Project 2: Vision-Based 3D Attitude Estimation Using AprilTags

Robot Localization and Navigation
ROB-GY 6213
Version 1.0

Shantanu Ghodgaonkar
Univ ID: N11344563
Net ID: sng8399
Ph.No.: +1 (929) 922-0614

Contents

Abstract—**This report presents a vision-based 3D attitude estimation system for a Nano+ quadrotor using AprilTags. The system leverages camera calibration data and AprilTag information to estimate the pose of the quadrotor. Corner detection and tracking techniques are employed to extract and track image features across consecutive frames. Optical flow is then computed to relate the feature motion to the quadrotor's velocities. A linear relationship between optical flow and velocities is established, and depth information is incorporated to account for the 3D nature of the problem. Finally, RANSAC is employed to address outliers in the optical flow data, leading to more robust velocity estimation.**

1. Summary

In this report, we present the culmination of our efforts in implementing a robust system for vision-based 3D attitude estimation utilizing AprilTags. Our project focuses on accurately determining the positioning and orientation of a Nano+ quadrotor through meticulous calibration procedures and advanced computational techniques. Leveraging corner extraction methods, intrinsic camera calibration, and the KLT tracker, we establish a solid foundation for pose estimation. Through the analysis of image data and precise velocity estimation, we gain valuable insights into the quadrotor's motion dynamics. Additionally, the implementation of RANSAC-based outlier rejection techniques enhances the accuracy and reliability of velocity estimates derived from optical flow computations. Our findings not only showcase significant advancements in autonomous navigation systems but also underscore the effectiveness of vision-based methods for real-time pose estimation, particularly in the realm of aerial vehicles.

2. Introduction

2.1. Motivation and Applications

Accurate estimation of a quadrotor's 3D attitude (position and orientation) is crucial for various autonomous flight applications. This information is essential for tasks such as:

- **Navigation and obstacle avoidance**: Precise localization allows the quadrotor to navigate through a defined path while avoiding obstacles in its environment.
- **Landing and takeoff**: Accurate pose estimation ensures safe and controlled landing and takeoff manoeuvres.
- **Formation flying**: Maintaining a specific formation with other drones requires precise knowledge of each drone's relative position and orientation.
- **Visual servoing**: Vision-based pose estimation enables the quadrotor to control its movement based on visual cues from the environment.

Vision-based systems offer a promising approach for 3D attitude estimation due to their:

- **Low cost and weight**: Cameras are becoming increasingly affordable and lightweight, making them suitable for resource-constrained drones.
- **Versatility**: Vision systems can provide rich information about the surrounding environment, enabling real-time adaptation to changing conditions.
- **Passive sensing**: Unlike LiDAR or radar, cameras operate passively, making them less susceptible to interference.

2.2. Project Overview and Data Description

This project focuses on developing a vision-based 3D attitude estimation system for a Nano+ quadrotor using AprilTags. AprilTags are unique, black-and-white fiducial markers that can be easily detected and identified by a camera.

The provided data consists of the following:

- **Camera calibration data**: This data includes the intrinsic parameters of the camera, such as focal length and distortion coefficients, which are crucial for accurate image measurements.
- **AprilTag information**: This includes the physical dimensions of the AprilTags and their arrangement within the environment (AprilTag mat).
- **Image data**: A set of images captured by the camera mounted on the Nano+ quadrotor as it flies over the AprilTag mat. Each image contains information about the observed AprilTags, including their IDs and corner locations within the image frame. The data might also include rectified images (not required for this phase) and IMU data (for future extensions).
- **Ground truth data**: Vicon data is provided as the ground truth for comparison with the estimated pose. Vicon is a motion capture system that accurately tracks the position and orientation of objects in real time.

2.3. Skeleton Code Description

The project provides a skeleton code that serves as a starting point for implementing the vision-based pose estimation system. This code likely includes functions for:

- **Image loading and pre-processing**: Reading image data and potentially applying necessary corrections or transformations.
- **AprilTag detection and identification**: Locating and identifying AprilTags within the image based on their unique patterns.
- **Corner extraction and tracking**: Identifying feature points (corners) in the image and tracking their movement across consecutive frames.
- **Optical flow calculation**: Determining the apparent motion of features between images, providing information about the scene's movement relative to the camera.
- (Future implementation) **Pose estimation**: Combining information from detected AprilTags, corner features, and optical flow to estimate the 3D pose of the quadrotor.

This report details the implementation of the core functionalities within the skeleton code to achieve vision-based 3D attitude estimation using the provided data.

3. Methodology

3.1. Inital Analysis of Given Code

3.1.1. Part 1. Upon observation of the given skeleton code, the following inferences were made -

- The file titled *getCorner.m* was the definition of a function that took the ID of an April Tag as an input and produced the *(x,y)* coordinates of its corners as the return value.
- The file titled *estimatePose.m* was the definition of the function responsible to find the estimated position and orientation of the drone based on captured image data from the camera. The images were taken when the drone was flown over a bed of 108 April Tags arranged in a 12x9 grid. The function *getCorner.m* was called within this one.
- The file titled *PoseEstimation.m* was the responsible for loading & iterating over sensor data, feeding it to *estimatePose.m* and arranging the returned value appropiately for plotting.
- The file titled *init.m* was the definition of a function responsible for loading the chosen dataset.
- The file titled *plotData.m* was responsible to plot the predicted and actual output of the pose.

3.1.2. Part 2. Upon observation of the given skeleton code, the following inferences were made -

- The files titled *getCorner.m*, *init.m* and *estimatePose.m* were replicated here with the sole difference that *estimatePose.m* also returned the rotation matrix R_c^w.
- The file titled *OpticalFLow.m* was the responsible for the computation of the optical flow and thus the linear and angular velocity of the UAV. It also had space for the implementation of the RANSAC Algorithm.
- The file titled *velocityRANSAC.m* was the definition of a function that took the following input parameters and produced the inlier velocity increasing the robustness of the system -
  - optV = The optical Flow
  - optPos = Position of the features in the camera frame
  - Z = Height of the drone
  - R_c2w = Rotation defining camera to world frame
  - e = RANSAC hyper parameter
- The file titled *plotData.m* was responsible to plot the predicted and actual velocity values.

