Vision-Based Odometry System for Localization of Aerial Vehicles

A project presented to
The Faculty of the Department of Aerospace Engineering
San José State University

in partial fulfillment of the requirements for the degree
Master of Science in Aerospace Engineering

by

Connor J. Miholovich

December 2024

approved by

Dr. Jeanine Hunter
Faculty Advisor

ABSTRACT

Vision-Based Odometry System for Localization of Aerial Vehicles

Connor J. Miholovich

The purpose of this report is to show the design, implementation, and testing of an online Visual Odometry (VO) algorithm used for navigation and localization of an Unmanned Aerial Vehicle (UAV). Aerospace navigation systems primarily rely on Global Positioning System (GPS) and Inertial Navigation Systems (INS) for information on the aircraft's position and orientation. However, aircraft operations in GPS-denied or degraded environments such as cities, forests, or indoors are becoming more prominent with UAVs and classic navigation methods begin to fail in GPS-denied environments. The report proposes a practical method for navigation in GPS-denied environments to be incorporated on a multirotor UAV using low-cost generalized hardware. The method is based on comparing the onboard camera data of a UAV to a pre-built map of the drone's environment using satellite images. The VO navigation systems aim to reduce errors introduced into the system through sensor bias and drift and rely solely on onboard data rather than constellation data that may not be available in any environment. Simulation and testing of the methodology aim to prove the performance of the navigation system and implement changes needed to bridge the simulation to the real-world gap.

Acknowledgments

I would like to give special thanks to my advisor Professor Jeanine Hunter for her support and guidance throughout the project. Her role has been pivotal in unlocking new resources and skills in my academic experience. Her contributions were critical for ensuring that the project was relevant to real industry needs and all experiments were done with safety considerations in mind. I would also like to give thanks to my professional mentors who I worked alongside for introducing me to Computer Vision and offering many great resources along the way. I would like to thank the members of the Santa Clara County Model Airfield for being welcoming and allowing me to use their park for flight testing. Their passion for the hobby of model aircraft made testing on the project enjoyable and I was able to learn more about model aircraft design and flight! I would like to thank my parents for supporting my academic journey and for inspiring me to work harder each day. Lastly, I would like to thank my friends for all the conversations, meeting up for late-night coffee runs, and support along the way.

Table of Contents

List Of Figures

Symbols

Symbol	Definition	Units (SI)
W	3D point in world coordinate system	m
P’	Projected 2D point on image plane	pixels
u, v	Pixel coordinates	Pixels
f	Focal length	m
fx, fy	Focal lengths in x and y directions	pixels
cx, cy	Optical center coordinates	Pixels
R	Rotation matrix	Dimensionless
t	Translation Vector	m
K	Camera intrinsic matrix	Dimensionless
F	Fundamental matrix	Dimensionless
E	Essential matrix	Dimensionless
H	Homography matrix	Dimensionless
n	The normal vector of a plane	Dimensionless
d	Distance to a plane	m
G	Gaussian Function	Dimensionless
σ	Standard Deviation	Varies
Gx, Gy	Sobel operators for x and y directions	Dimensionless
Θ	Gradient direction	Radians
Ip	Pixel brightness	Dimensionless
T	Threshold for FAST feature detection	Dimensionless
q0, q1, q2, q3	Quaternion components	Dimensionless
x	State Vector	Varies
z	Measurement Vector	Varies
w	Process Noise	Varies

v	Measurement Noise	Varies
P	Covariance Matrix	Varies
Q	Process Noise Covariance Matrix	Varies
K	Kalman Gain	Dimensionless
Λ	Weighing factor in EKF estimate	Dimensionless
C	Speed of Light vacuum	m/s
L	Loss Function	Dimensionless
Y	Feature Map	Pixels
k_h	Kernel Height	Pixels
k_w	Kernel Width	Pixels
W	Weight matrix	Dimensionless
B	Bias Vector	Dimensionless
S	Stride	Pixels
W_k	Kernel	Dimensionless
I	Pixel Intensity	Dimensionless
σ	Standard Deviation	Varies
μ	Average Value	Varies
v_i	Original Pixel Value	Dimensionless
σ_a	Activation Function	Dimensionless
δ_k^o	Hidden Layer Error	Dimensionless
δ_{output}	Output Layer Error	Dimensionless
x_i	Input Unit	Dimensionless
w_{ij}	Hidden and Input Layer Weights	Dimensionless

1. Introduction

1.1 Motivation

Commercial aviation utilizes Global Positioning System (GPS) as one of the main sensors responsible for localization and navigation. Depending on the degree of accuracy required, some systems may elect to use Real-Time-Kinematics (RTK) GPS to increase localization accuracy to the order of centimeters. [1] RTK GPS operates similarly to GPS where it receives a signal from satellites, but also receives a signal from a stationary (on Earth) beacon that provides corrections to the satellite's signal. However, GPS signal interruption from areas of low network coverage or high interference (signal jamming, buildings, underground, canyons, etc.) limits aircraft navigation capabilities. While most modern aircraft feature highly redundant systems, GPS corrects other sensors which are more prone to drift in measurements over long flights. When GPS signals are not present, human pilots use Visual Flight Rules (VFR) where sight is used to localize the aircraft in the surrounding area. Some limitations to using VFR are altitude, visibility, and airspeed which can all hinder a pilot's ability to observe, process, and react to surroundings accordingly. For remote-controlled aerial vehicles pilots may rely on onboard cameras or visual line-of-sight for navigation when GPS signals are not reliable or present. In robotics applications global localization is the ability of a robot to know its position and orientation in an environment without knowledge of its original state and orientation. Simultaneous Localization And Mapping (SLAM) is a method used by autonomous vehicles to localize, map, and track the environment around them. After a robot has built a map of the environment around it and localized itself, SLAM can be applied to collision avoidance and path planning tasks. [2] Operating aerial vehicles in GPS-denied environments proposes difficulties for guidance and navigation problems, specifically with respect to path planning and localization which is the problem this project focuses on solving.

1.2 Literature Review

In recent years, advancements in camera technologies and compute platforms have spurred significant research in vision-based navigation systems within aerospace and robotics fields. Vision-based systems excel in localization and mapping regions in GPS-denied environments, leveraging data of varying resolutions. In robotics, Visual Odometry (VO) is a critical process for determining the motion of a stereo or monocular camera through video input. VO operates by comparing and tracking features between video frames, subsequently using these tracked features to form paths at the video rate. [3] Consequently, VO algorithms are instrumental in determining a robot's relative position and orientation. Camera models handle the transformation of the 3-dimensional (3D) real-world reference frame and the 2-dimensional (2D) pixel frame. The camera model plays a crucial role in undoing distortion on raw images and correcting imperfections in the camera sensors. For computer vision purposes Real World, Camera, and Image reference frames are related to one another and are critical in the 3D to 2D transformations.

The Earth-centered, Earth-fixed (ECEF) frame is a global reference frame with its origin in the center of the Earth with three orthogonal axes fixed to the Earth. In the ECEF reference

frame as shown in Figure 1.1 the $\boldsymbol{e_z}$ axis points towards the North Pole, the $\boldsymbol{e_x}$ axis points through the intersection of the IERS Reference Meridian (IRM) and equator, the $\boldsymbol{e_y}$ axis is 90 degrees to the $\boldsymbol{e_x}$axis. Within the ECEF frame, two primary coordinate systems define a system position: cartesian and geodetic. Cartesian coordinates reference an object's position relative to the Earth with $\boldsymbol{e_x}$, $\boldsymbol{e_y}$, and $\boldsymbol{e_z}$ axes. Geodetic coordinates reference an object's position relative to Earth with longitude, latitude, and altitude. A local reference frame that can be fixed to a vehicle's body frame is the North-East-Down (NED) reference frame which is a common convention for aerospace. The $\boldsymbol{n_x}$ axis points toward the North Pole, the $\boldsymbol{n_y}$ axis pointing East, and the $\boldsymbol{n_z}$ axis points to the center of the ECEF reference frame.

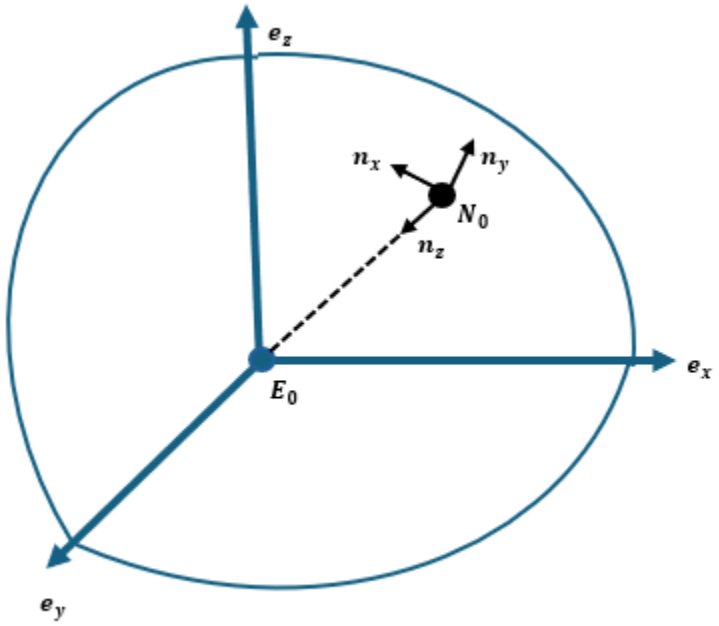


Figure 1.1: ECEF and NED reference frames.

The camera reference frame is defined as principle at $\boldsymbol{C_o}$ and three orthogonal axes $\boldsymbol{c_x}, \boldsymbol{c_y}$, and $\boldsymbol{c_z}$. A point in the world reference frame is defined about $\boldsymbol{W_0}$ and three orthogonal axes $\boldsymbol{w_x}, \boldsymbol{w_y}$, and $\boldsymbol{w_z}$ with units in meters. The body frame is defined with a center at $\boldsymbol{B_o}$ and three orthogonal axes $\boldsymbol{b_x}, \boldsymbol{b_y}$, and $\boldsymbol{b_z}$ with units in meters. The North-East-Down (NED) frame, is common in aerospace applications, and is defined with at center at $\boldsymbol{N_o}$ and three orthogonal axes

$\boldsymbol{n_x}, \boldsymbol{n_y}$, and $\boldsymbol{n_z}$ with units in meters. The ECEF frame is defined with center at $\boldsymbol{E_0}$ and three orthogonal axes $\boldsymbol{e_x}$, $\boldsymbol{e_y}$, and $\boldsymbol{e_z}$.[4]

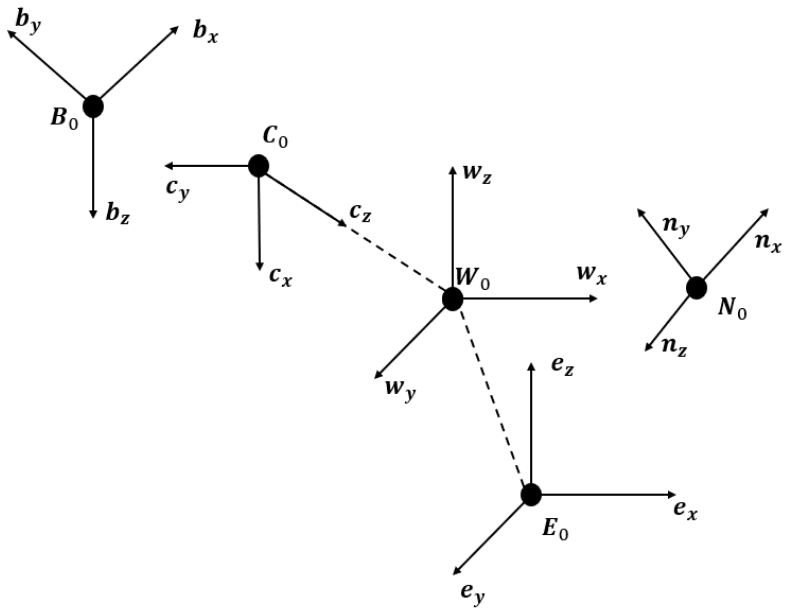


Figure 1.2: NED, Body, Camera, and Point Reference frame.

