Vision Based Tracking and Estimation

Alexander Garcia

amg9558

Tandon School of Engineering
04/12/2022

NYU

1. Pose Estimation

The first step in the development of the pose estimation was writing a function to return the coordinates in the world frame of the April Tags detected by the camera in each frame. The function takes the id numbers as an input and returns the coordinates of the corners. The data is returned such that each column has the values for a given id and each row is x and y of the 4 corners, resulting in a matrix of 8 by the number of ids.

To get the values, the dimensions of the tags, spaces between the tags, and anomalous spaces are stored as constants. Using the id numbers, the row and column numbers are calculated using the mod function, returning the remainder. This allows for systematically returning the x and y values. The vector of row values is multiplied by twice the dimension of the tag plus the width of a tag to get the lower left x. From here all other x values for that tag are calculated.

In calculating the y values, the first step is to handle any necessary adjustment to the y due to the 2 spaces that are slightly longer according to the handout between columns 3 and 4 and then columns 6 and 7. The row and column vectors start at index zero, hence the if statement using greater than 5 and greater than 2 to determine when to make the adjustment. Again, after the first y for an id has been determined, the other three corners are determined from that one. The matrix of tag corner coordinates is then returned.

The coordinates of the tags in the world frame and the data from the camera are passed to the function used to calculate the H matrix. This function follows the steps outlined in the algorithm at the end of the lecture on projective geometry titled “The four-point algorithm for a planar scene.” In this algorithm, an H matrix is developed using all the corner points available from the camera data other than the center point. The function loops over the corner data, creating skew symmetric matrices for each point in the camera frame. A matrix x is developed by taking the Kronecker Tensor Product of the coordinates of the corners in the world frame and the skew symmetric matrices of the points in the camera frame. As the for loop concatenates from a matrix with a column of zeros, this first column must be removed by selecting columns two to the end. Finally, the transpose is taken and the singular value decomposition computed. This provides the HL matrix by

taking the ninth column of Vx and unstacking it. H needs to then be normalized by performing another SVD and dividing the HL matrix by the second largest singular value. Lastly, the sign of H must be corrected by checking the sign of the transpose of a point in the camera frame multiplied by H multiplied by the coordinates of that point in the world frame. If the result is negative, the sign of H is swapped.

After H has been determined, an estimate of the rotation and position of the camera can be made by pre-multiplying by the inverse of the camera calibration matrix K that has been provided. The first two columns of the rotation matrix are extracted and the cross product taken to get the third, as it must be orthogonal to the first two. Another SVD is performed on the three estimations of the rotation columns and a final rotation matrix computed. The translation vector is normalized by dividing the estimated translation by the magnitude of the first column of the rotation matrix. This rotation and translation describe the transformation from the world frame to the camera frame. Taking the inverse of this gives a transformation from the camera frame to the world frame.

Next a transformation needs to be developed to go from the camera frame to the robot (IMU) frame. This is done with a rotation by rotating the x axis of the camera frame by 180 degrees, as the Z of the camera is pointed down, and rotating about the Z by -pi/4. The translation is provided so a transformation from the robot to camera can be developed. Combining these two transformations gives the transformation from the robot frame to the world frame. This is finally returned as the last column gives the position of the robot and the rotation used with the rotm2eul function in order to give the orientation in ZYX euler angles.

2. Corner Extraction and Tracking

Part 2 starts with reinitializing some necessary values such as the camera calibration matrix, and transformations from robot to camera frame. Additionally, a low pass filter is applied to the change in time between each sample of the camera data as recommended in the project handout.

The loop runs over every sample of the camera data. It starts by extracting the current image frame as well as the previous image frame. Using the previous

image, corners are detected for the point tracker using the function detectHarrisFeatures. This function was selected after reviewing the toolboxes provided in the handout, in particular the example for facial recognition and point tracking. The point tracker object is then initialized by passing the tracker the corner coordinates and the image. Corners from the current image are then generated by passing the current image to the tracker. The points are confirmed to be valid as both sets of corner locations are filtered against the validity vector returned by the tracker.