3.2. Dataset structure

Each dataset had three variables inside it -

- $sampledData$ was a $1 \times n$ array of type $struct$ containing the sensor packet at each timestamp. It's components were as follows -
  1) is_ready : True if a sensor packet is available, false otherwise

2) t : Time stamp for the sensor packet, different from the Vicon time
3) rpy : orientation of the body
4) omg : Body frame angular velocity from the gyroscope
5) acc : Body frame linear acceleration from the accelerometer
6) img : Undistorted image.
7) id : IDs of all AprilTags detected, empty if no tag is detected in the image
8) $p0$: Corners of the detect AprilTags in the image,
9) $p1$: the ordering of the corners, and the distribution of the tags
10) $p2$: the ordering of the corners, and the distribution of the tags
11) $p3$: the ordering of the corners, and the distribution of the tags
12) $p4$: the ordering of the corners, and the distribution of the tags

- $sampledVicon$ was a $12 \times n$ array of type $double$ where each column's structure was defined as follows -

$$\left[x \ y \ z \ roll \ pitch \ yaw \ v_x \ v_y \ v_z \ \omega_x \ \omega_y \ \omega_z\right]^T$$

- $sampledTime$ was a $1 \times n$ array of type $double$ containing each timestamp

3.3. Part 1 implementation

3.3.1. getCorners.m. The logic for this function was fairly simple. First, let us observe the given bed of April tags as shown in below Figure (1) -

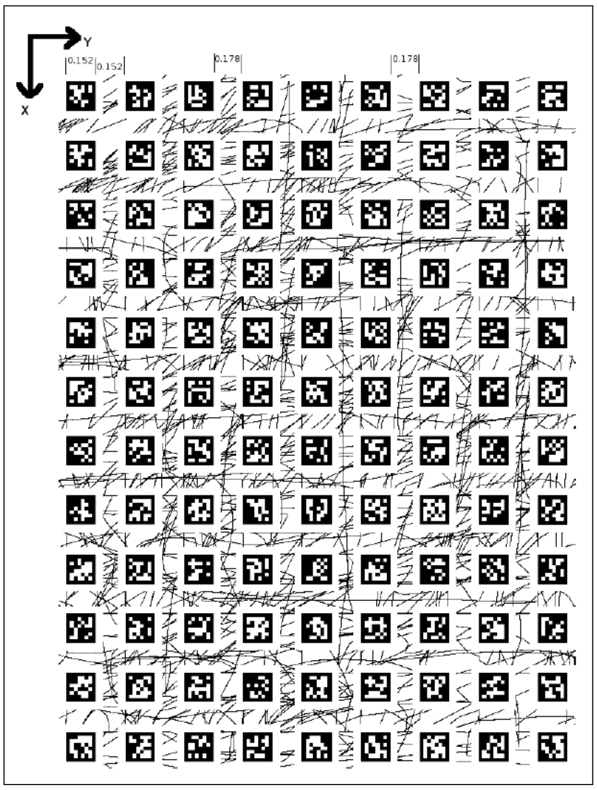


Figure 1. April Tag Mat or Bed *(Excerpt from Project 2 Handout)*

It shows us the size of each April Tag and the spacing between each of the rows and columns. Please note the exception of the space between columns 3 and 4, and 6 and 7, which is 0.178m.

The numbering of each corner is given as shown in below Figure (2)

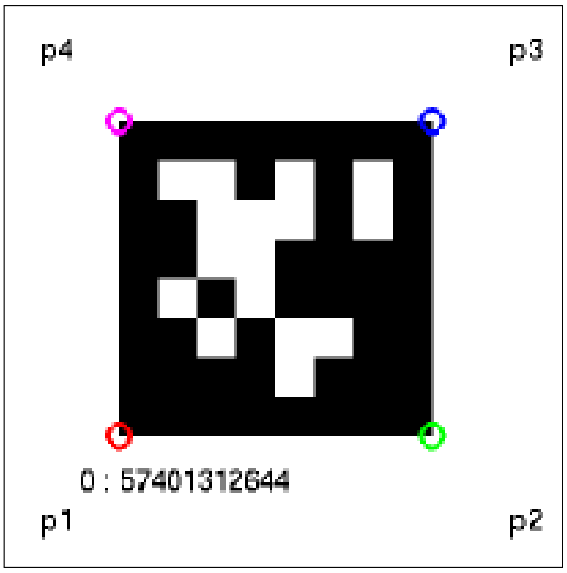


Figure 2. Corners of the AprilTag *(Excerpt from Project 2 Handout)*

Based on this, the *getCorners.m* function was formulated to find p4 of each April tag using below given logic -

```
col = floor((id / 12)) + 1; %
    Calculate the column number of
    the tag
row = mod(id, 12) + 1; % Calculate
    the row number of the tag
p4X = 2 * ((row - 1) * 0.152); %
    Calculate the X coordinate of p4
% Calculate the Y coordinate of p4
    based on the column number
% accounting for the uneven column
    spacing
if(col <= 3)
     p4Y = 2 * ((col - 1) * 0.152);
elseif ((col > 3) && (col <= 6))
     p4Y = ((2 * (col - 1)) * 0.152)
         + 0.026;
elseif (col > 6)
     p4Y = ((2 * (col - 1)) * 0.152)
         + 0.052;
end
```

Finally, the other corners were found by simply adding or not adding the side length of an April Tag to the x or y or both coordinates as shown below -

```
% res = [p1, p2, p3, p4]
res = [ [(p4X + 0.152); p4Y],
      [(p4X + 0.152);(p4Y + 0.152)],
      [p4X; (p4Y + 0.152)],
      [p4X; p4Y]];
```

3.3.2. estimatePose.m. The implementation for this function took the corners of all the full April Tags detected (partial ones are neglected) in the camera frame and the corners of all the same April Tags (found using *getCorners.m*) in the world frame to first find the homography matrix.