From Figure 1.2 the position vector for a point about the world frame can be defined as:

$$^{E_0}\boldsymbol{P}^{W} = X_w\boldsymbol{e_x} + Y_w\boldsymbol{e_y} + Z_w\boldsymbol{e_z} \tag{1.1}$$

The unit vector for a point about the camera frame can be defined as:

$$^{C_0}\boldsymbol{P}^{W} = X_c\boldsymbol{c_x} + Y_c\boldsymbol{c_y} + Z_c\boldsymbol{c_z} \tag{1.2}$$

The 3D coordinates of a point (W) in the world coordinate system are defined as X_w, Y_w, and Z_w and they represent the scalar position of our point. Camera models are fundamental to vision-based navigation systems, as the 3D world is projected onto a 2D image plane. The most common camera model used in understanding the 3D to 2D projection is the pinhole camera model. Figure 1.2 shows the relationship between the 3D and 2D coordinates of a point in the world. Equation 1.2 shows the mathematical relationship between image coordinates

and real-world coordinates. [4] Camera intrinsic parameters are defined as focal length (f_x, f_y) and optical centers (c_x, c_y) which are different than the camera reference frame.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} \tag{1.3}$$

In practice, camera lenses introduce distortion based on their shape, causing straight lines in the real world to appear curved in images. Predominant distortions include barrel-like and cushion-like distortions (Figure 1.3). Camera calibration processes measure and correct these distortions, using patterns of known dimensions. Radial distortion, caused by the lens shape, and tangential distortion, due to assembly imperfections, are the two main types of distortion. [4]

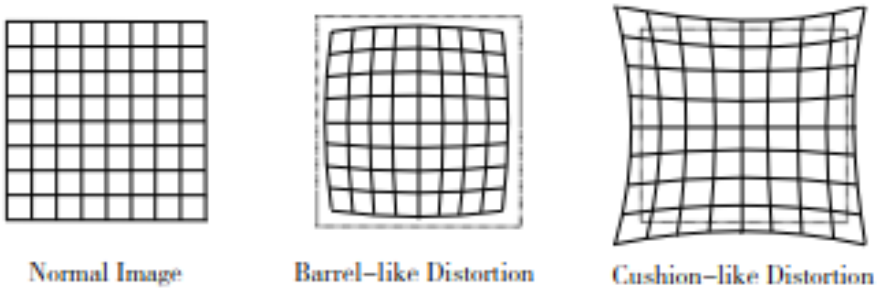


Figure 1.3: Camera distortions on an image. [4]

Using extrinsic parameters, rotation (R), and translation (t), a point in the world coordinate system can be transformed into the camera coordinate system, as shown in Equation 1.2. Extrinsic parameters ${}^{C}R^{W}$ and ${}^{C}t^{W}$ are found through the camera calibration process. [4]

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} = {}^{C}R^{W} \begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} + {}^{C}t^{W} \tag{1.4}$$

Where ${}^{C}R^{W} = \begin{pmatrix} r_{11} & r_{12} & r_{12} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$ and ${}^{C}t^{W} = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$ [4]

The foundational paper on Visual Odometry presented to IEEE in 2004 outlined methods for estimating camera motion: feature extraction, feature matching, and robust estimation. The original method compares a ground truth Inertial Navigation System (INS) and GPS to the visual odometry system mounted on a ground robot. The results were that the visual odometry system

accurately reproduced trajectory over hundreds of meters traversed with varying lighting conditions using only stereo cameras. [3] In more recent works, additional sensors such as Inertial Measurement Units (IMU) are integrated for additional information regarding the camera's pose (orientation?) or correction to the trajectory. The extension of the work is Visual Inertial Odometry (VIO) which leverages the strength of camera and IMU sensors to provide more robustness to VO systems. VO features limitations from lighting conditions, feature sparsity, and scale ambiguity. VIO systems use the IMU to provide resilience to lighting change during visual tracking, provide inertial data for fast movements, and aid in scale estimation by providing data for acceleration and rotation of the system. While VIO offers improvements over VO through the correction provided through inertial data, the visual pipeline is similar between the two systems. However, depending on the specific scenario or system requirements, VO may be more suited for the use case despite the improved performance of VIO.

Feature extraction involves identifying distinctive points or landmarks in the video frames that can be robustly tracked over time. In computer vision, features are pieces of information about an image's content. Common features are points, edges, or objects, and are represented by pixel coordinates of an image as defined by x and y points of a pixel's location in an image. These features serve as the basis for computing the camera's motion and generating a spatial map of the environment. Common methods for feature extraction include corner detectors (e.g., Harris corner detector, Shi-Tomasi corner detector) and blob detectors (e.g., Scale-Invariant Feature Transform - SIFT, Speeded Up Robust Features - SURF). [5] A component of feature extraction is rotation-invariant features, which are features that can still be detected even if the image is rotated. Harris corner detection is an example of a rotation-invariant feature extraction method. Another component of feature extraction is scale-invariant features which do not change as an image is scaled up or down. [6]

Feature matching is the process where a computer matches corresponding features between frames. During feature matching the coordinates of pixels may shift between image frames, but the relative distance between points in a feature remains similar. This process ensures that the same physical point in the environment is consistently tracked across different viewpoints and lighting conditions. Matching techniques often employ descriptors (e.g., SIFT descriptors, ORB descriptors) to establish correspondence between features, optimizing robustness and accuracy. [5]

Motion estimation computes the relative movement of the camera between frames based on the matched feature points. Various algorithms, such as optical flow methods, iterative closest point (ICP) algorithms, or direct methods like Lucas-Kanade, estimate the camera's translation and rotation parameters. [3] These parameters are crucial for updating the camera's position and orientation in real time. Motion estimation can be used to calculate a pose estimation of the camera where the camera's position and orientation are computed over time.

In aerospace applications, the environments being tested are usually dynamic and adverse conditions (varied lighting, low features, etc.). These dynamic environments often induce motion blur, lens glare, and noise into camera data, causing many VO algorithms to be unable to track features. A method using offline 3D reconstructed environment maps and comparing images from an aerial vehicle found the most success in feature tracking through edge alignment. [7] While the algorithm found success with feature alignment in indoor environments where the lighting

conditions were near constant, the feature alignment failed because of lighting variations between frames in outdoor environments.

NASA's Terrain Relative Navigation (TRN) takes onboard images from a vehicle descending and compares the images to a map generated before flight to extract the spacecraft's altitude and attitude with respect to the map. The pre-generated map was built from a satellite orbiting Mars taking photos of the landing site's surface before the mission. TRN has successfully demonstrated the fusion of IMU, Light Detecting And Ranging (LiDAR), and camera sensors for visual odometry in the Mars 2020 mission by landing in the Jezero Crater. [8] The TRN system expands past a navigation system and additionally provides the spacecraft with collision avoidance systems and elevation mapping systems. The success of the Mars 2020 mission demonstrates that using an onboard map with some inherent errors can be performed online during a mission with robustness to imperfections in the system.

A similar approach to the NASA TRN is the AGL-Net approach which compares satellite imagery to a map generated by LiDAR point clouds. AGL-Net provides a solution to two problems: comparing image data to LiDAR data and comparing scale discrepancies between satellite and ground view data. AGL-Net first takes an initial pass of feature matching between the satellite images and the LiDAR data and then uses a Convolutional Neural Network (CNN) to learn scale-invariant features for tracking. CNNs learn hierarchical features from raw input images, which is crucial for understanding and interpreting complex visual patterns. This ability to learn multi-level features, ranging from simple edges to complex textures and objects, makes CNNs highly effective for tasks like object recognition, detection, and segmentation. CNNs use convolutional layers to apply filters across the image which enables the neural network to detect features whether they are scale-invariant, rotation-invariant, or if the location of the feature shifts. A novel result of the AGL-Net during testing was the elimination of the need to pre-process the satellite imagery maps ahead of time. Eliminating the pre-processing was achieved through a CNN allowing the algorithm to operate fully online. The result is substantial for aerospace applications, particularly for real-time testing where the system can handle dynamic environments. [9] While the application of a CNN increases the robustness of the system to environmental conditions the amount of data required, and computational complexity may not be viable to small aerial vehicle applications or in real time if the network is not trained for the scenario.

To avoid the scaling issues between satellite and aerial frames, researchers use a combination of aerial and ground vehicles to perform feature matching due to the smaller distance between the reference frames. Researchers at the Autonomous Systems Lab in Zurich found that many VO algorithms used for localization experience drift from ground truth over longer trajectories. [10] Researchers resolved this approach by using VIO to help reduce sensor drift. The experimental setup uses an aerial drone and a differential drive rover which map and attempt to localize themselves in a local environment. Researchers found that raw VIO still experienced drift from ground truth and corrected this error using localization against previously built maps. Using the hybrid VIO and map-localized system, the aerial vehicle and ground rover localized themselves, allowing the drone to land on the rover. The results show success in the drone and the rover locating each other despite different viewpoints of the same environment. While results demonstrate the ability to localize separate vehicles in a local environment, the results also show sensitivity to misalignment of the local and reference maps to the global alignment.