Before the next steps are executed the estimatePose function developed in part 1 is called as the Z value is needed to calculate the H matrix.

The values returned by the tracker are transformed from pixel values to coordinates in the camera frame using values from the camera calibration by:

$$X_c = (X - X_0)/f$$
$$Y_c = (Y - Y_0)/f$$

The optical flow is then calculated by:

$$u = (X_{curr} - X_{prev})/dt$$
$$v = (Y_{curr} - Y_{prev})/dt$$

These vectors are woven together so as to alternate u and v. A skew symmetric matrix is developed for the current position and the A and B matrices calculated by:

$$A = pe_3^T - I$$
$$B = (I - pe_3^T)[p] \times$$
$$e_3 = [0\ 0\ 1]^T$$

These matrices, with A divided by Zc, are concatenated into an H matrix which is then used to calculate Hcross by:

$$H^{\dagger} = (H^T H)^{-1} H^T$$

This is multiplied by the optical flow in order to get the velocity of the camera relative to the world in the camera frame.

These values need to be transformed to the world frame. This is done by first multiplying the velocity by the adjoint in order to get the velocity of the body relative to the world in the body frame and then multiplying that by the rotation to go from body to world frame.

The resulting velocities are passed through another low pass filter to provide better results.

3. RANSAC

The final portion of the project was to develop a RANSAC algorithm to reject outlier data points. The desired probability of success was set to 0.99, and the required number of iterations determined by:

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

M was set to 4 for a homography and epsilon set to 0.8 as the points are generally believed to be trustworthy. This works out to needing to run the algorithm at least 9 times.

The loop consists of generating a list of all indices of the position matrix in random order using randperm(). The first three values are selected and the H matrix generated the same way as part 2. The corresponding optical flow values are extracted and the velocity calculated.

The next step is to calculate the optical flow for every point in sets of three and using the same H as above. If the resulting velocity is within a threshold band set at the top of the function, those indices are added to the set of inliers. After checking all points, if the number of inliers found is greater than previously found, the indices of the inliers are overwritten along with the longest length.
After the process of determining inliers has been completed, the process of computing H and velocity from the optical flow is repeated, now using just the

inliers. The resulting velocity is then transformed as before using the adjoint and then converting to the value to be expressed in the world frame.

