrotor that was either held by hand or flown through a prescribed trajectory over a mat of April Tags, each of which has a unique ID that is stored in text file. The calibration

matrix which has the structure

$$K = \begin{pmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{pmatrix}$$

and the transformation between the camera and robot center is in the parameters.txt file as well.

3.1 MATLAB

The data for each trial is provided in a mat file that contains a struct array of image data called *data*, The format of the image data struct is as follows:
1. time stamp(t) in seconds
2. ID of every AprilTag that is observed in the iamge (id)
3. The center and four corners of each ID are stored by notation p0(center), p1(bottom left), p2(bottom right), p3(top right), p4(top left)
4. Rectified image (img)

The mat file also contains Vicon data taken at 100 Hz, which will serve as our ground truth measurements. The vicon data is stored in two matrix variables, time and vicon.
The vicon data contains the Vicon data in the following format:

$$[x \quad y \quad z \quad roll \quad pitch \quad yaw \quad v_x \quad v_y \quad v_z \quad w_x \quad w_y \quad w_z]^T$$

3.2 MATLAB functions

The functions that were used are getCorner.m, init.m plotData.m and estimatePose.m

getCorner.m: The input to this function is the list of all AprilTag ids detected in the current image. The 9 columns of the tag ids with special spacing between 3-4th and 6-7th columns are taken care of, using different "if statements" for each column based on the positioning of the ids. The final output is then stored in a column vector **res**

estimatePose.m: This function calls the getCorner function to get the corners in world frame. Then it calculates the A matrix

$$A = \begin{pmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x'_i x_i & -x'_i y_i & -x'_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -y'_i x_i & -y'_i y_i & -y'_i \end{pmatrix}$$

Here, x_i and and y_i are points in the world frame where as $(x_i)'$ and $(y_i)'$ are in the image coordinates (pixels).
Here i denotes the point/feature selected, each tag gives us 5 points and therefore the A matrix becomes a $10x8$ matrix for one tag. Substituting the values of all tag ids by stacking them on top of another creates an $nx8$ matrix where is the total number of features, in this case, 12 ids in every time-step with 5 points each leading to $n = 60$. After which SVD is performed on the matrix to get the homography matrix h in row form. [5]. After which it is scaled by multiplying by $sign(v(9,9)$.

This **h** matrix contains the parameters of the projective transform, representing the transformation of the planar object in the image.
Now the next step is to calculate the pose i.e. the rotation and translation from this matrix.

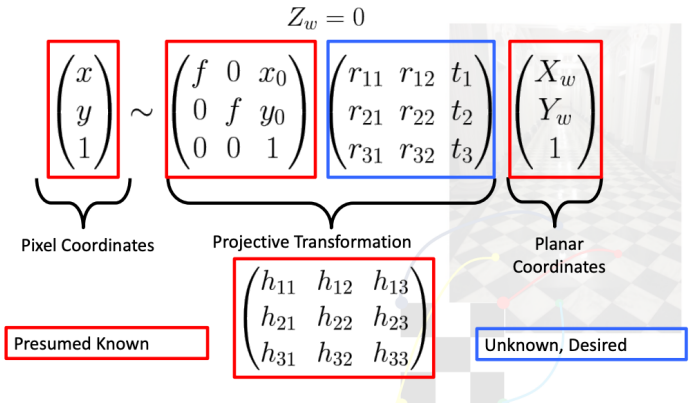

Figure 4: Relation of H to R, t and intrinsic matrix

This involves taking the inverse of the camera intrinsic matrix (k) and multiplying it with h to get:

$$\begin{pmatrix} \hat{R1} & \hat{R2} & \hat{T} \end{pmatrix}$$

We need to satisfy the constraints:

$$R_1^T R_2 = 0$$

$$||R_1|| = ||R_2|| = 1$$

Singular Value Decomposition is taken on

$$\begin{pmatrix} R_1^T & R_2^T & R_1 x R_2 \end{pmatrix} = USV^T$$

We get

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

$$T = \hat{T}/||R_1^T||$$

The R and T matrices combined gives the transformation from **world to camera frame**

Transformation matrix for camera to robot frame is :

$$R_c^r = \begin{pmatrix} 0.707 & -0.707 & 0 & -0.04 \\ -0.707 & -0.707 & 0 & 0 \\ 0 & 0 & -1 & -0.03 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Since the final pose needs to be in the world frame:

$$pose - world = inverse(R_c^r * R_w^c)$$

The other functions in this phase are **plotData.m**, **init.m**
init.m acts as an initializer to import the required Dataset which is used in the main file.
plotData.m plots the ground truth VICON data with the estimates.