Let the coordinates of each point in the world frame be given by, $p_w = \begin{bmatrix} x_i & y_i \end{bmatrix}^T$, while those of each point in the camera frame be given by, $p_c = \begin{bmatrix} x_i' & y_i' \end{bmatrix}^T$. Then the relation between the two is given by the homography matrix h as shown in below Eq^n (1)

$$p_c \sim h\ p_w \Rightarrow \hat{p_c}\ h\ p_w = 0 \tag{1}$$

Where $\hat{p_c}$ is the skew-symmetric form of p_c. A few algebra tricks later we get,

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'\,x_i & -x_i'\,y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'\,x_i & -y_i'\,y_i & -y_i' \end{bmatrix} h = 0 \tag{2}$$

But as per above Eq^n (2), $\hat{p_c}$ imposes only 2 independent constraints due to its rank 2. In order to get all 8 degrees of freedom, we can stack all the four 2x9 matrices to get a big 8x9 matrix called A such that

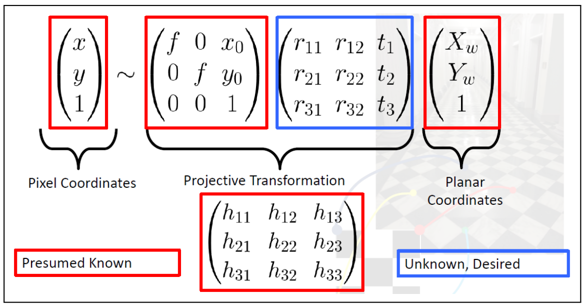
$$Ah = 0 \tag{3}$$

This relation can help us find the homography for a single April Tag. All the A matrices computed for each April Tag can be stacked over one another such the number of the columns remains the same. Once the vertically concatenated A matrix is formed, the homography h can be easily found by performing SVD on the A matrix and extracting the 9^{th} column of V and storing the elements in row order into the 3x3 homography matrix. Finally, to guarantee the solution with positive z, h is scaled with the last element of the last column of V, *i.e.*, $h = sign(V(9,9)) * h$.

Now, we must extract the pose from the homography matrix. Firstly, we are already given the camera intrinsic matrix,

$$K = \begin{bmatrix} 311.0520 & 0 & 201.8724 \\ 0 & 311.3885 & 113.6210 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

Assuming that we are finding the points on the ground plane, we can consider the Z coordinate to be zero. Then, we can form the logic,

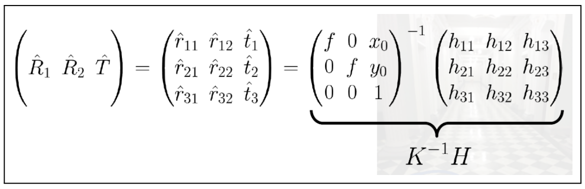


$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \sim \begin{pmatrix} f & 0 & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

Figure 3. Relation between h, K, R & T *(Excerpt from Lecture 8 PPT)*

Building upon the same logic, we can see that,



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

Figure 4. Extraction of rotation & translation *(Excerpt from Lecture 8 PPT)*

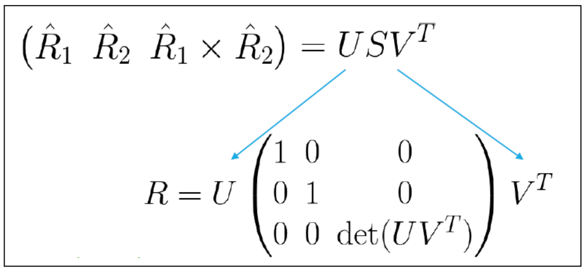
But this is not the end. R_1 and R_2 need to satisfy the constraints,

$$R_1^T R_2 = 0 \qquad ||R_1|| = ||R_2|| = 1 \tag{5}$$

Additionally, we are yet to find the Z part of the rotation and translation. Firstly, to satisfy the above constraints, we form the matrix,

$$\begin{bmatrix} R_1 & R_2 & R_1 \times R_2 \end{bmatrix} \tag{6}$$

Where $R_1 \times R_2$ gives us the Z part as it is orthogonal to the plane formed by R_1 and R_2 (cross product of two vectors results in a third vector that is orthogonal to the plane formed by the initial two). We then take the SVD of this newly formed matrix to obtain the rotation estimate using the logic shown in below Figure (5)



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

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

Figure 5. Finding the rotation *(Excerpt from Lecture 8 PPT)*

The diagonal guarantees it is a rotation matrix.

To find our estimate of the translation we just make sure it is in the right scale:

$$T = \hat{T}/||\hat{R}|| \tag{7}$$

Note that we still have to transform everything from the camera frame into the IMU frame. This can be done by combining the estimates of rotation and translation into a 4x4 matrix and multiplying it with the Camera to Body transformation that was found using the given images in Figures (6) & (7).

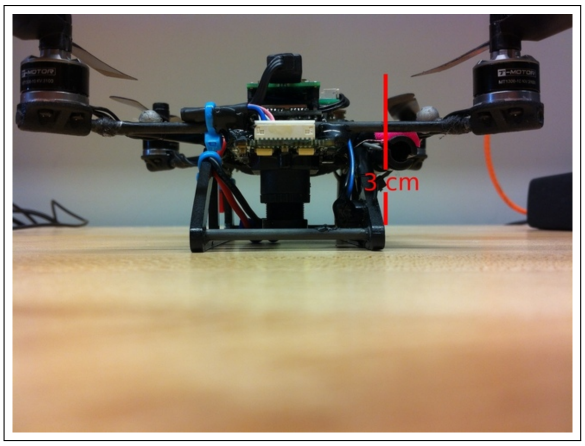


Figure 6. Side View of quadrotor

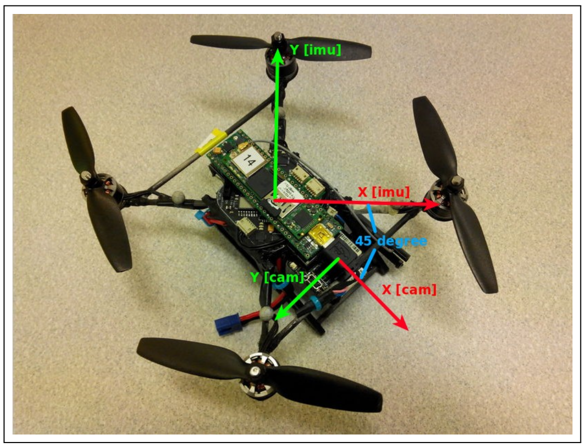


Figure 7. Top View of quadrotor

$$H_c^b = \begin{bmatrix} 0.7071 & -0.7071 & 0 & -0.04 \\ -0.7071 & -0.7071 & 0 & 0 \\ 0 & 0 & -1 & 0.03 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

Finally, upon applying this homogeneous transformation to the rotation (convert to Euler Angles) and translation estimate, we get the position and orientation of the drone in 3D.

