NYU TANDON SCHOOL OF ENGINEERING

ROBOT LOCALIZATION AN NAVIGATION

Project – 2

Name: Rallapalli Dinesh Sai, Naidu
ID: N19602446
NetID: dr3696

Masters in Mechatronics and Robotics Engineering, Department of Mechanical and Aerospace Engineering, Tandon School of Engineering

ROB-GY_6213_1_A **Spring'24 (04/7/2024)**

Instructor: Prof. Giuseppe Loianno

Rallapalli Dinesh Sai, Naidu

N19602446
NetID: Dr3696

1. Overview and Process

Overview

Project 2 in the ROB-6213 course focuses on developing a vision-based 3-D pose estimator for a Nano+ quadrotor using AprilTags. The project entails setting up the environment by calculating the world frame coordinates of AprilTag corners, followed by camera calibration and pose estimation using provided data and matrices. Additionally, corner extraction and tracking methods, including optical flow computation and RANSAC-based outlier rejection, are implemented to ensure accurate and robust pose estimation. Through this project, students gain practical experience in developing vision-based localization systems for robotics, with a specific focus on utilizing AprilTags for accurate pose estimation in a controlled environment.

Process

1. Vision-Based Pose Estimation

Environment Setup: Use data collected from a Nano+ quadrotor, flying over a mat of AprilTags arranged in a 12x9 grid. Calculate the location of every tag corner in the world frame based on the provided dimensions and arrangements.

Calibration: Utilize the intrinsic camera calibration matrix and the transformation between the camera and the robot center (provided in parameters.txt) to find the homography. Adjust the homography matrix to ensure positive Z values and transform the camera-based pose estimate to the robot frame.

PoseEstimation: Analyze the data in a provided .mat file, which includes timestamps, AprilTag IDs, and positions in image coordinates. Use the camera calibration data, tag corners, and known world frame locations to compute the Nano+'s measured pose for each data packet.

2. Corner Extraction and Tracking

Corner Extraction: Detect corners in each image, which may include both AprilTag corners and additional detected corners. Utilize built-in MATLAB functions, external libraries, or your own implementation.

Optical Flow and Velocity Estimation: Compute the sparse optical flow between consecutive images using a KLT tracker to get velocity estimates in the calibrated image frame. Apply a low-pass filter on time deltas for improved results.

RANSAC-Based Outlier Rejection: Implement RANSAC to reject outliers in optical flow computation, requiring three sets of constraints to solve the velocity estimation system.

2. Process

GetCornerFuntion

The function aims to find the coordinates of all the AprilTags present on the mat in the world frame based on given dimensions. It takes a list of IDs captured by the camera in one time interval as input and outputs a list of the coordinates of the four corners for each particular ID. Initially, a zero matrix is created with eight rows (representing the x and y coordinates of all four corners) and a number of columns equal to the number of IDs present in the list. This matrix will be populated with the coordinates of the tag corners as the function proceeds.

After initializing the zero matrix, the column number for each ID is calculated. This formula likely calculates the column number based on the given ID and the assumption of a 12x9 grid layout for the tags. Subsequently, the x-coordinate for each corner is calculated, presumably using the assumption of uniform distances between tags. However, for the y-coordinates, the column number is taken into account, indicating that the distances between columns are not uniform. Finally, the x and y coordinates for all corners are stacked in a matrix, likely forming a representation of tag corner coordinates in the world frame.

estimatePoseFunction

The function aims to estimate the pose of the quadrotor based on images captured of AprilTags and then plots this data alongside Vicon data for comparison. Initially, the camera calibration matrix is extracted from the parameter.txt file. Then, a list of all the IDs captured by the camera at a specific timestamp is obtained. Subsequently, a null matrix A is created to hold the coordinates of all corners of all tags in the camera image. The matrix A is structured such that each column represents the coordinates of the corners of a single tag, stacked vertically.

The homographic transformation matrix h is a column vector containing vertically stacked elements of the holography matrix, enabling the transformation from the camera frame to the world frame.

To find the value of the homographic transformation matrix h, we calculate the Singular Value Decomposition (SVD) of the matrix A. Given that matrix A has dimensions 8*no. of ids and the no. of columns would be 9. Hence, we'll use the inbuilt MATLAB function for calculating the SVD of the vector A i.e. [S,U,V] = svd(A). Hence, h3x3 = V(9) i.e. all the elements in V make up a 3x3 h matrix.

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

Vector of unknown transformation parameters

Need 4 of these to get all 8 degrees of freedom

The elements are stored in row order

$$Ah = 0$$

Solve using SVD with the smallest singular value

$$A = USV^T \implies h = \boxed{V_9}$$

9th Column of V

Furthermore, we take the h matrix and multiply it with the inverse of the camera calibration matrix K, to give Now, to calculate the Rotation matrix we take the SVD of the matrix on the LSH in the equation.