4. Results

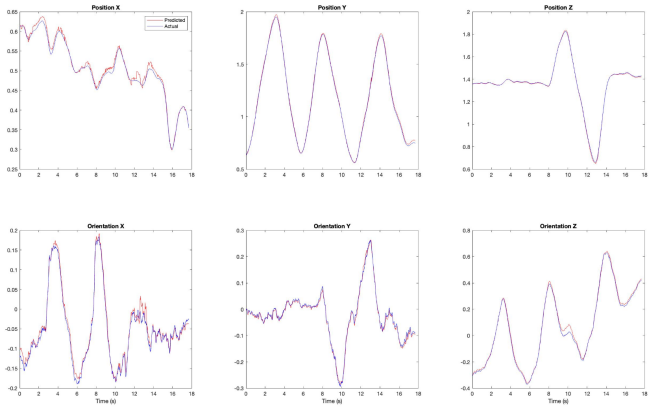

Figure 1: Part 1, Dataset 1

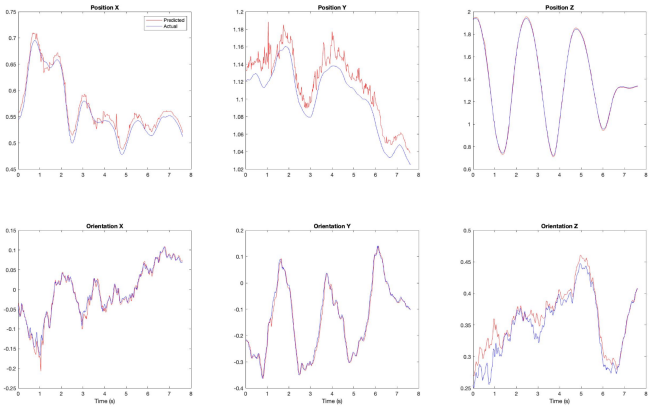

Figure 2: Part 1, Dataset 4

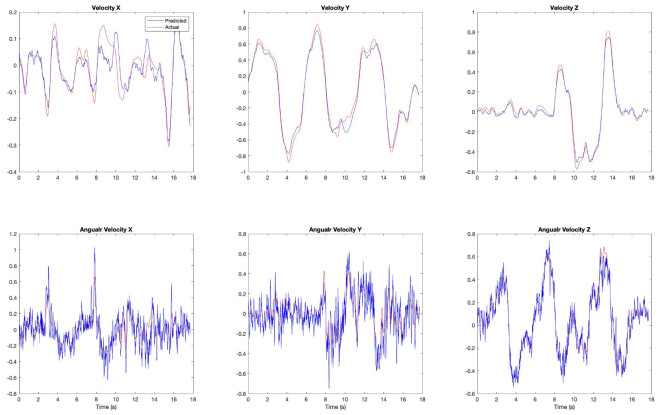

Figure 3: Part 2.1, Dataset 1, Pre Filtering Corners

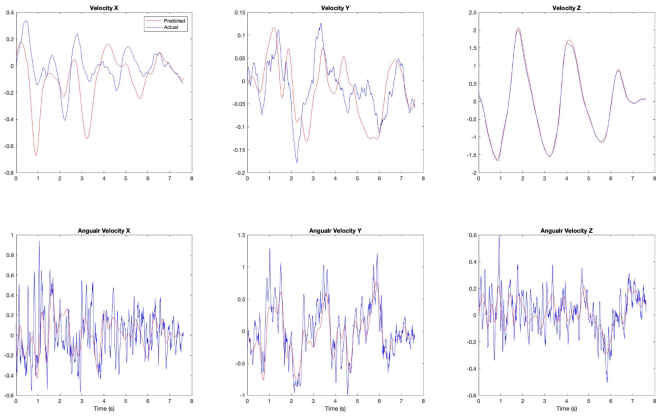

Figure 4: Model 2.1, Dataset 4, Pre Filtering Corners

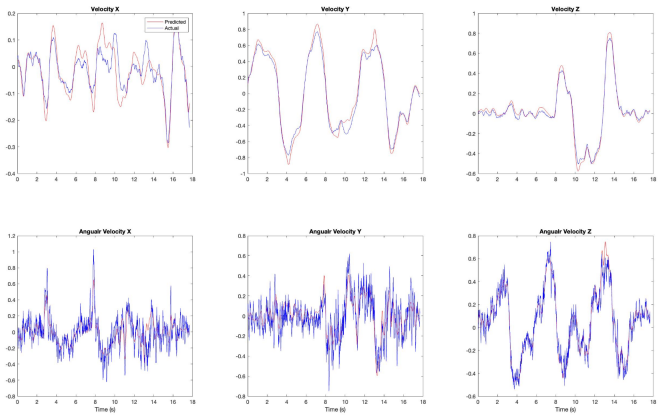

Figure 5: Model 2.2, Dataset 1, Pre Filtering Corners

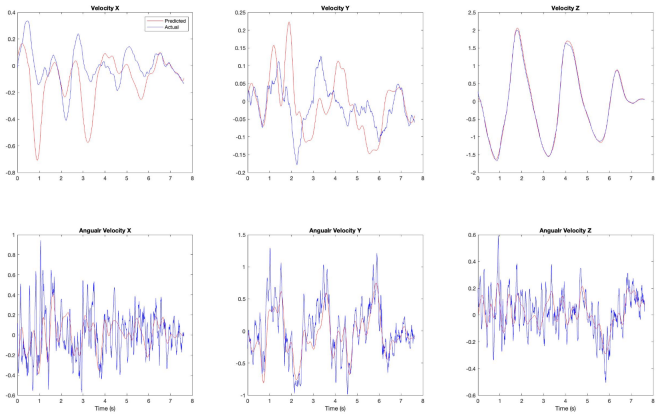

Figure 6: Model 2.2, Dataset 4, Pre Filtering Corners

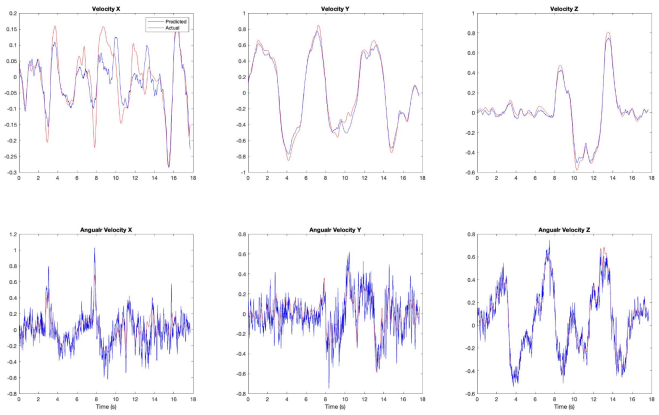

Figure 7: Model 2.1, Dataset 1, No Pre Filtered Corners

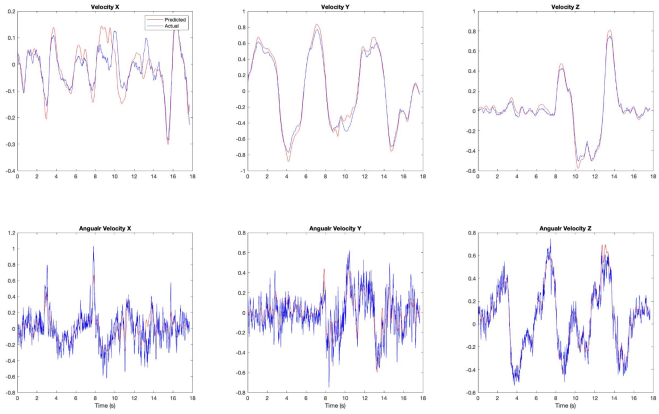

Figure 8: Model 2.2, Dataset 1, No Pre Filtered Corners

5. Discussion

The biggest point to note is the lack of change when comparing the RANSAC to the velocities generated in section 2.1. This is likely due to the implementation of pre-filtering the points returned by the corner tracker by checking against the validity vector that it returns. Presumably, this is removing all the points that should be rejected before the program has reached the RANSAC. By taking all the returned corners to be valid, a difference can more readily be seen by looking at figures 7 and 8. In 7 outliers can be seen, particularly in the x linear velocity that do not appear in other tests. The RANSAC then looks quite similar to figures 3 and 5 where the validity vector is being used to filter out corners.

The value of the threshold was set relatively higher than might initially be expected. The results seemed to work best when only the most extreme outliers were being rejected. This makes sense because a low number of iterations are being run, which is acceptable because generally the data can be considered trustworthy. This is expressed by the relatively high epsilon value 0.8 which was in the range of what was taught in class. Better results can be obtained but the improvement is not worth the cost in computation time.

The fact that the models do not align as well with dataset 4 is more difficult to explain. One possibility is that the vicon is detecting fairly large oscillations in the angular velocity, particularly on Z. It is possible that the amount of jitter being generated is diminishing the quality of the optical flow and ultimately velocities that are being calculated.

It should be noted, as there is a strict time requirement for the code to give an output within a minute, that part 2 is running in about 33-35 seconds for the larger dataset 1 during tests.