A similar approach from the Robotics and Perception Group (University of Zurich) focuses on globally localizing and tracking the pose of an aerial vehicle in low-altitude, GPS-denied

environments. The approach uses an air-ground image-matching algorithm that compares images from the aerial vehicle to images on the ground vehicle in a three-dimensional city model. Correction to trajectory drift of the system occurs when the aerial vehicle and ground vehicle images match with sufficient margin. The system is tested on a 2-kilometer trajectory of the drone and demonstrates global localization and position tracking using a single onboard camera on the drone. [11] Researchers found that brute-force visual-search algorithms were required to compensate for illumination, lens distortion, viewpoint deviations, and over-season vegetation variation. The algorithm was successful at globally localizing the drone within the pre-built model of the environment and used uncertainty estimations to compensate for the model. Results of the algorithm's performance show that the vision-based approach outperforms satellite-based GPS at low altitudes and in a city environment, but the system requires high levels of computational complexity. Researchers suggest using cloud-based computing as a viable alternative for the computation complexity of the algorithm but demonstrate a viable alternative to GPS localization for urban environments. [11]

Previous research focused on using prebuilt environment maps with various viewpoints to perform localization. While these methods are useful in off-line work, prebuilt maps impose restrictions on an online system that may exceed the bounds of the previously built maps. Researchers from the Autonomous Systems Lab in Zurich propose a Visual Simultaneous Localization And Mapping algorithm (VSLAM) using a singular monocular camera mounted to an aerial vehicle. Simultaneous Localization and Mapping is a method that uses onboard sensors to build a map of the environment surrounding a robot and localize the robot with respect to the map. The VSLAM algorithm tracks the pose of the camera and simultaneously builds a map of the surrounding area. Results of the approach show that the aerial vehicle can navigate through unexplored environments without the aid of GPS or beacons. The system also shows stability against scale and orientation drift within the map that was built from the VSLAM algorithm and is resilient to wind or similar external disturbances. Limitations to the approach feature sparse environments which cause the vehicle to disable localization and the algorithm’s ability to only localize locally not globally. The online approach demonstrated no time drift and accuracy up to 2-4 cm (about 1.57 in) using the vision-based system. [12]

Researchers at the University of Turku (Finland) approach propose a Global Navigation Satellite System (GNSS) free, vision-based approach to navigation for non-urban environments. Many of the GNSS/GPS vision-based methods mentioned above focus on low-altitude and urban environments, but struggle in feature-sparse environments. Researchers demonstrated a non-traditional VO algorithm that is optimized for long-range, high-altitude UAV flights. [13] The method again uses a map generated before flight and uploaded to the drone for online use. The uploaded map features multiple labeled latitude and longitude coordinates that aid the drone in its navigation across the trajectory. The algorithm uses the following features to localize itself: segmenting geo-referenced satellite images, feature matching (UAV to satellite images), rotation adjustment, and perspective adjustment. The paper presents significant advancements in UAV localization by accounting for perspective and rotation misalignments between the UAV and satellite frames which improves feature matching.

In conclusion, navigating through GPS-denied environments in aerospace applications commonly involves using a pre-developed map and performing online feature matching. Common failure modes for these methods include variations in lighting conditions, feature sparsity, and perspective distortions from different reference frames. Research indicates that there is no single

method suitable for all flight operations; some methods are more successful at low altitudes, while others perform better at high altitudes. Overall, current research shows that there are several viable approaches for using visual odometry as a navigation method in GPS-denied environments for aerial vehicles.

1.3 Project Proposal

The goal of this project is to design and implement a VO algorithm capable of navigating an Unmanned Aerial Vehicle (UAV) without GPS data. The VO algorithm results will be compared against GPS as ground truth data to evaluate the approach's accuracy. From the above literature review, the main problem to be solved by the VO algorithm will be dynamic scaling and lighting conditions that cause many VO algorithms to fail or be limited to use indoors. The project will be split into two parts. The first part will be an evaluation of the VO algorithm in simulation to verify the approach methodology. The second will be the implementation of a low-cost multirotor to evaluate the simulation-to-real-world gap. The simulation will feature ideal scenarios for lighting, camera distortion, and overall system noise. The low-cost drone can be seen in Figure 3.0 which will serve as the real-world hardware the algorithm will be tested on. The algorithm will use a pre-generated map of an environment composed of satellite images and be responsible for localizing itself within the environment using a single monocular camera.

Figure 1.4: Tarot FY690S Hexacopter. [14]

1.4 Methodology

The proposed methodology is to use a downward-facing camera and IMU sensor mounted to a drone to calculate the relative pose of the camera with respect to the drone and world-centered frame. Before flight, a map of satellite images will be stitched together from a set of user-defined waypoints. Then in flight, rotate the camera images of the drone to the world-centered frame to compare the images from the drone to satellite images of the prebuilt map. The next steps will be feature extraction, feature matching, and motion estimation. Feature extraction will be responsible

for finding distinguishable features from the onboard camera. Feature matching will then compare the onboard camera data to the prebuilt satellite image map. Then motion estimation will be calculated to determine the relative motion of the drone/camera. The above methodology would provide a visual odometry algorithm that is fully self-reliant and does not require any GPS correction for waypoint-based navigation. Some of the main resources used would be OpenCV for handling the computer vision calibrations and image transformations, Tensorflow and Keras for the Machine Learning/Deep Learning algorithms, and Google Maps for the satellite imagery. The drone will use a PID (Proportional, Integral, Derivative) controller for its velocity and position controllers. PID controllers were selected for their ease of implementation, tuning, and lower computational complexity (when compared to other modern control techniques).

2. Computer Vision

2.1 Pinhole Camera Models

Continuing the camera model discussed in Chapter 1, a 3D point *X* is projected through the camera's optical center onto an image plane and produces image point *p*. Figure 2.1 shows the perspective projection (a), catadioptric projection (b), and spherical model (c) with image points. Chapter 2 will focus on using the perspective projection model for all of the calculations.

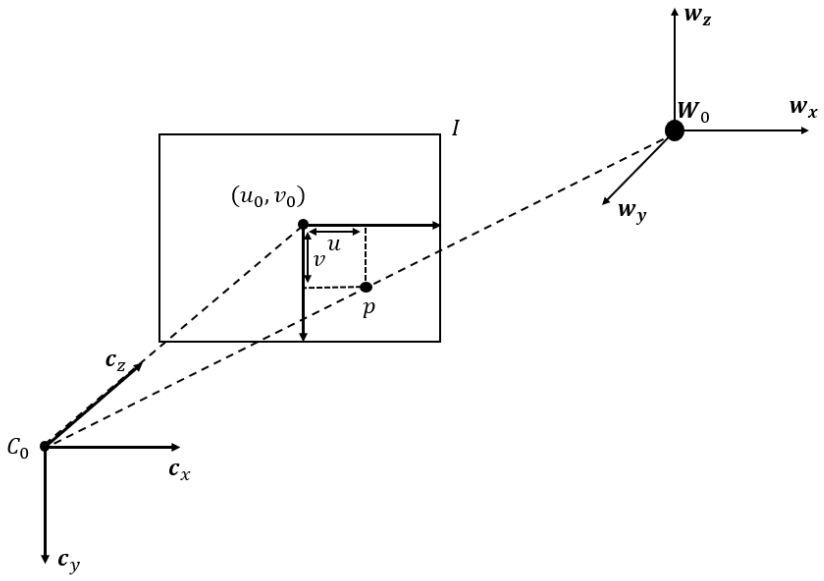


Figure 2.1: Camera projection model for perspective projection for a point in the real world.

The coordinates of X can be defined as $[X_c, Y_c, Z_c]^T$, p is $[p_x, p_y, p_z]^T$ and the physical distance between the imaging plane and the camera plane is f (focal length). Using the classical camera model an image will be inverted from projection as signified by the negative sign:

$$\frac{Z_c}{f} = -\frac{X_c}{p_x} = -\frac{Y_c}{p_y} \tag{2.1}$$

Modern cameras do not invert the image so the formula can be rewritten as:

$$\frac{Z_c}{f} = \frac{X_c}{p_x} = \frac{Y_c}{p_y} \tag{2.2}$$

The projected coordinates on the image plane can be calculated as:

$$p_x = f\frac{X_c}{Z_c} \tag{2.3}$$

$$p_y = f\frac{Y_c}{Z_c} \tag{2.4}$$

Camera sensors convert light from real-world objects into electrical signals which can be reconstructed into image pixels. The pixel plane is defined as $o - u - v$, if the pixel plane is fixed on the physical imaging frame, then the pixel coordinates of p can be defined as: $[u, v]^T$ Then using the intrinsic camera parameters, the pixel coordinates can be calculated as:

$$u = f_x\frac{X_c}{Z_c} + c_x \tag{2.5}$$

$$v = f_y\frac{Y_c}{Z_c} + c_y \tag{2.6}$$

Where camera intrinsic parameters are defined as focal length (f_x,f_y) and optical centers (c_x, c_y) which can be calculated from the camera calibration process. [4]

2.2 Camera Calibration

A limitation to the pinhole camera model is the distortions caused by projecting the 3D world onto a 2D plane when a camera lens is used. The two main distortion categories are barrel-like distortion and cushion-like distortion. In barrel distortion, the radius of pixels decreases as the optical axis distance increases. Cushion-like distortion is when the radius of the pixels increases as the optical axis decreases. The shape of the lens introduces radial distortion while the assembly

and geometry of the camera introduce tangential distortion. Tangential distortion is predominately caused by the sensor plane not being perfectly parallel to the lens. Radial distortion can be represented as:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad [16] (2.7)$$

$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad [16] (2.8)$$

Where *r* is the distance from the image center. Tangential distortion can be represented as:

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \quad [16] (2.9)$$

$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \quad [16] (2.10)$$

The process of camera calibration aims to solve the five unknown distortion coefficients: $(k_1, k_2, k_3, p_1, p_2)$. A common calibration method uses corner detection with a chessboard with known lengths of each of the sides of the marks and a known number of marks on the board as seen in Figure 2.1. Additionally, intrinsic parameters of the camera need to be solved for such as focal length (f_x, f_y) and optical centers (c_x, c_y). An example of distortion before calibration shown in Figure 2.2 shows the radial bowing of the checkerboard past the red lines (the red lines represent ground truth). [5]

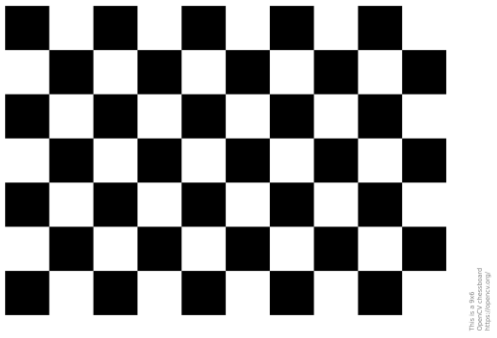


Figure 2.2: Chessboard for calibration. [16]

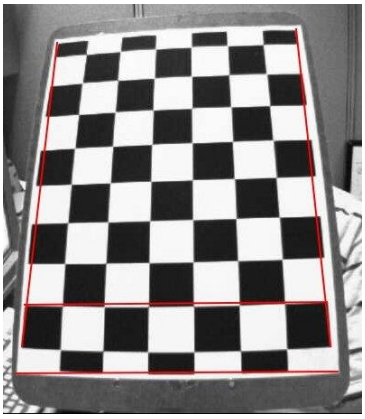


Figure 2.3: Chessboard with distortion against ground truth (red lines). [16]