3.4. Part 2 implementation

For this part, *getCorner.m* and *estimatePose.m* remain the same with the sole difference that *estimatePose.m* also returned the rotation matrix R_c^w.

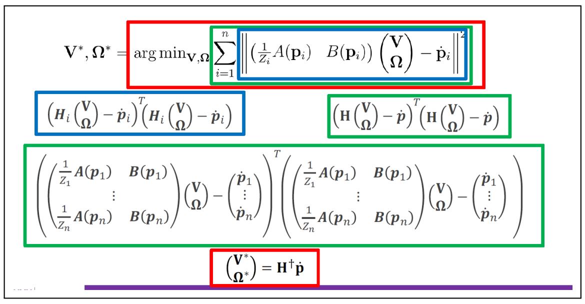
3.4.1. OpticalFLow.m. The first part of this implementation involves the intialisation of certain data like the camera intrinsic matrix, the body to camera transformation, camera to body rotation, skew symmetric matrix form of the translation from camera to body and finally, the filtered values of sampled time. The filtration is performed using Savitzky-Golay finite impulse response (FIR) smoothing filter [8] of polynomial order 1 and frame length 101.

Next, a for loop is set to iterate from 2 to the length of the sampled data. Inside the first iteration of this for loop, we take the first image in the sampled data and find all corners using the Harris–Stephens algorithm [4]. We shall consider only the first 100 strongest detected points and extract each of their locations in the camera frame.

Based on the detected corners, we shall initialise a point tracker that uses Kanade-Lucas-Tomasi (KLT) algorithm [9] to track points. We shall then supply the next image in the sensor data structure to this point tracker to get new coordinates of the same corners detected in the previous image.

The last thing that we need before we can compute the linear and angular velocity is the pose of the drone at given time t. This can be found using the *estimatePose.m* function that we perfected earlier. Using this, we can extract the height(Z) of the drone's camera as per the current pose by rotating its position from the body to camera frame and extracting the third element of the resulting 4x1 vector.

Now that we have everything, it is only a matter of writing another for loop to iterate over all the corners points detected & tracked, calculating the $\dot{p}$ by finding the difference between the current & previous coordinates of each detected corner and dividing by dt and applying the logic as shown in below Figure (8). Note that we take the pseudoinverse of H using the Moore-Penrose pseudoinverse method [7].



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

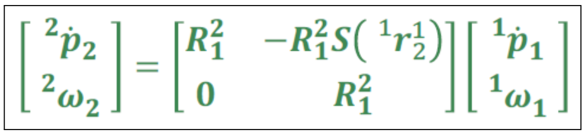
$$\boxed{\left(H_i \begin{pmatrix} \mathbf{V} \\ \mathbf{\Omega} \end{pmatrix} - \dot{p}_i\right)^T \left(H_i \begin{pmatrix} \mathbf{V} \\ \mathbf{\Omega} \end{pmatrix} - \dot{p}_i\right)} \qquad \boxed{\left(\mathbf{H} \begin{pmatrix} \mathbf{V} \\ \mathbf{\Omega} \end{pmatrix} - \dot{p}\right)^T \left(\mathbf{H} \begin{pmatrix} \mathbf{V} \\ \mathbf{\Omega} \end{pmatrix} - \dot{p}\right)}$$

$$\boxed{\left(\begin{pmatrix} \frac{1}{Z_1}A(p_1) & B(p_1) \\ \vdots & \\ \frac{1}{Z_n}A(p_n) & B(p_n) \end{pmatrix} \begin{pmatrix} \mathbf{V} \\ \mathbf{\Omega} \end{pmatrix} - \begin{pmatrix} \dot{p}_1 \\ \vdots \\ \dot{p}_n \end{pmatrix} \right)^T \left(\begin{pmatrix} \frac{1}{Z_1}A(p_1) & B(p_1) \\ \vdots & \\ \frac{1}{Z_n}A(p_n) & B(p_n) \end{pmatrix} \begin{pmatrix} \mathbf{V} \\ \mathbf{\Omega} \end{pmatrix} - \begin{pmatrix} \dot{p}_1 \\ \vdots \\ \dot{p}_n \end{pmatrix} \right)}$$

$$\boxed{\begin{pmatrix} \mathbf{V}^* \\ \mathbf{\Omega}^* \end{pmatrix} = \mathbf{H}^\dagger \dot{\mathbf{p}}}$$

Figure 8. Motion Field Equations for Known Depth *(Excerpt from Lecture 9 PPT)*

Finally, we must transform the resulting 6x1 matrix from the camera into the IMU or body frame using the adjoint of the matrix (Figure (9)).



$$\begin{bmatrix} {}^2\dot{p}_2 \\ {}^2\omega_2 \end{bmatrix} = \begin{bmatrix} R_1^2 & -R_1^2 S({}^1r_2^1) \\ 0 & R_1^2 \end{bmatrix} \begin{bmatrix} {}^1\dot{p}_1 \\ {}^1\omega_1 \end{bmatrix}$$

Figure 9. Adjoint formula *(Excerpt from Lecture 3 PPT)*

3.4.2. velocityRANSAC.m. This is the final part of the project. First, we must set a few parameters -