3.3 Plots- Model 1

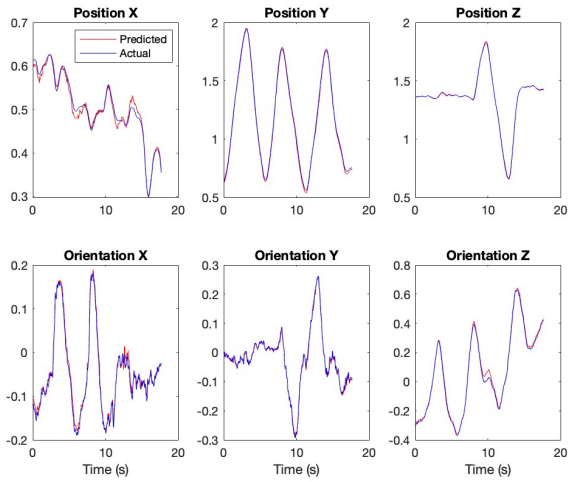

Figure 5: Dataset-1: pose estimates

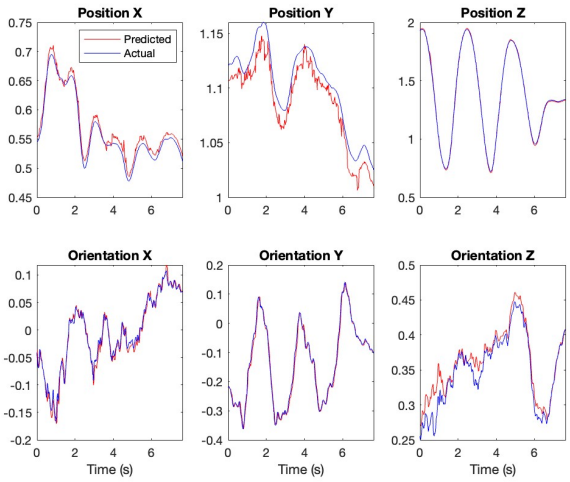

Figure 6: Dataset-4: pose estimates

Observations: The pose estimates follow the ground truth very closely which means the corners points selected are accurate, and give good results.

4 Phase-2

4.1 Corner Extraction and Tracking

In this phase, a corner/feature detector is implemented in MATLAB to find trackable features to compute the sparse optical flow between two consecutive images .
Sparse optical flow gives the flow vectors for the selected features in the image frame whereas dense optical flow computes the flow vectors for all pixels present in the image. [1]

The goal is to find correspondence between two consecutive images

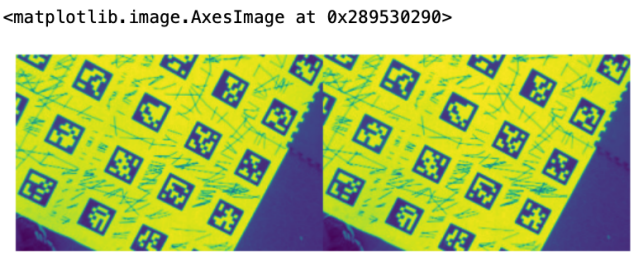

Figure 7: Consecutive images

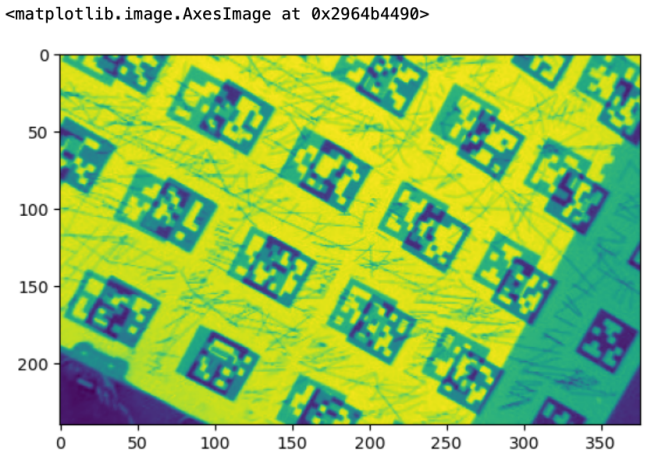

Figure 8: Visualization of change (not between consecutive images)