2.3 Fundamental Matrix

The fundamental matrix *F* is a 3x3 matrix that relates corresponding points between two images taken from different viewpoints. It encapsulates the intrinsic and extrinsic parameters of the camera system as saved using camera calibration. Given a point p_1 in the first image and a point p_2 in the second image, the fundamental matrix satisfies the equation given in Equation 2.11.

$$p_2^T F p_1 = 0 \qquad \text{[4] (2.11)}$$

Where $p_1 = (x_1, y_1, 1)$ and $p_2 = (x_2, y_2, 1)$. The fundamental matrix can be decomposed in the camera's intrinsic parameters (*K)* and the extrinsic parameters of rotation (*R*) and translation (*t*).

$$F = K_2^{-T}[\,{}^{c}t^{W}]\,{}^{c}R^{W}K_1^{-1} \qquad \text{[4] (2.12)}$$

Where:

$$\hat{t_k} = \begin{bmatrix} 0 & -X_W & Y_W \\ Z_W & 0 & -X_W \\ -Y_W & X_W & 0 \end{bmatrix} \qquad (2.13)$$

Given a set of corresponding points, the fundamental matrix can be estimated. Once *F* is obtained constraints for matching points between two images are provided. [4]

2.4 Essential Matrix

The geometric relationship between two images using a calibrated monocular camera can be described with the essential matrix *E*. *E* contains the camera's motion parameters for some

unknown scaling factor (in 2-D to 2-D motion). The essential matrix can be defined by equation 2.15, where ${}^{C}R^{W}$ is the rotation matrix.

$$E_k \simeq \hat{t_k}\, {}^{C}R^{W} \qquad \text{[4] (2.14)}$$

Both rotation and translation of the camera can be directly extracted from *E,* extracting the rotation and translation of the camera serve as the foundation of motion estimation for visual odometry. 2-D to 2-D motion-based estimation is founded on the epipolar constraint between two images which defines the line on which a feature lies between image frames where p and p' are normalized coordinates of corresponding points between two images. The epipolar constraint can be formulated by:

$$p'^{T} E p = 0 \qquad \text{[4] (2.15)}$$

In stereo vision, epipolar geometry is used to describe the geometric relationship between two images from different perspectives. Using a monocular camera, epipolar geometry can be used to calculate the position and rotation between image frames. The point where the optical axis of the first and second camera pose intersects along the image plane is the epipole. The epipolar line is the projection of a 3D point onto the image plane of each camera pose. By leveraging the geometric relationships between the epipole, epipolar line, and fundamental matrix, pose estimation between image frames can be achieved through feature matching. The epipolar constraint between two images is a point on the one image must lie on the epipolar line of the second image which is derived from the essential matrix. Figure 2.3 shows the epipolar constraint between two images. [4][15]

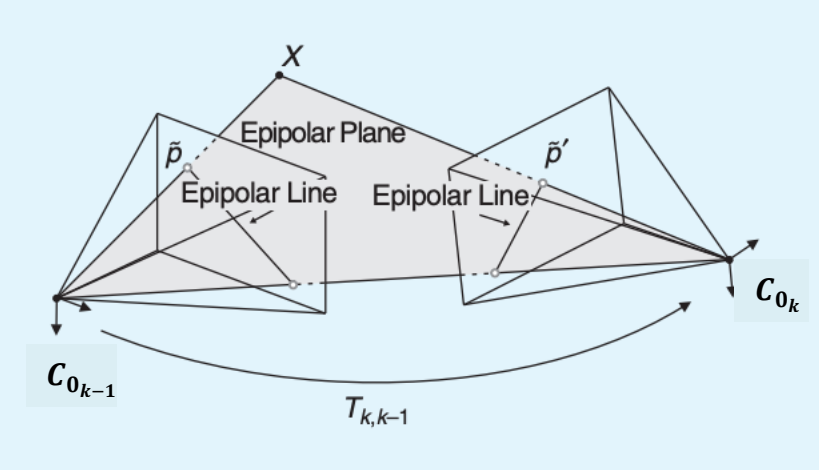


Figure 2.4: Epipolar constraint between two images. [15]

2.5 Image Filtering

Image filtering plays a crucial role in enhancing the visual data present in an image. Image filtering uses a variety of techniques such as adding or removing noise from an image to manipulate the pixel values. Image filtering is commonly used to either enhance edges and either extract or remove features from an image. The process involves applying a kernel over an image to produce a filtered output. Common filters are Gaussian filters, median filters (noise reduction), and Sobel filters (edge detection). Equation 2.17 shows the Gaussian function which can be used to add or remove noise and Equation 2.18 shows the operation for a median filter. [4]

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad \text{[4] (2.16)}$$

$$I'(x,y) = median\{I(i,j)\} \quad \text{[4] (2.17)}$$

The process of applying a Sobel filter uses a calculation of the gradient of the image intensity. The Sobel operator uses two 3x3 kernels for detecting horizontal (G_x) and vertical edges (G_y):

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 2 & 1 \end{bmatrix} \quad (2.18)$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (2.29)$$

Then the gradient magnitude is computed in Equation 2.21:

$$G = \sqrt{G_x^2 + G_y^2} \quad (2.20)$$

Then the direction of the gradient can be computed in Equation 2.22:

$$\theta = arctan(\frac{G_y}{G_x}) \tag{2.21}$$

2.6 Feature Matching

In the early 2000's researchers predominately relied on corners, edges, and blocks for feature detection. While corners and edges are more distinguishable than just pixels across multiple images, these features are still subject to difficulties, when the camera is in motion. Corners' and edges' appearance change when a camera rotates and tracking the same points becomes rather difficult. Figure 2.5 depicts representative places in an image such as corners, edges, and blocks of pixels. Common features used in modern feature matching algorithms are Scale-Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF), and ORB (Oriented FAST and Rotated Brief). Researchers evaluate SIFT, SUFR, and ORB based on the following criteria: repeatability, distinctiveness, efficiency, and locality. The combination of the previously mentioned criteria allows for features to be found in multiple images with different scales, locations, and even expressions (color or size due to lighting conditions). [4]

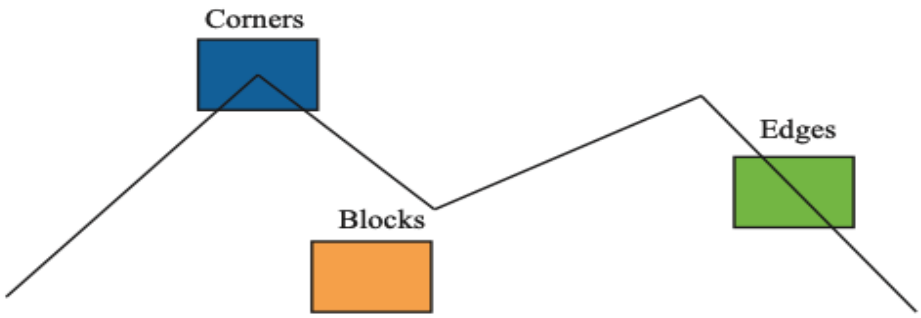


Figure 2.5: Corners, edges, and blocks represented as unique features in an image. [4]

A feature point is composed of key points and descriptors. The key points refer to the 2D position of a feature in an image. Some key points contain other information like orientation and size. The descriptor is a vector describing the pixels around the key point. If two feature descriptors are relatively close in the vector space, then they are considered the same feature. The choice to use one of the above three methods comes to computation power and real-time performance. SIFT and SURF require longer calculation times but usually provide more robust features to be detected during varied lighting conditions, rotation, and translation. ORB and SIFT can be used for real-

time feature extraction by using FAST detection for key points which makes the feature extraction process practical for real-time SLAM and navigation applications.

ORB features are comprised of ORB key points and ORB descriptors. Extraction of ORB features requires a FAST corner point extraction which finds the corner point in the image. Then the BRIEF descriptor is generated which describes the surrounding area of the feature points described in the previous step. The FAST corner point extraction mainly detects changes in the local grayscale. FAST functions by comparing how bright pixels are from neighboring pixels, and if the change is large then the point is likely a corner. The following steps are taken to extract a FAST point which can be visualized in Figure 2.6:

- Select a pixel *p* in the image, assume brightness as I_p
- Set a threshold *T* for filtering
- Take pixel *p* and select 16 pixels with a radius of 3 pixels surrounding point *p*
- If consecutive N points are greater than $I_p + T$ or less than $I_p - T$, then the central point is a feature point (common N values are 9,11, and 12)
- Iterate through each of the above steps for every pixel in an image

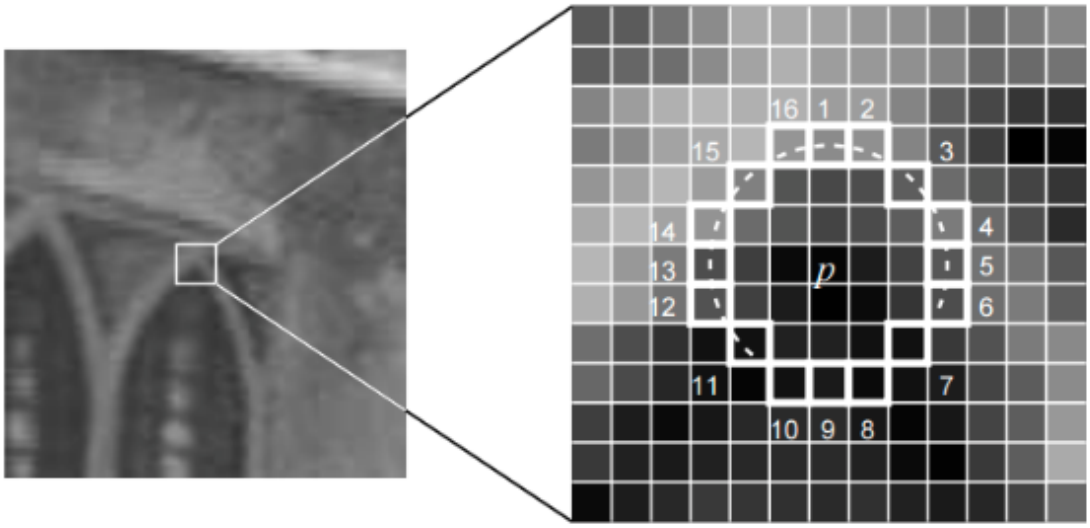


Figure 2.6: FAST key points on an image from some radius p. [16]