- $p_{success} = 0.925$
- $M = 3$
- $\epsilon = 0.7$
- $\beta = 0.1$

Then we must find the number of iterations needed for the maximum inlier detection using the formula -

$$k = \frac{log(1 - p_{success})}{log(1 - \epsilon^M)} \tag{9}$$

Now we simply implement this algorithm (in pseudocode) as shown below -

Algorithm 1 RANSAC Algorithm

```
for $i = 1, \ldots, k$ do
    Choose $M$ non-repeated points $p_1$, $p_2$ $\ldots p_M$ from all detected points for this sensor packet
    Compute the $H = \begin{bmatrix} \frac{1}{Z}A(p) & B(p) \end{bmatrix}$ for each point
    Vertically concatenate all $H$ matrices to get a matrix of size 6x6
    Compute $\dot{p}$ for each point and vertically concatenate them to get a matrix of size 6x1
    Compute $\begin{bmatrix} V^* \\ \Omega^* \end{bmatrix} = H^\dagger\ \dot{p}$
    Compare all points in this set of points with above $\begin{bmatrix} V^* \\ \Omega^* \end{bmatrix}$ using the formula
        $\left|\left| \begin{bmatrix} \frac{1}{Z}A(p_i) & B(p_i) \end{bmatrix} \begin{bmatrix} V^* \\ \Omega^* \end{bmatrix} - \dot{p}_i \right|\right|^2 \leq \beta \quad (10)$
    And keep a count of number points that satisfy the condition shown in $Eq^n$ (10)
    Keep maximum, if it exceeds a desired number of inliers, stop and return the velocity calculated with the greatest number of inliers
end for
```

Thus, the above algorithm was implemented in MATLAB and the results are discussed in the next section.

4. Results

The implemented Pose Estimation, Optical Flow and RANSAC combined is a efficient and robust system for the vision-based computation of quadrotor attitude in real-time.

4.1. Part 1: Pose Estimation

As evident in the figures, we are effectively able to estimate the pose for the given sensor data. Figures 10 to 15 are the plots for *studentdata1.mat* -