Single Value Decomposition

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

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

To

$$T = \hat{T} / \left\| \hat{R}_1 \right\|$$

To transform the values from the world frame to the body frame, we require the transformation matrix specified in the parameters.txt file. This matrix represents the homogeneous transformation from the world frame to the body frame. From this transformation matrix, we extract the rotation matrix, which encapsulates the orientation information. Using this rotation matrix, we calculate the Euler angles in ZYX orientation, representing the orientation of the quadrotor in terms of roll, pitch, and yaw angles. Additionally, we extract the position vector from the transformation matrix, providing the location of the quadrotor in the body frame. These steps enable us to obtain the pose of the quadrotor in its body frame, facilitating further analysis and comparison with the observed AprilTags data.

Velocity Estimation

We create a row vector containing all the timestamps from the given data and pass it through a low-pass filter. This step helps smooth out any noise or irregularities in the timestamps. Subsequently, we initialize a loop to load the captured images, creating variables for the previous and current images, and calculating the difference in the timestamps for the two images. We then detect good points which likely identifies robust and distinctive features in the image. Additionally, we initialize a tracker to the last frame using the Computer Vision toolbox in MATLAB. Next, we find the locations of the points from the current image and determine the location of the next point.

To calculate the depth (Z) of the points, we normalize the values of good points and points by multiplying them by the inverse of the camera calibration matrix (K). Finally, we calculate the velocity in the camera frame by taking the x and y coordinates of the

corners of the images and dividing them by the difference between the timestamps, thus obtaining the left-hand side of an equation.

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

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{Z} \begin{pmatrix} -1 & 0 & x \\ 0 & -1 & y \end{pmatrix} \begin{pmatrix} V_x \\ V_y \\ V_z \end{pmatrix} + \begin{pmatrix} xy & -(1+x^2) & y \\ 1+y^2 & -xy & -x \end{pmatrix} \begin{pmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{pmatrix}$$

And, hence further, we have (u,v), (x,y) and Z and we need to find V and Omega. We find x and y by using the normalized values. We put this in a loop to calculate the Ap and Bp matrices. When stacked together make up the H matrix. Further we multiply the pseudo-inverse of the H matrix to the velocity, and the transformation matrix, we get the estimated velocity.

RANSAC

Random Sample Consensus is a robust statistical algorithm used in computer vision and other fields to estimate model parameters from a set of observed data points that may contain outliers. It works by iteratively selecting random subsets of data points, fitting a model to each subset, and then determining a consensus set of inliers that best fits the model. RANSAC is effective in scenarios with noisy data as it can robustly estimate model parameters even in the presence of outliers. It is widely used in various computer vision applications, such as image stitching, object recognition, and structure from motion.

RANSAC: Random Sample Consensus

The algorithm (in pseudocode):

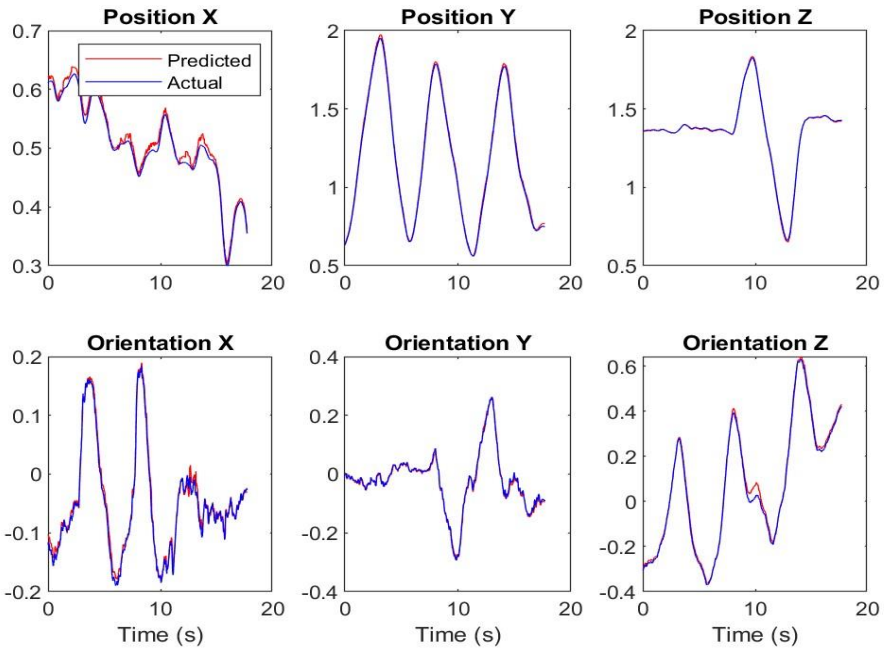
```
Repeat for k iterations
1.  Choose a minimal sample set
2.  Count the inliers for this set
3.  Keep maximum, if it exceeds a desired number of inliers
stop.
```