We can see the difference in pixels visually when places on top of one another in figure-3. **Note**: the above image does not represent overlap between consecutive frames as they are almost indistinguishable. The figure serves as a visual reference for understanding optical flow.
The idea to select features or corners in this case that can be tracked between frames to estimate the flow vector

4.2 MATLAB functions

OpticalFlow.m The MATLAB function detectMinEigenFeatures is used to get the strongest 250 keypoints in the current image. After which the tracker is initialized to find the location of points in the next image, the optical flow is calculated by dividing the change in position by the change in time and stored in a column vector. The next step here is to calculate the Height based on the estimates and considering the nonzero rotation angles. The next step involves calculating the A(p) and B(p) matrices:

$$\dot{\mathbf{p}} = \frac{1}{Z}A(\mathbf{p})\mathbf{V} + B(\mathbf{p})\mathbf{\Omega} = \begin{pmatrix}\frac{1}{Z}A(\mathbf{p}) & B(\mathbf{p})\end{pmatrix}\begin{pmatrix}\mathbf{V}\\ \mathbf{\Omega}\end{pmatrix}$$

Figure 9: If depth is known

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

Figure 10: Least squares problem

Therefore,

$$\begin{pmatrix}V^*\\ \Omega^*\end{pmatrix} = pinv(H)\dot{p}$$

velocityRANSAC.m [4] The working of this function is as follows: First, a minimal sample set (3) is chosen and for this randomly chosen sample set, then H matrix is calculated.
Based on the least squares solution involving psuedoinverse of H matrix, the linear and angular velocity of the camera expressed in the camera frame is computed.

Keep this matrix which we will call as 'VW' aside, now for all the other detected features find the H matrix one by one, get their optical flow velocity, compute the norm squares of the difference between the current H matrix (for selected feature) multiplied by VW (the linear and angular velocity matrix which we stored earlier) and subtract it from current optical flow velocity.
If this value is below a certain threshold, in this case, 0.005, then this point is an inlier.

$$||(1/Zi)A(p_i)B(p_i)(VW) - \dot{p}_i||^2 \leq \beta$$

Repeat this for all the points that are detected features and find out the total number of inliers for this randomly selected minimal set of 3 points.
After running this for one sample set, select another one, repeat the above process until K iterations, finding K using

$$K = log(1 - p_success)/log(1 + e^M)$$

where M is number of points (in this case:3), e is the probability that a point is an inlier (taken as 5) Update the maximum inliers variable if the number of inliers in current iteration is greater, otherwise continue until K iterations.

The other functions **plotData.m**, **init.m** ,**getCorners.m** and **estimatePose.m** have the same implementation as in the first phase.