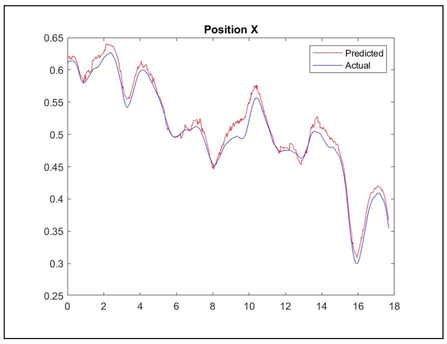


Figure 10. Position X.png

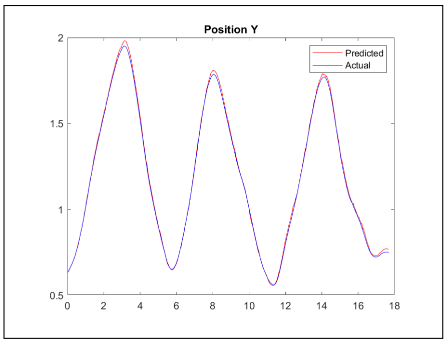


Figure 11. Position Y.png

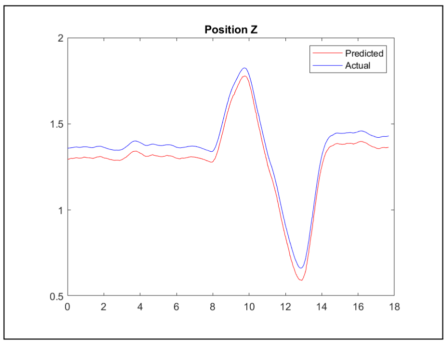


Figure 12. Position Z.png

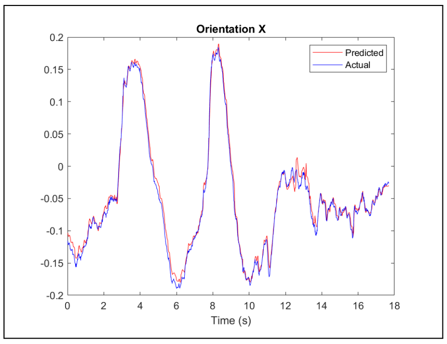


Figure 13. Orientation X.png

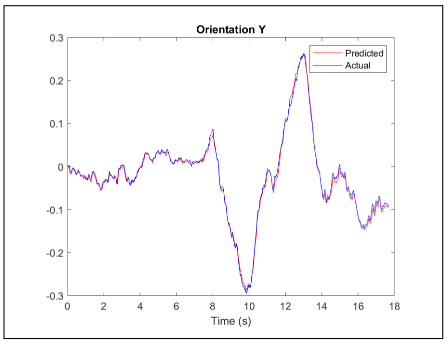


Figure 14. Orientation Y.png

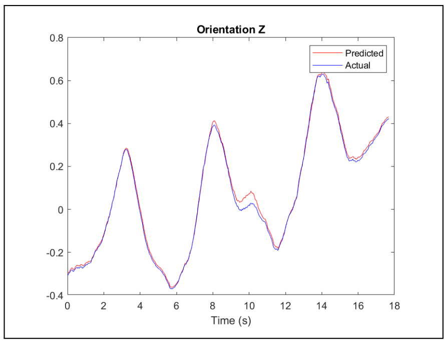


Figure 15. Orientation Z.png

Below Figures 16 to 21 are the plots for *student-data4.mat*-

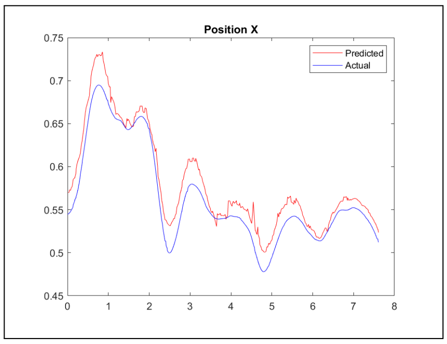


Figure 16. Position X.png

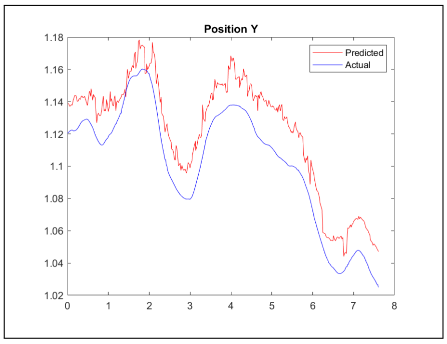


Figure 17. Position Y.png

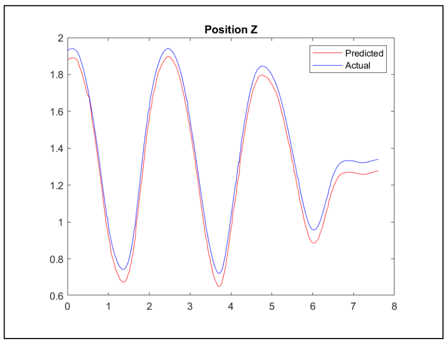


Figure 18. Position Z.png

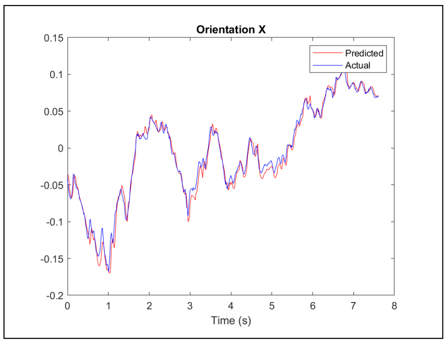


Figure 19. Orientation X.png

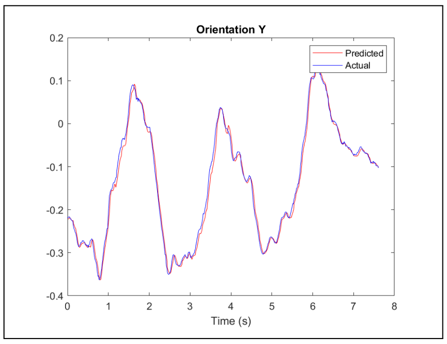


Figure 20. Orientation Y.png

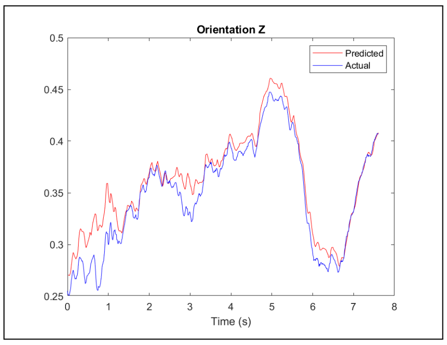


Figure 21. Orientation Z.png

4.2. Part 2: Velocity Estimation & RANSAC

The velocity estimation is rather jittery although there exists a LPF to reduce the jitters. Shown in below Figures 22 to 27 are the plots for *student-data1.mat without using* RANSAC-

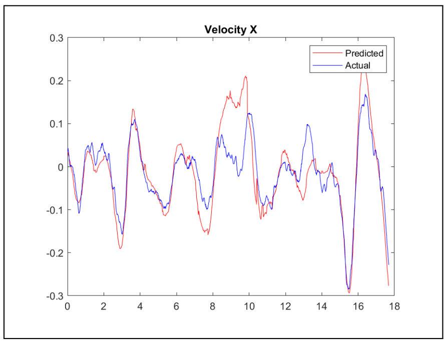


Figure 22. Velocity X.png

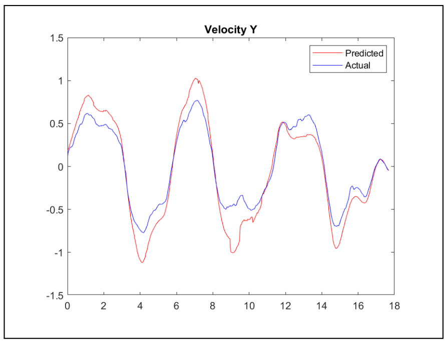


Figure 23. Velocity Y.png

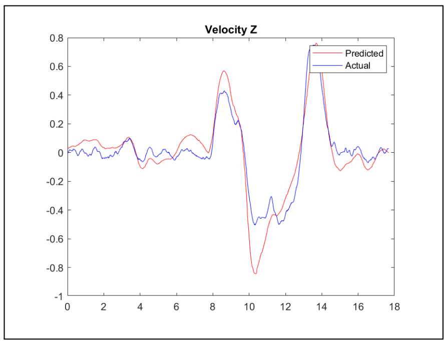


Figure 24. Velocity Z.png

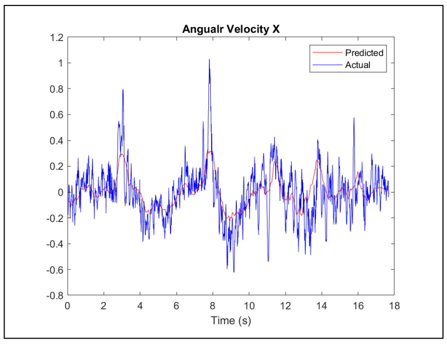


Figure 25. Angular Velocity X.png

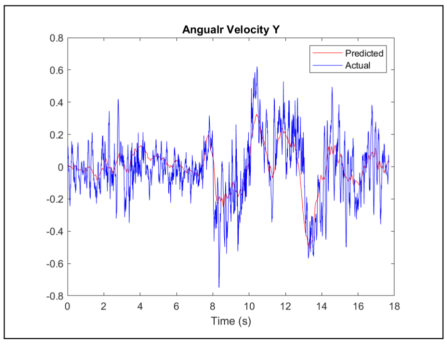


Figure 26. Angular Velocity Y.png

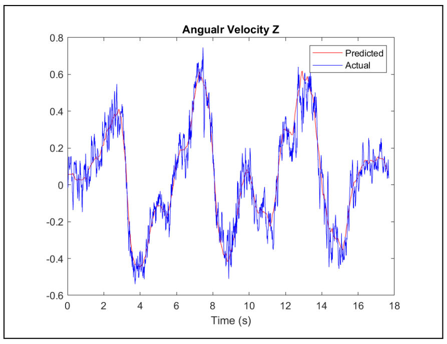


Figure 27. Angular Velocity Z.png

Below Figures 28 to 33 are the plots for *student-data4.mat without using* RANSAC-