The velocityRANSAC function accepts input as the velocity that is in the camera image frame, along with the normalized pts, Z, and e i.e., the probability of one point being an inlier. So, we start by determining the probability of success of at least hitting one inlier as 0.99. Then we calculate the no. of iterations k

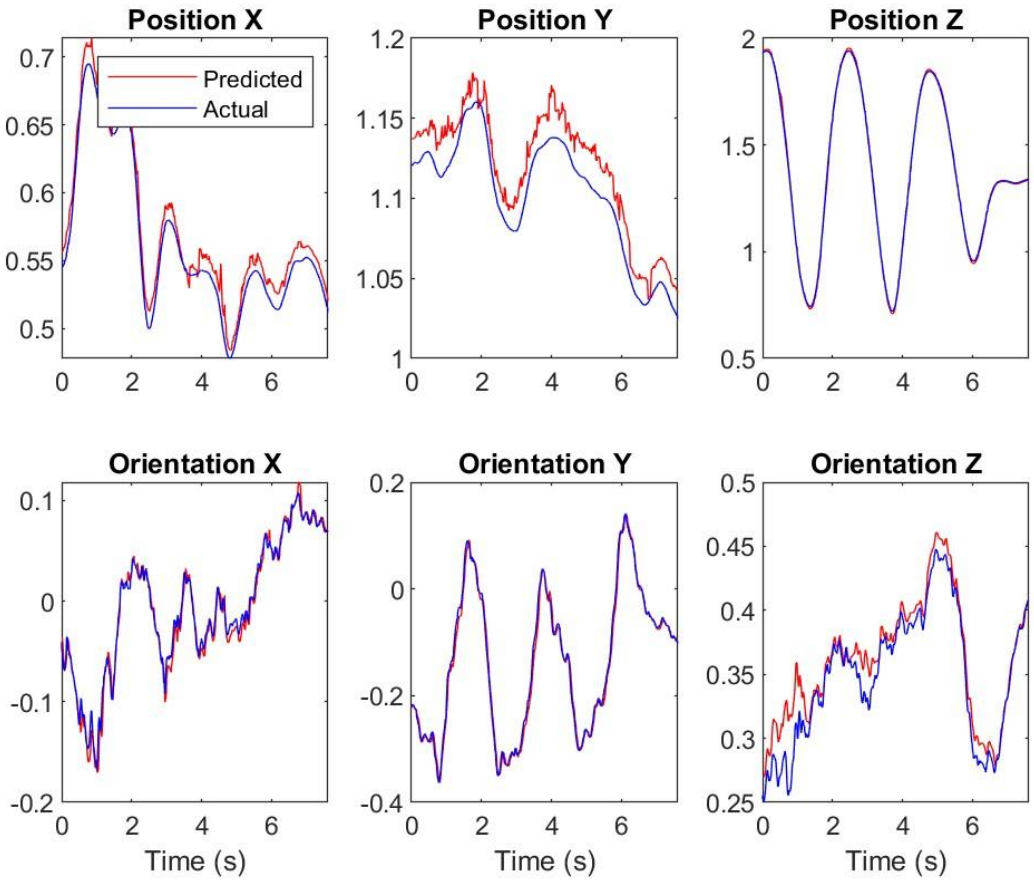
$$k = \frac{log(1 - p_{success})}{log(1 - \epsilon^M)}$$