Common descriptions in FAST corner points are scale and rotation which aid in the detection of points between consecutive images, these become known as Oriented FAST key points. After extracting the Oriented FAST key point the binary descriptor BRIEF is calculated. A BRIEF descriptor is represented as a 0 or a 1. BRIEF is 1 when a second pixel in a pair (p, q) is greater than the first, otherwise, the descriptor is repressed as 0. Using the directional information provided from the FAST feature point orientation becomes encoded in the BRIEF descriptor. Feature

matching compares the descriptors from different images to find pairs of matching features. Feature matching relied predominantly on Euclidean distance for SIFT and SURF, and Hamming distance for ORB. Equation 2.23 shows the calculation of Euclidean distance in 2-D and Equation 2.24 shows the calculation of Hamming distance. [4]

$$d = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2.22}$$

$$d(a, b) = \sum_{i=1}^{n} (a_i \oplus b_i) \tag{2.23}$$

2.7 Homography

Using the epipolar constraint alongside the fundamental matrix and essential matrix, the homography matrix *H* describes the relationship between planes. When feature points lie along the same plane, the homography matrix can be used to estimate motion. This method of motion estimation is particularly common in top-view cameras for robotics applications. The homography matrix captures the transformation between points on a common plane across consecutive images. Using the homography matrix, essential matrix, and fundamental matrix points between consecutive images with changing camera poses, the motion estimation can be extracted by aligning all the images on the same plane. Considering a pair of matched feature points p_1 and p_2 in images I_1 and I_2, assume all these points lie on a plane P. The plane equation is derived in Equation 2.25 and rearranged to Equation 2.26. [4]

$$n^T P + d = 0 \tag{2.24}$$

$$n^T P = -d \tag{2.25}$$

Then using the equation relating a matched point to the camera intrinsics and the plane in Equations 2.27 and 2.28 we can then derive the homography matrix in Equation 2.29. [4]

$$p_2 \simeq K({}^{C}R^{W}P + t_k(-\frac{d}{n^T P}) \quad \text{[4] (2.26)}$$

$$p_2 \simeq K({}^{C}R^{W} - \frac{t_k n^T}{d})P \quad \text{[4] (2.27)}$$

$$H \simeq K({}^{C}R^{W} - \frac{t_k n^T}{d})K^{-1} \quad \text{[4] (2.28)}$$

The transformation between p_1 and p_2 can then be expressed in Equation 2.30 and then expanded to the form in Equation 2.31.[4]

$$p_2 \simeq Hp_1 \quad \text{[4] (2.29)}$$

$$\begin{pmatrix} u_2 \\ v_2 \\ 1 \end{pmatrix} \simeq \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix} \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} \quad \text{[4] (2.30)}$$

To estimate the homography matrix at least four pairs of matched points are required. This method is known as Direct Linear Transform (DLT), sometimes referred to as the Direct Method, which treats *H* as a vector and solves a series of linear equations. After obtaining the estimation for *H* the matrix can be decomposed into solutions for *R* and *t.* Any inconsistencies require additional constraints to be added or information regarding the camera's state to correct the solution. The series of linear equations to solve for *H* can be seen in Equation 2.32. [4]

$$\begin{pmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1u_2 & -v_1u_2 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -u_1v_2 & -v_1v_2 \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \end{pmatrix} = \begin{pmatrix} u_2 \\ v_2 \end{pmatrix} \quad [4] (2.31)$$

Once the homography matrix is decomposed the Direct Method has been solved for the rotation and translation of the camera between frames giving a motion estimation. Typically, the Direct Method will suffer from error accumulation over long durations if left uncorrected or if sparse features are present. [4] To develop a vision-based odometry system for aerial applications visual odometry will be used to extract and track the heading of the aircraft over time. Elements such as feature extraction will be used for feature tracking in developing the navigation functionalities of the algorithm. [4]

3. Computer Vision-Based Navigation

3.1 Extracting Rotation and Translation

After solving the homography matrix H, the matrix can be decomposed into solutions for R and t. [4] The decomposition process typically involves singular value decomposition (SVD) and can be represented as:

$$H \simeq K({}^{c}R^{w} - \frac{t_k n^T}{d})K^{-1} \quad [4] \ (3.1)$$

After solving H for R and t the rotation and translation of the aerial vehicle can be tracked locally. Once we have the rotation matrix R, we can convert it to rotation quaternions which is another way to represent rotations about three axes. Another common alternative to quaternions is Euler angles and is often seen in aerospace applications to describe the orientation of an aircraft. However, in instances where the pitch angle is $+90^{\circ}$ or -90° the functions representing roll and yaw become undefined inducing a "gimbal lock" where there is no unique solution. To avoid this problem, quaternion representations of rotation prevent any such instance from being unable to compute a unique solution. Equations 3.1 - 3.4 represent the calculations of the magnitude of each of the quaternion components.

$$|q_0| = \sqrt{\frac{1 + r_{11} + r_{22} + r_{33}}{4}} \quad (3.2)$$

$$|q_1| = \sqrt{\frac{1 + r_{11} - r_{22} - r_{33}}{4}} \tag{3.2}$$

$$|q_2| = \sqrt{\frac{1 - r_{11} + r_{22} - r_{33}}{4}} \tag{3.3}$$

$$|q_3| = \sqrt{\frac{1 - r_{11} - r_{22} + r_{33}}{4}} \tag{3.4}$$

After computing the magnitude find the largest value of q_0, q_1, q_2, q_3. [18] Then compute the remaining components following the bellow conditions:

- If q_0 if the largest:

$$q_1 = \frac{r_{32} - r_{23}}{4q_0} \tag{3.5}$$

$$q_2 = \frac{r_{13} - r_{31}}{4q_0} \tag{3.6}$$

$$q_3 = \frac{r_{21} - r_{12}}{4q_0} \tag{3.7}$$

- If q_1 if the largest:

$$q_0 = \frac{r_{32} - r_{23}}{4q_1} \tag{3.8}$$

$$q_2 = \frac{r_{12} - r_{21}}{4q_1} \tag{3.9}$$

$$q_3 = \frac{r_{13} - r_{31}}{4q_1} \tag{3.10}$$

- If q_2 if the largest:

$$q_0 = \frac{r_{12} - r_{31}}{4q_2} \tag{3.11}$$

$$q_1 = \frac{r_{12} - r_{21}}{4q_2} \tag{3.12}$$

$$q_3 = \frac{r_{23} - r_{32}}{4q_2} \tag{3.13}$$

- If q_3 if the largest:

$$q_0 = \frac{r_{21} - r_{12}}{4q_3} \tag{3.14}$$

$$q_1 = \frac{r_{13} - r_{31}}{4q_3} \tag{3.15}$$

$$q_3 = \frac{r_{23} - r_{32}}{4q_3} \tag{3.16}$$

The rotation quaternion can then be represented as $q = [q_0 \quad q_1 \quad q_2 \quad q_3]^T$ and tracked over time. A similar example in aerospace is dead reckoning. Dead reckoning uses onboard sensors to measure the rates of change of the aircraft, and then by integrating, the aircraft's rotation and translation can be calculated over time. Dead reckoning and visual odometry are both methods to track a vehicle's position without celestial measurements (GPS, GNSS, etc). However, these methods suffer from accuracy over time due to sensor errors and often require corrections depending on accuracy needs. The local tracking using visual odometry is an example of SLAM for our vehicle where we are independent of any prior state data. [18]

3.2 Satellite Image Map Preparation

To provide the vehicle with a global reference, a set of stitched satellite images will be used to generate a 2D map of the terrain. The satellite image will provide a set of known reference points along the trajectory that can be compared against the drone's onboard camera for feature matching. The feature matching allows the drone to localize itself within the reference map. Some important assumptions that play a role in generating and utilizing the map are as follows:

- Neglecting Earth's rotation
- Neglecting the curvature of the Earth
- Short flight time (less than 1 hour)
- The desired flight path is known ahead of time

These assumptions enable the use of the stitched satellite image map without the need to compensate for rotating reference frames and the obliqueness of the Earth. In particular, the known flight path assumption plays a crucial role. By knowing the flight path in advance, a pre-downloaded map can be generated using GPS coordinates and then provided to the algorithm. The map can then be used for real-time operations to provide a global reference to the vehicle. The limitation to the predetermined path assumption is if the vehicle deviates a large amount from the original flight path, the algorithm may not have information about the given location of the vehicle.

The OpenCV library has a built-in stitching function that takes a list of images then performs feature matching to overlay the images onto one another then crops the overlaid portion

of the images. The flowchart seen in Figure 3.1 shows the steps taken by the OpenCV stitcher class to overlay and crop the images.

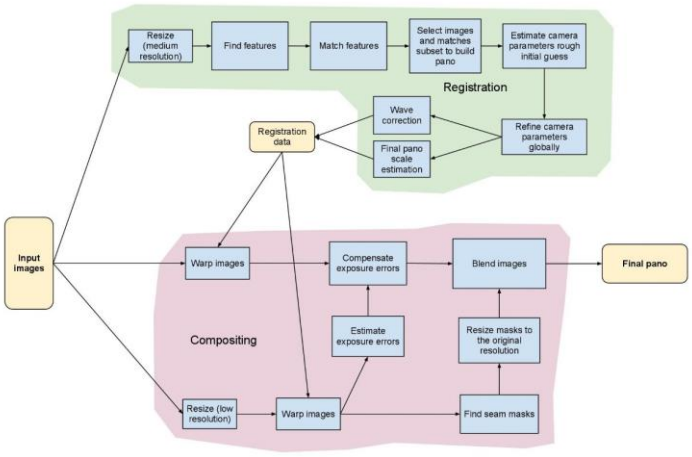


Figure 3.1: Image Stitching pipeline flowchart using OpenCV Stitcher Class [17].

De-rotate and flatten all satellite images to a stitched map. Combine satellite images with GPS coordinates on the stitched map. Additionally, reference stitched maps as storage. Figure 3.2 shows a set of sample images from Google Earth in the “layers” view that will be stitched together using OpenCV. Figure 3.3 shows the set of sample images stitched together.