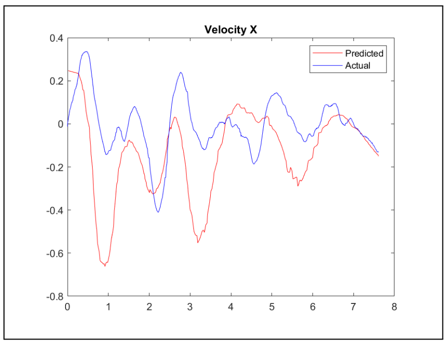


Figure 28. Velocity X.png

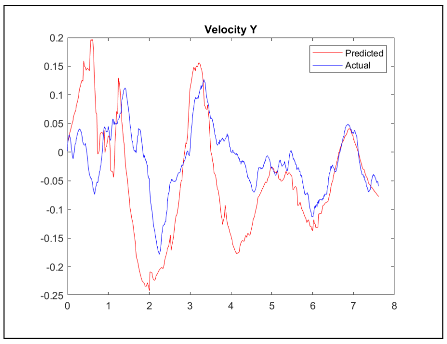


Figure 29. Velocity Y.png

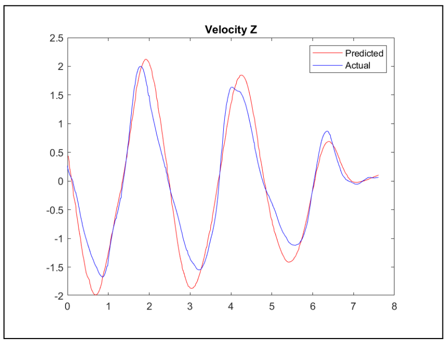


Figure 30. Velocity Z.png

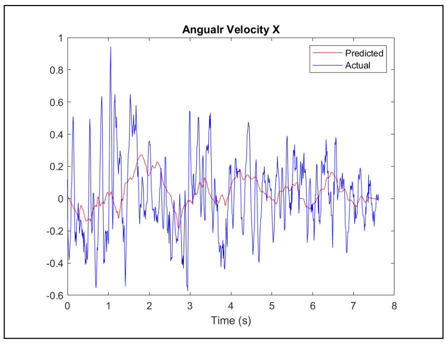


Figure 31. Angular Velocity X.png

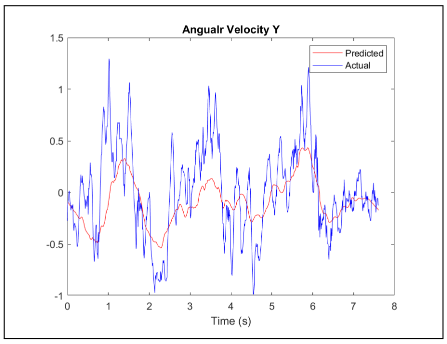


Figure 32. Angular Velocity Y.png

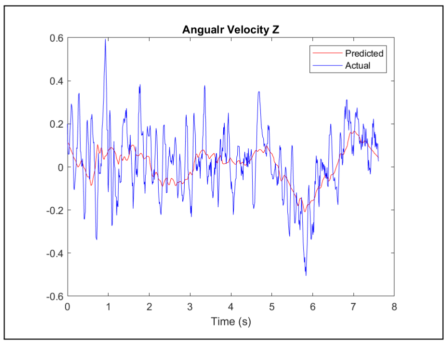


Figure 33. Angular Velocity Z.png

Below Figures 34 to 39 are the plots for *student-data1.mat using* RANSAC-

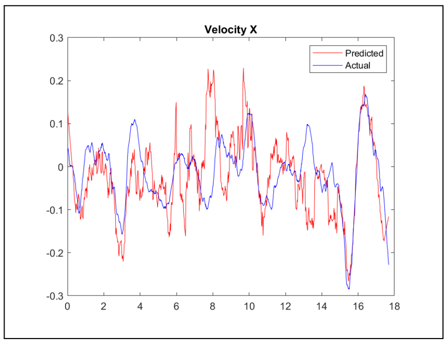


Figure 34. Velocity X.png

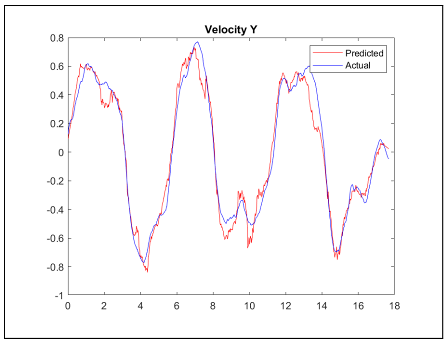


Figure 35. Velocity Y.png

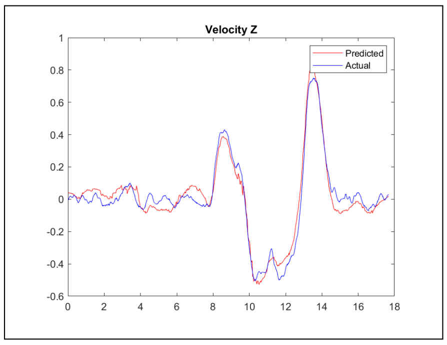


Figure 36. Velocity Z.png

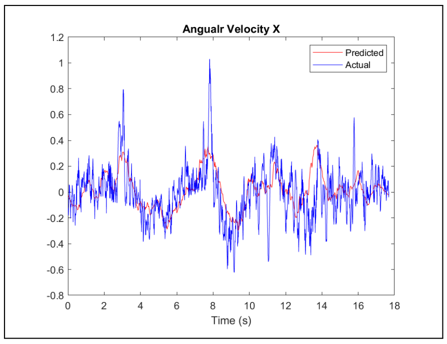


Figure 37. Angular Velocity X.png

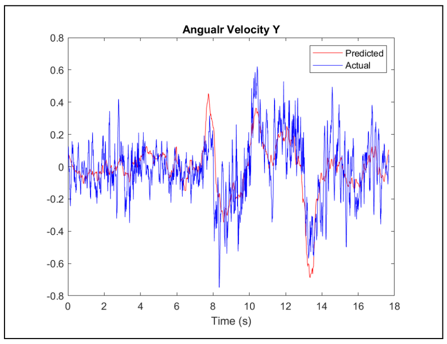


Figure 38. Angular Velocity Y.png

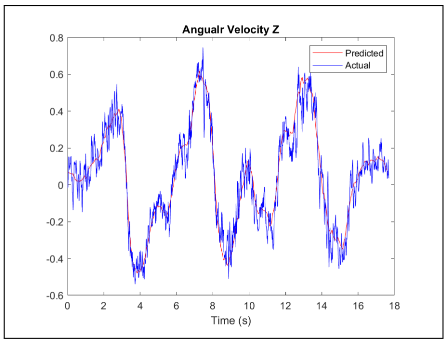


Figure 39. Angular Velocity Z.png

Below Figures 40 to 45 are the plots for *student-data4.mat using* RANSAC-

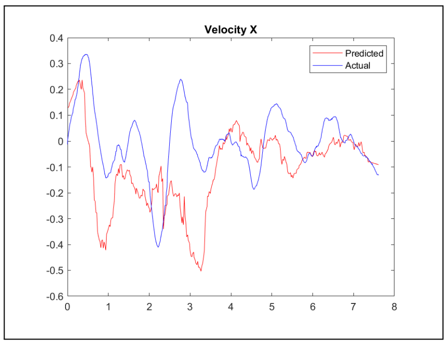


Figure 40. Velocity X.png

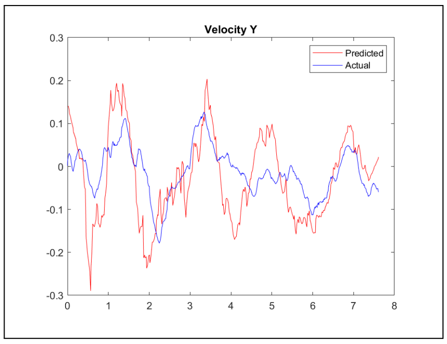


Figure 41. Velocity Y.png

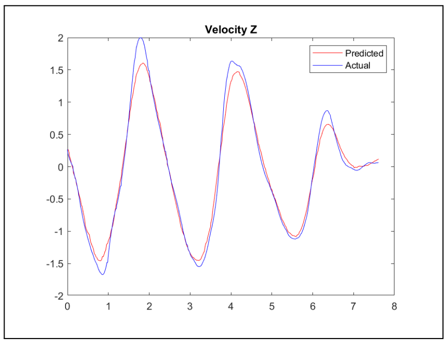


Figure 42. Velocity Z.png

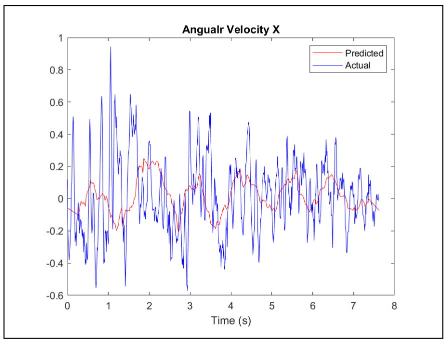


Figure 43. Angular Velocity X.png

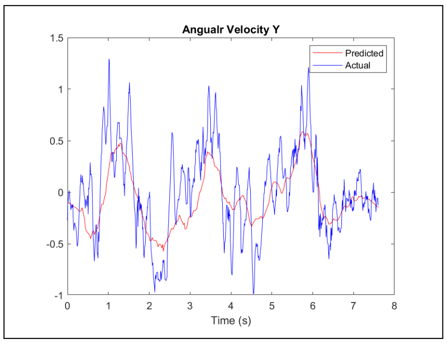


Figure 44. Angular Velocity Y.png

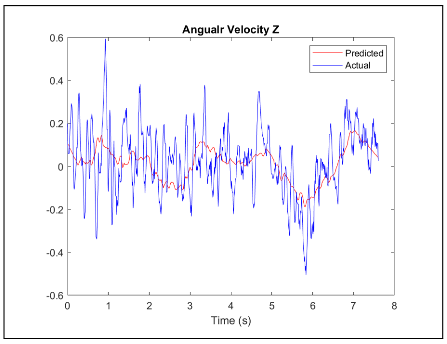


Figure 45. Angular Velocity Z.png

5. Conclusion

In conclusion, This project presented a vision-based system for estimating the 3D attitude (position, orientation and velocity) of a Nano+ quadrotor using AprilTags. The system leveraged camera calibration data, AprilTag information, and corner features extracted from captured images. Optical flow was computed to relate feature motion to the quadrotor's velocities. A linear relationship between optical flow and velocities was established, incorporating depth information to account for the 3D nature of the problem. Finally, RANSAC was employed to address outliers in the optical flow data, leading to more robust velocity estimation.