3. Plots

PoseEstimation-Dataset-1



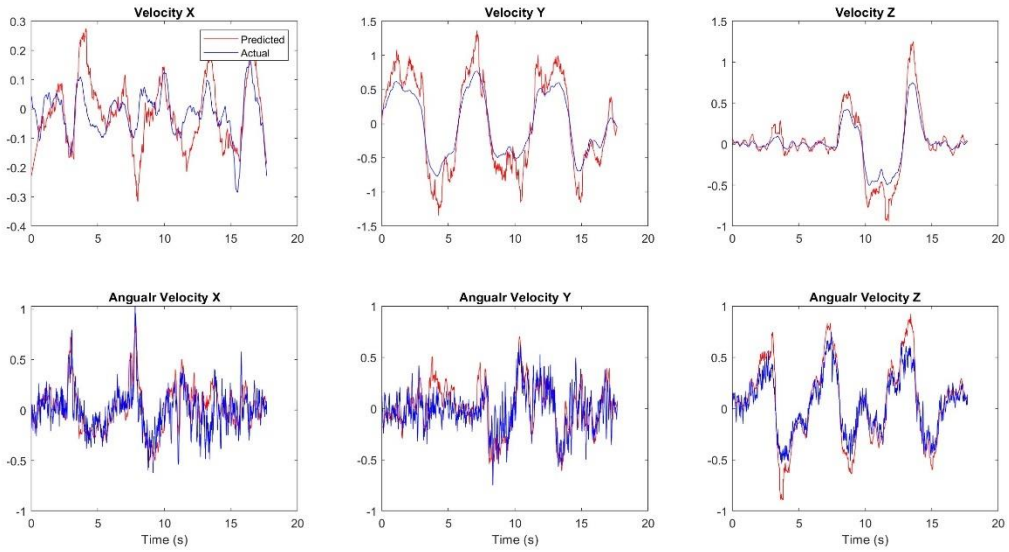
PoseEstimation-Dataset-4



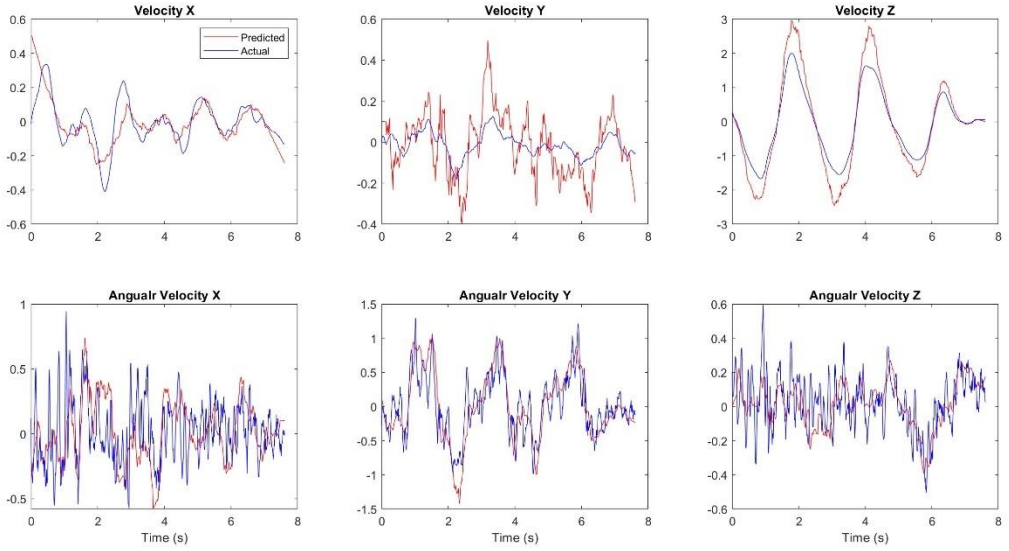
Note:

RANSAC 0: Not using RANSAC
RANSAC 1: Using RANSAC (In this case, Demonstrated using 1, other than 0 all positive integer values uses RANSAC)

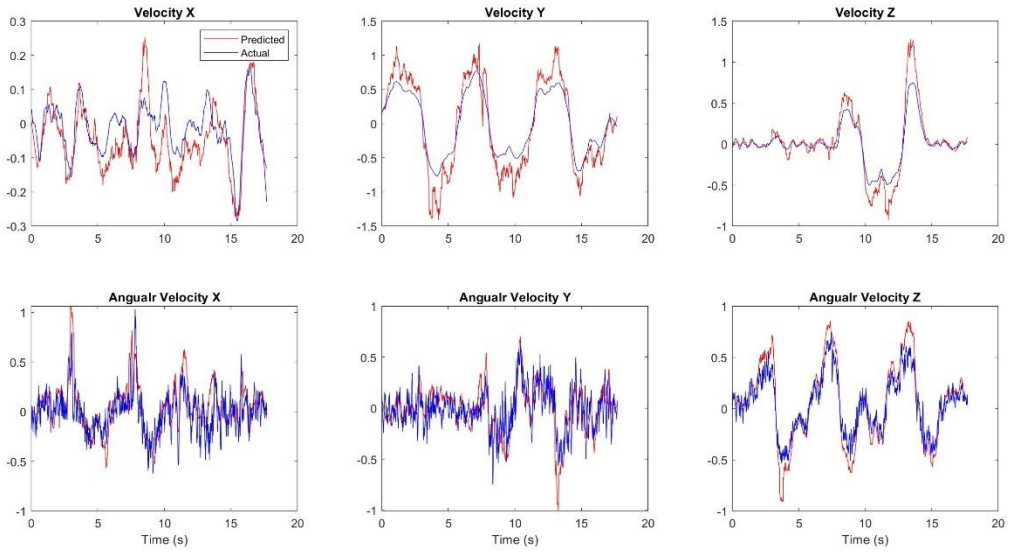
Opticalflow(RANSAC-0,DATASET-1)



Opticalflow(RANSAC-0,DATASET-4)



Opticalflow(RANSAC-1,DATASET-1)



Opticalflow(RANSAC-1,DATASET-4)