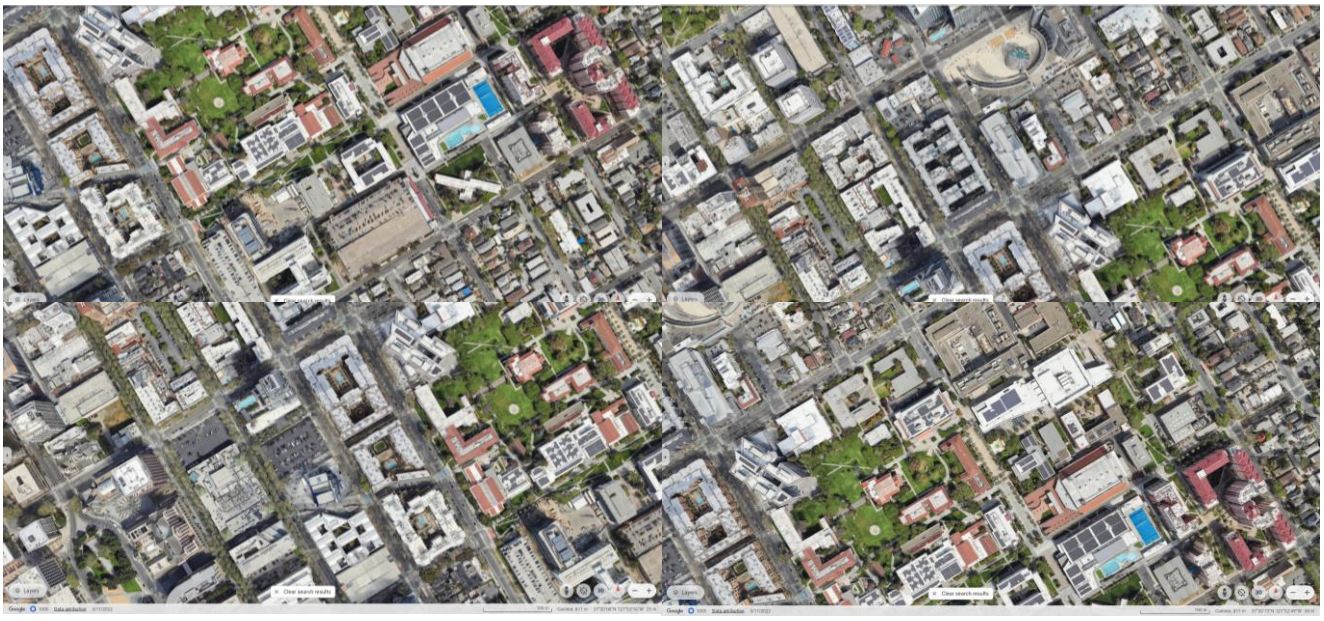


Figure 3.2: Sample images to be stitched together.

Figure 3.3: Final output of stitched images from Figure 3.2.

3.3 Visual Waypoint Integration

Given the assumptions specified in Section 3.2, a global map of the flight path is known. Using the map, a set of waypoints can be provided to the algorithm and compared to the onboard camera stream. Visual waypoints are defined as distinctive features or landmarks on the preloaded map. A waypoint used for matching will consist of some number of feature descriptors and the 2D position of the waypoint in the map.

Figure 3.4: Visual waypoint with features and waypoint coordinates.

In traditional GPS-waypoint-based navigation, the position of the vehicle is calculated using the process of ranging. Ranging determines the position and velocity of a receiver (the GPS sensor) and measures the distance from several radio sources. To calculate the position of some receivers using ranging, the following assumptions must be made:

- A receiver is on Earth and can receive some number of *N* radio signals from satellites
- The coordinates of the position of the satellite p_N are known
- The precise time instances t_i for each of the broadcasted signals are known, where *i* is the number of the satellite
- The precise time instances t_r of the received signal are known
- The distance r_i from the receiver to each of the satellites can be calculated

$$r_i = c(t_r - t_i) \tag{3.18}$$

Where *c* is the speed of light in vacuum $c = 2.998x10^8 m/s$. Then given that the position of the satellite (p_i) is known the position of the receiver (p_r) can be calculated using Equation 3.19.

$$||p_r - p_i|| = \sqrt{(p_{x_r} - p_{x_i})^2 + (p_{y_r} - p_{y_i})^2 + (p_{z_r} - p_{z_i})^2} \tag{3.19}$$

The process of navigating through a set of waypoints *W* requires a comparison between the features in the map and the features in the onboard camera. The feature comparison will require

that some number of features between the stored (map) and observed features are matched. A tolerance around the waypoint location will be required, otherwise the problem will not be bounded. Therefore, a tolerance of radius *R* will be applied to the features around the waypoint such that the number of features in the region of interest is *N*. Then if the number of matched features from the feature matching is equal to *N* it can be determined that the vehicle has arrived at the waypoint. The algorithm will then iterate through the series of waypoints until it has reached its goal.

3.4 Environmental Factors

Visual odometry systems often face challenges in real-world scenarios due to dynamic objects in the environment and varying lighting conditions. Factors such as moving objects, dynamic lighting conditions, and induced vibrations can cause large enough variations between frames to disrupt feature extraction, feature matching, and motion estimations. While hardware such as vibration dampers and onboard lighting may aid in reducing variation between image frames, software estimates may be more capable of handling large variations. Feature descriptors such as BREIF and SIFT are non-intensity based which provides some robustness to lighting variations. Another option may be to use thermal cameras that can operate outside of boundaries that visible band cameras are limited by.

Aside from lighting challenges, scene dynamics such as featureless environments or dynamic objects can interrupt VO system measurements. Low feature environments make it difficult to detect and track features causing homography between image frames to fail or lose its accuracy. Additionally, rapidly changing features such as moving objects, large deviations in the camera's motion, and scale can cause difficulty in feature tracking. These environmental factors can cause interruptions in tracking and cause either a large drift in the accuracy of the measurements or the algorithm to fail.

3.5 Kalman Filter Integration

For correcting our trajectory estimation from section 3.1 applying an Extended Kalman Filter (EKF) will be used for local and global estimation. Due to the non-linear aspects of visual odometry (measurement model and coordinate transformations), an EKF will be required to handle the non-linearities rather than a traditional Kalman Filter (KF). The EKF is more flexible than the KF as it can handle both linear and non-linear systems. The EKF is extended from the original KF using a first-order Taylor series expansion. The expansion limits the EKF by making the function locally linear using a first-order expansion for the approximation, the accuracy of predictions made by the EKF degrades after the first order. For our EKF we have a State Vector x which is comprised of our translation t and rotation R. [19]

$$x = \begin{bmatrix} {}^{c}t^{w} \\ {}^{c}R^{w} \end{bmatrix}^{T} \tag{3.20}$$

The first step in the EKF is the process model or prediction step:

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}) + w_{k-1} \tag{3.21}$$

Where f() is the state transition function, w is the noise in the system, $\hat{x}_k$ is the state estimate, and u_k is the input. Then the measurement function is calculated which relates the state to the expected measurement.

$$z_k = h(\hat{x}_k^-) + v_k \tag{3.22}$$

Where h() is the non-linear measurement function. The next step in EKF is to compute the Jacobians of f() and h() at each step.

$$F_k = \frac{\partial f}{\partial x}|_{\hat{x}_{k-1}, u_{k-1}} \tag{3.23}$$

$$H_k = \frac{\partial h}{\partial x}|_{\hat{x}_k^-} \tag{3.24}$$

Now that the Jacobians for f() and h() are solved the predictions can be calculated:

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}) \tag{3.25}$$

$$P_k^- = F_k P_{k-1} F_k^T + Q_{k-1} \tag{3.26}$$

Then the Kalman gain, and predictions can be updated.

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \tag{3.27}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-)) \tag{3.28}$$

$$P_k = (I - K_k H_k) P_k^- \tag{3.29}$$

After the predictions are updated, the algorithm repeats for the next estimation. [19]

3.6 Extended Kalman Filter Error Correction

Section 3.5 highlights the steps taken by and EKF algorithm, the above steps provide corrections that improve the overall navigation accuracy, by improving the estimations of position and rotation provided by the VO system. The following steps are taken to initialize and apply corrections to the algorithm's measurements using the EKF:

- Set the initial state estimate based on the VO measurement and initialize the covariance matrix with uncertainties
- Compute the filtered state estimation using the EKF at each step
- Compute the difference between the EKF state estimate (x_k) and the raw VO state estimate
- Compute the corrected visual odometry estimate with the weighing factor applied to the EKF estimate

$$\hat{x}_k^{corrected} = \hat{x}_k + \lambda(\hat{x}_{VO} - \hat{x}_k) \tag{3.30}$$

λ is used as a scaling factor which can be used to adjust confidence in the visual odometry estimate. The above steps improve the accuracy of the system by applying a filtered estimate to the existing odometry measurements. If the above corrections are not sufficient for performance, then tuning the process noise covariance and measurement noise covariance would improve the characteristics of the system. The filtered estimate aims to remove noise that was observed by the sensors during measurement processes. The EKF also can use the previous time step estimates to provide an estimate during interruptions in visual tracking/measurements. To further improve the measurements and estimates made by the algorithm additional sensor data can be used for sensor fusion to provide further corrections to the system. [19]

4. Convolutional Neural Network

4.1 Introductions to Convolutional Neural Networks

Convolutional neural networks (CNN) have led to progress in image classification and feature detection in machine vision. CNNs are neural networks that leverage the unique properties of convolution for image and video processing tasks. CNN applications across many industries, including medicine, banking, manufacturing, insurance, transportation, and social media. CNNs fall under a category of Machine Learning (ML) known as Supervised Learning, other common ML methods are Unsupervised Learning and Reinforcement learning. The use of the word "deep" coupled with one of the ML methods implies the use of neural networks to perform calculations. [20]

- Supervised Learning: a model trained on a labeled dataset, where each input has a corresponding target output label. The supervised learning model then learns to map inputs to outputs by minimizing the error between the prediction (output of the model) and the target output.
- Unsupervised Learning: a model trained on a dataset without any labeling and learning underlying patterns or structures.
- Reinforcement Learning: an agent interacts with an environment and receives feedback on the actions taken. The feedback can be either a reward or penalty dictating how good an action was. The agent aims to maximize the cumulative reward (feedback) over time.

4.2 CNN Architecture

AlexNet and ResNet are influential architectures for CNNs that have contributed to image recognition tasks. [20][21] While specific architectures will vary from task to task the order of operations for CNNs will broadly remain the same. Specific deviations include layer sizing, layer depths, activation functions, preprocessing, normalization, and network outputs. Factors that influence architecture are specific tasks, datasets, computational resources, and training time. In general, the following layers are required and shown in Figure 4.1:

- Input Layer
- Convolutional Layer
- Activation Function
- Pooling Layer
- Flatten Layer
- Fully Connected Layer
- Output Layer

[20]

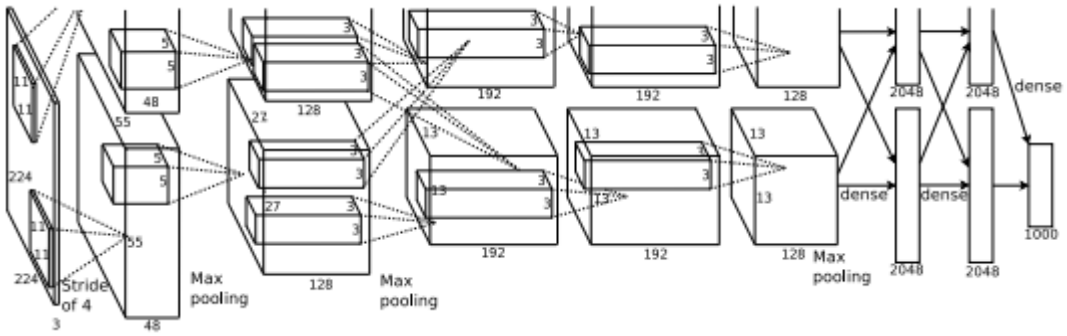


Figure 4.1: Sample CNN Architecture from the AlexNet paper for image classification tasks. [20]

4.3 Input Layer

The input layer is where information is provided to the model, where the number of equal to the number of features in the data being fed into the layer. In the case of the image, the number of neurons would be equal to the number of pixels. A critical constraint for a CNN is that the data must be passed in a grid format. Typically, an input is represented as a 3D tensor with dimensions:

- Height
- Width
- Channels

Where the height and width are represented as pixels and the channel is either represented as RGB or grayscale.