4.3 Plots- Model-2

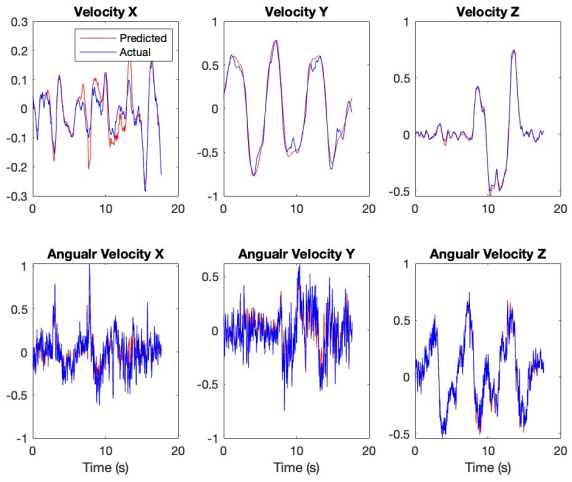

Figure 11: Dataset-1-without-RANSAC

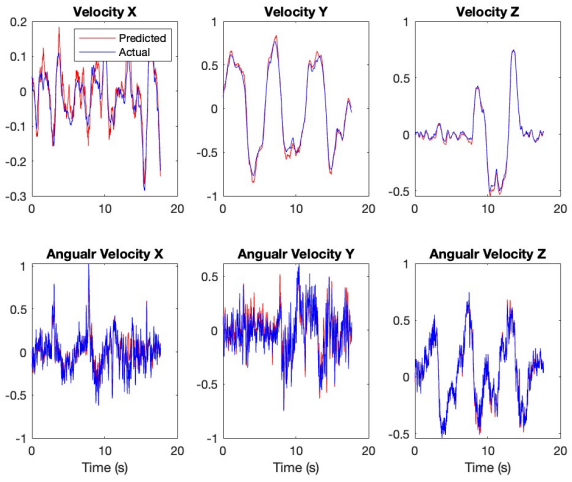

Figure 12: Dataset-1-with-RANSAC

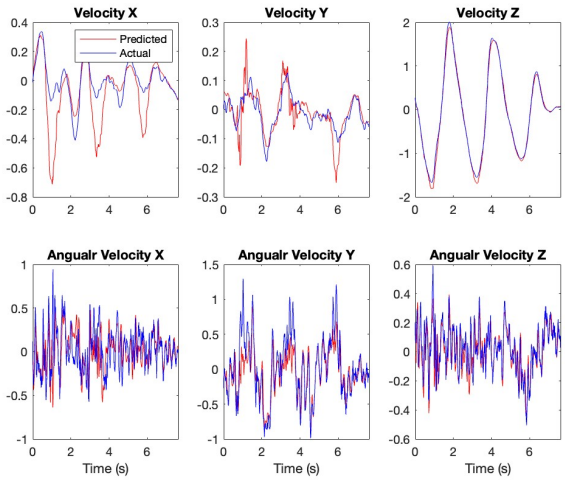

Figure 13: Dataset-4-without-RANSAC

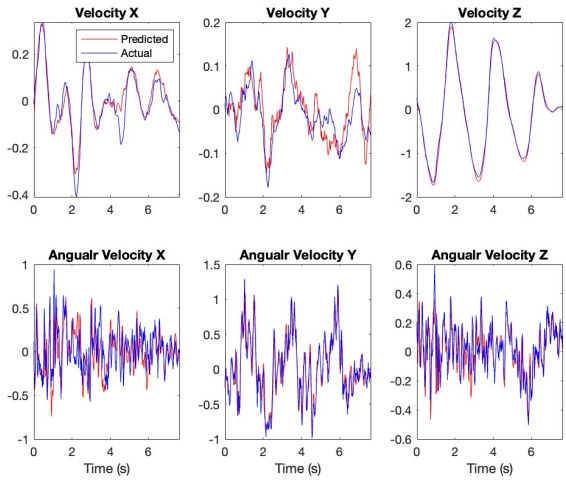

Figure 14: Dataset-4-with-RANSAC

Observations: For the first dataset, figure:11 and figure:12 the features detected were not very noisy and therefore there is not much of a difference between RANSAC and non-RANSAC plots.
However, for the dataset number 4, there are some visible differences between figure:13 and figure:14 which means there is some noise in features set and RANSAC is able to filter that.

5 Conclusion/Future Work

The mathematical models presented in this report demonstrate that April Tags can be used to find the pose and velocities reliably in this case.

Future Work:

1. Test my skills by solving the calibration challenge set by a self driving company `https://github.com/commaai/calib_challenge`

2. Write my own open-source feature/corner detector.

Valuable Reference: Professor Giuseppe Loianno's notes.

References

[1] Y. Ma, S. Soatto, J. Kosecka, and S. Shankar Sastry, "Chapter 4," in *An Invitation to 3D Vision*, Springer, 2004.

[2] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2011, pp. 3400–3407.

[3] K. Hata and S. Savarese, "Cs231a course notes 1: Camera models," 2017. [Online]. Available: `https://api.semanticscholar.org/CorpusID:41993084`.

[4] R. Szeliski, *Computer Vision: Algorithms and Applications*, 2nd. Seattle, WA: University of Washington, 2022. [Online]. Available: `https://szeliski.org/Book/`.

[5] Carnegie Mellon University, *Svd for total least squares*, PowerPoint slides, Retrieved from URL, Year. [Online]. Available: `https://www.cs.cmu.edu/~16385/s17/Slides/11.5_SVD.pdf`.