The developed system demonstrates the feasibility of using AprilTags and vision techniques for quadrotor pose estimation. The project successfully implemented core functionalities within the provided skeleton code and provided a fairly refined output, with small scope for improvement.

References

[1] APRIL Robotics Laboratory, University of Michigan. *AprilTags Visual Fiducial System*. 2010. URL: https://april.eecs.umich.edu/software/apriltag.

[2] Bruce D. Lucas and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *International Joint Conference on Artificial Intelligence*. 1981. URL: https://api.semanticscholar.org/CorpusID:2121536.

[3] Yi Ma et al. *An Invitation to 3-D Vision: From Images to Geometric Models*. SpringerVerlag, 2003. ISBN: 0387008934.

[4] MathWorks. *Detect corners using Harris–Stephens algorithm*. 2013. URL: https://www.mathworks.com/help/vision/ref/detectharrisfeatures.html?s_tid=doc_ta#btolyhy-1_vh.

[5] MathWorks. *MATLAB Computer Vision Toolbox*. 2024. URL: https://www.mathworks.com/products/computer-vision.html.

[6] MathWorks. *MATLAB Image Processing Toolbox*. 2024. URL: https://www.mathworks.com/products/image-processing.html.

[7] MathWorks. *Moore-Penrose pseudoinverse*. 2021. URL: https://www.mathworks.com/help/matlab/ref/pinv.html.

[8] MathWorks. *Savitzky-Golay filtering*. 2006. URL: https://www.mathworks.com/help/signal/ref/sgolayfilt.html#f8-307571_vh.

[9] MathWorks. *Track points in video using Kanade-Lucas-Tomasi (KLT) algorithm*. 2012. URL: https://www.mathworks.com/help/vision/ref/vision.pointtracker-system-object.html.

[10] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 2010. ISBN: 1849966346.

[11] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 1st. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN: 1848829345.