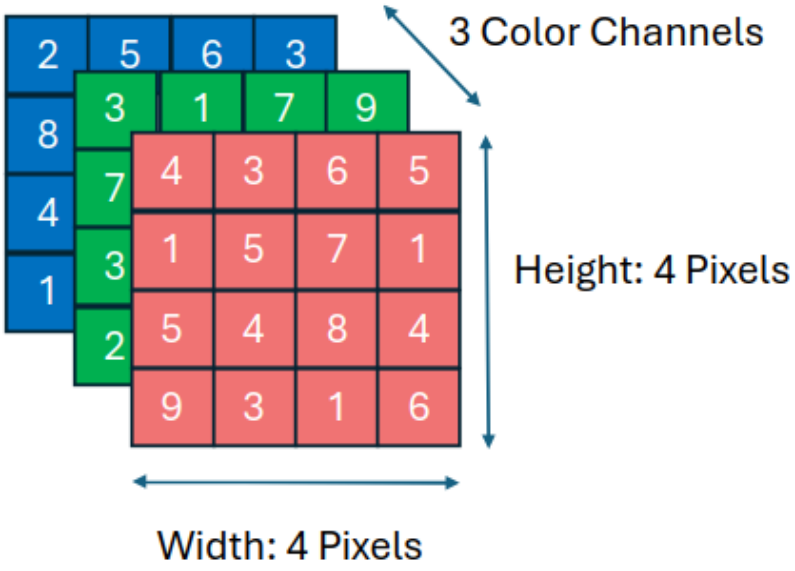


Figure 4.2: Input representation of a 4x4 image with RGB channels can be represented as a 4x4x3 tensor.

Depending on the dataset, if the data is preprocessed then the data may just be passed through, otherwise normalization and augmentation techniques may be applied. Normalization scales the pixel values to be within a set range such that the model can converge faster during training. For normalizing an image in range [0,1] the following equation can be used:

$$I = \frac{v_i - \mu}{\sigma} \tag{4.1}$$

Where v_i is the original pixel value, μ is the average pixel value across the image, σ is the standard deviation of the pixel values. Augmentation can also be done to increase the diversity in training data which may include:

- Rotation
- Cropping
- Translating
- Flipping (horizontal or vertical)

Augmentation and preprocessing steps help ensure that the input dimensions of the data are consistent across the whole dataset and that the data is organized for the following convolutional layers. Depending on the neural network's architecture predefined filters or feature detection algorithms may be applied to the data before passing for convolutional operations in subsequent layers.

4.4 Convolutional Layer

The convolutional layer is the core of what makes CNN ideal for images. It performs the following operations:

- Receiving an input tensor (image or feature map from a previous layer)
- Apply a set of learnable filters (also known as kernels) to the input
- Computes the convolution operation for each position of the kernel
- Produces a feature map from the kernels

The convolutional layer is a connection to a local region of the previous layer where weights are shared across spatial positions. The convolutional layer starts by applying a collection of kernels (or filters) $\boldsymbol{W}_k$ to the input tensor of the image. Each kernel extracts a specific feature such as an edge, corner, texture, or pattern. The kernels are typically applied to partial regions (e.g. 3x3 or 5x5) of the image rather than the whole image. During the application of the kernels, a convolution operation is performed which involves sliding the filter across the input tensor and computing the weights for each section of the input tensor. For each position (*i, j*) of the input tensor the output feature map **Y** is calculated using the equation as found in Equation 4.2. The convolution process results in a feature map that represents the presence of features from the applied kernels.

$$Y(i,j,k) = \sum_{m=0}^{K_h-1} \sum_{n=0}^{K_w-1} X(i \cdot S + m, j \cdot S + n, :) \cdot W_k(m,n,:) + b_k \tag{4.2}$$

Where K_h and K_w are the width and height of the kernel, S is the stride, and b_k is the bias of the kernel. The stride refers to the number of pixels the kernel shifts after each application. A stride of 1 means the kernel moves one pixel at a time, whereas a stride of 2 means the kernel skips every other pixel. For kernels that lie on the edges of an image padding is used to control the spatial dimensions of the output. Figure 4.3 shows the application of a 3x3 kernel when applied at the edge of an image. [22]

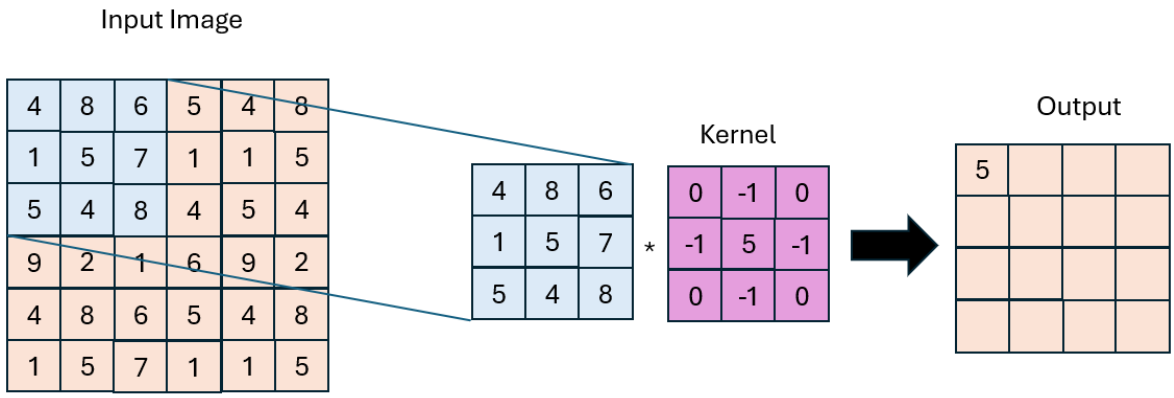


Figure 4.3: Application of a 3x3 filter to an image with no padding.

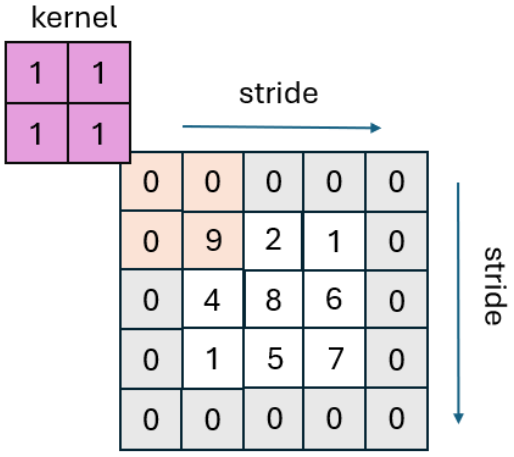


Figure 4.4: Application of a 2x2 filter to an image with padding of 1.

4.5 Activation Functions

Some architectures leave out the activation function where the process does not learn weights or parameters and is assumed that activation immediately follows a convolutional layer. Each of the activation functions are applied elementwise so the input and output dimensions are the same. Activation functions introduce non-linearities to the model, enabling the model to learn more complex patterns. Functions such as sigmoid, tanh, ReLU, and Leaky-ReLU are commonly seen activation functions in research papers. [20][21][23]

- Sigmoid [23]:

$$f(x) = \frac{1}{(1 + e^{-x})} \tag{4.3}$$

- Tanh [23]:

$$f(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \tag{4.4}$$

- Rectified Linear Unit [23]:

$$f(x) = max(0, x) \tag{4.5}$$

- Leaky ReLU [23]:

$$f(x) = max(\propto * x, x), \propto = constant \tag{4.6}$$

The AlexNet paper uses ReLU as an activation function for its hidden layers due to it being simpler and more efficient than the sigmoid activation function by avoiding exponential terms. Additionally, ReLU ensures that during backpropagation the gradient does not approach near zero in the positive interval, making training the model more efficient by avoiding exponential calculations. [23]

4.6 Pooling Layer

The pooling layer of a CNN is responsible for downsizing the spatial size of a feature. The process of pooling is used to decrease the required computational power and extract dominant features while training the model. Additionally, applying pooling to the input allows features to be both rotationally and translationally invariant. The two predominate types of pooling are max pooling and average pooling. Max pooling selects the pixels with the maximum value to add to the output array. [20][23]

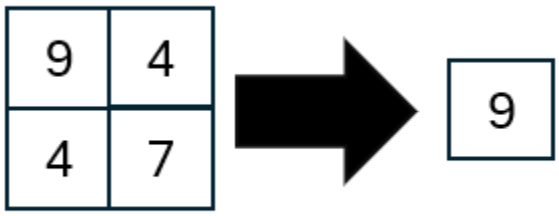


Figure 4.5: Max pooling on a 2x2 image.

Average pooling calculates the average value within the receptive field to add to the output array.

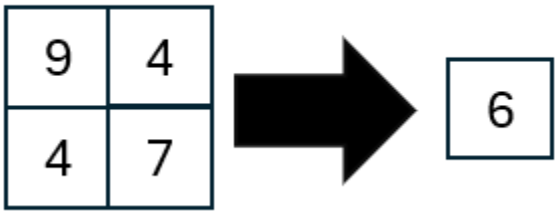


Figure 4.6: Average pooling on a 2x2 image.

Pooling operations are typically applied with a stride equal to their window size to avoid overlapping. [24]

4.7 Fully Connected Layer

The fully connected layer in a CNN architecture is used after convolutional and pooling layers for the final classification of the neural networks. The fully connected layer is connected to every neuron in the previous layer where each connection has its own weight. [20]

$$\hat{y} = \sigma_a(Wx_i + b) \tag{4.7}$$

Where:

- $\hat{y}$ is the output prediction
- W is the weight matrix
- x_i is the input
- b is the bias vector
- σ_a is the activation function

Fully connected layers take the features extracted from the kernels applied in the convolutional layers and learn features on a global scale. The Fully connected layers combine high-level features from the feature maps and patterns to make final predictions. The output layer is the final layer of

the network which produces the network's predictions. The structure of the output is task-specific, common tasks for CNNs are:

- Classification: typically uses SoftMax activation function and uses one neuron per class
- Regression: typically uses a linear activation function and uses one neuron
- Object Detection: typically produces a set of coordinates and class probabilities

The output layer is the layer in which the error of the network's predictions is calculated. Specific loss functions for the output layer are chosen depending on the structure of the output layer (e.g. classification, regression, etc). [25]

4.8 Forward Propagation

Forward Propagation is the first step in training a neural network where the data if passed through the entire network generates a prediction. In a feedforward neural network, the data only moves through the network in one direction: from input to output. As the data passes through the network each neuron in the hidden layers calculates the weighted sum to its inputs and applies an activation function before passing the data to the next layer. After the convolutional and pooling layers are completed the final feature maps are flattened to a 1D vector before being passed to the fully connected layers. The process of summing the weights and applying activation functions is repeated until the output layer is reached, which then returns the predicted value. The predicted value is then compared to the expected output value and the difference provides the error in the network position. The most common equation for calculating the error/loss in a CNN is cross-entropy loss for classification as seen in Equation 4.8. [23][26]

$$L = -\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log\left(p_{ij}\right) \tag{4.8}$$

4.9 Backpropagation

Before training the neural network, the weights and biases are initialized randomly for each neuron, then the training adjusts the weights and biases to create more accurate predictions. Randomly initializing the weights and biases allows the network to learn different features during training. The process of updating the weights and biases based on the error in the network's predictions is known as backpropagation, which occurs after forward propagation. Once receiving the error from the forward propagation step the error is used to calculate the gradient of error for each weight and bias. Backpropagation involves the following steps:

- Compute the loss between the network predictions and labeled values
- Calculate the gradient of the loss with respect to the weights and biases
- Update the weights and biases of the network using an optimization algorithm (e.g. Stochastic Gradient Descent, Batch Gradient Descent, etc) to minimize loss

The backpropagation step involves a learning rate (α) which is responsible for scaling the step so the gradient can adjust the weights of the network. In practice, it is common to experiment with different learning rates to find a balance between stability and convergence speed during training. [23] After calculating the loss as seen in Section 4.8 the next step in backpropagation is calculating

the gradient of the loss with respect to each parameter in the network. Equation 4.9 shows the error attributed to the output and hidden layers. Equations 4.10 and 4.11 show the gradient descent steps for the network where the weights are getting updated. [23][26]

$$\delta_j^h = \sum W_{jk}\delta_k^o f'(h_j) \tag{4.9}$$

$$\Delta w_{ij} = \alpha\delta_j^h x_i \tag{4.10}$$

$$\Delta w_{pq} = \alpha\delta_{output} x_i \tag{4.11}$$

Where δ_k^o is the error for each output unit k, δ_{output} is the output error, x_i us the input unit values, and w_{ij} is the weights between the hidden and input layers. After these values are calculated we can then use it to update the weights in the model and improve our results during training using gradient descent. Additional steps such as normalization and regularization can be taken to further improve the performance of the model being trained.

5. Experimental Setup and Validation

5.1 System Design

The navigation system implements a distributed architecture designed to ensure safety, reliability, and real-time performance. The experiment aims to implement the navigation algorithm developed in paper on real-world hardware and validate using simulation and flight test data. The direct outcome of the experiments will be to test the accuracy and performance of the feature matching, position estimation, and trajectory tracking of the navigation algorithm. The separation of flight control, vision-processing, and ground control provides redundancy and fault isolation by a distributed architecture. The navigation system consists of several distinct subsystems that control the vehicle and process sensor inputs in real time. To increase the safety of operation the drones, flight controls, and data handling are processed on different computers, and commands are sent between the connected devices. The computer platform controlling the drone is a Holybro pix32 Flight Controller (FC) which handles autopilot hardware and fuses all the drone hardware onto a singular device. The Holybro Pix32 flight controller operates using PX4 v1.13.3 firmware, providing essential flight control capabilities through a real-time operating system (NuttX). The firmware manages critical flight operations including:

- Controls flight modes and transitions
- Handles multiple sensor inputs and calibration
- Implements control loops for stability
- Monitors system health and manages failsafe responses

The computer platform responsible for receiving and processing the camera data is a Raspberry Pi RP4. The RP4 and FC are connected to a Ground Control Station (GCS) which allows the camera stream from the RP4 to send navigation commands to the FC. [27][28]

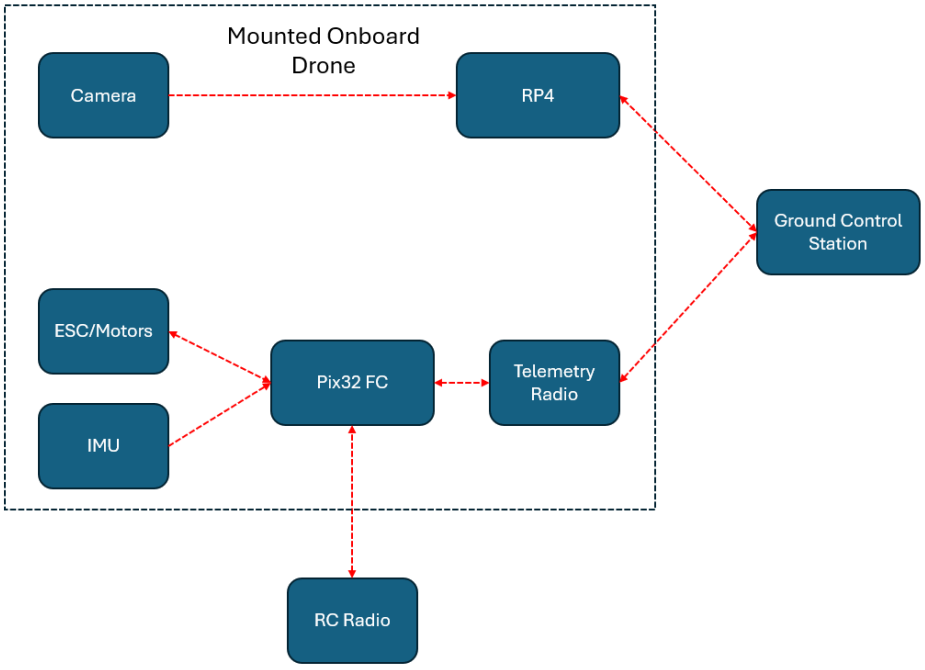


Figure 5.1: Simplified architecture between the drone subsystems.

The GCS communicates with the FC using the messaging protocol MAVLink which follows a publish-subscribe design pattern to send commands and receive data from the flight data. MAVLink is responsible for handling communication and detecting communication errors (packet drops, corruption, and packet authentication). The navigation system is integrated with MAVSDK which is a collection of libraries that interface with MAVLink that enable tasks such as computer vision, obstacle avoidance, and route planning. In the scope of the project MAVSDK handles processing the images received from the camera, the output of the visual odometry system, and transforming the output to commands that control the drone. [27] [28]

5.2 Hardware Setup

Table 5.1 shows the hardware used in the experimental setup and if it is used in Hardware-In-The-Loop (HITL) simulation or Flight Hardware (FH). Please note that specific wiring harnessing and mounting hardware (bolts, washers, nuts) are not included in the hardware list as are general use hardware and not custom manufactured components.

Table 5.1: Hardware list used in flight hardware and HITL testing.

Unit Name	**Definition**	**Use Case**
Holybro Pix32	Flight Controller Hardware	HITL and FH
Tarot FY690S	Quadcopter frame and mounting hardware	FH
Brushless DC Motor	Multirotor propulsion actuator	FH
40A ESC	Motor speed controller	FH
1255 12.5 Carbon Fiber	Propellor (CW and CCW)	FH
4400MAH LiPo	LiPo Battery	FH
1080p LogiTech Camera	USB Connected Camera	FH
RP4	Raspberry Pi (onboard compute platform)	FH
FrSky Taranis QX7	Radio RC Controller	HITL and FH
FrSky X8R	Radio Receiver	HITL and FH
Holybro SiK V3	Telemetry Radio	HITL and FH
3D Printed Enclosure	Housing for USB Camera and onboard compute	FH

The USB camera interface runs at a frame rate of 30 HZ, image compression format of MJPEG, and autoexposure for varied lighting conditions. The camera driver gains CPU priority on the RP4 hardware for real-time scheduling of vision tasks. Where vision processing tasks can be computationally expensive, the payload and drone will use two isolated wiring harnesses. The

payload and drone will use two batteries to ensure that each of the components receives adequate power during flight operations.

The physical mounting of the camera and onboard computer is a 3D-printed enclosure that positions the camera 90 degrees downward, so it is perpendicular to the ground. The 3D printed enclosure does not feature any vibration dampening and the components are hard mounted to the enclosure using bolts and nuts. The 3D-printed enclosure is then bolted onto mounting harnesses which use rubber dampers to mate with the Tarot multirotor frame. The rubber dampers reduce the induced vibration from the motors but have no gimbal or other anti-vibration electronics to smooth the camera stream. The FC is soft mounted to the multirotor frame using gel dampers that reduce the vibrations from the motors while keeping the IMU nearly in line with the drone's CoG. Typically, the IMU is required to be fixed with a semi-rigid structure to the same reference frame as the drone such that the measurements taken are representative of the drone's actual position and orientation (as opposed to being mounted to a gimbal).

5.3 Simulation Environment and Considerations

Gazebo, AirSim, and JMAVSIM were all simulations considered for testing due to containing interfaces with PX4 autopilot and HITL capabilities [29]. Originally, the simulators were also considered for testing the navigation system by using the camera model built into the simulators. However, the rendering rate of the simulators and overall reconstruction quality from the simulator were inadequate for the visual odometry algorithms written for the navigation system. The primary issue was missing pixels or dropped pixels which caused issues with the feature extraction and matching algorithms. The missing and dropped pixels created a high deviation between the frames, causing the algorithm to produce a high rate of false positive results. Correcting the pixel issues would either require a better rendering engine or a camera model in the simulator. Both solutions are out of the scope of the project and overall, are not critical to ensuring safe operation or mission success during flight testing. The simulator evaluation was then based on the simulator's ability to mimic real-world communications, firmware, and interfaces as closely as possible. Gazebo features a high-fidelity physics simulation and detailed environment modeling, but has camera rendering limitations and requires a high computational overhead. AirSim features information-dense APIs, detailed physics engines, and customizable sensor simulations, but suffers from frame rate issues, complex configuration, and high computational overhead. JMAVSIM features lightweight computational requirements, direct PX4 and MAVLink interfaces, and quick rendering, but suffers from low-fidelity camera models, simple physics engines, and limited sensor customization. With the advantages and limitations of each simulator considered, JMAVSIM was chosen for HITL testing due to its direct interface with PX4 autopilot and quick rendering, at the cost of image fidelity and physics simulation.

To test communications for safe flights the GCS was programmed to fly a set of waypoints using velocity profiles (like the output of the navigation system) then the pilot will interrupt with joystick input to the RC Radio. The expected outcome of the experiment is that the manual inputs from the pilot will override the GCS commands and allow the pilot to regain control. The following shows the software architecture for the HITL testing in the simulation environment:

- JMAVSIM instance
- PX4 SITL/HITL bridge
- MAVLink router configuration
- QGroundControl interface

Then for the control systems:

- MAVSDK command generation
- RC input processing
- Failsafe monitoring
- State machine management

Figure 5.2 shows the JMAVSIM Simulator running, and Figure 5.3 shows the QGroundControl user interface with a waypoint mission loaded onto the vehicle.

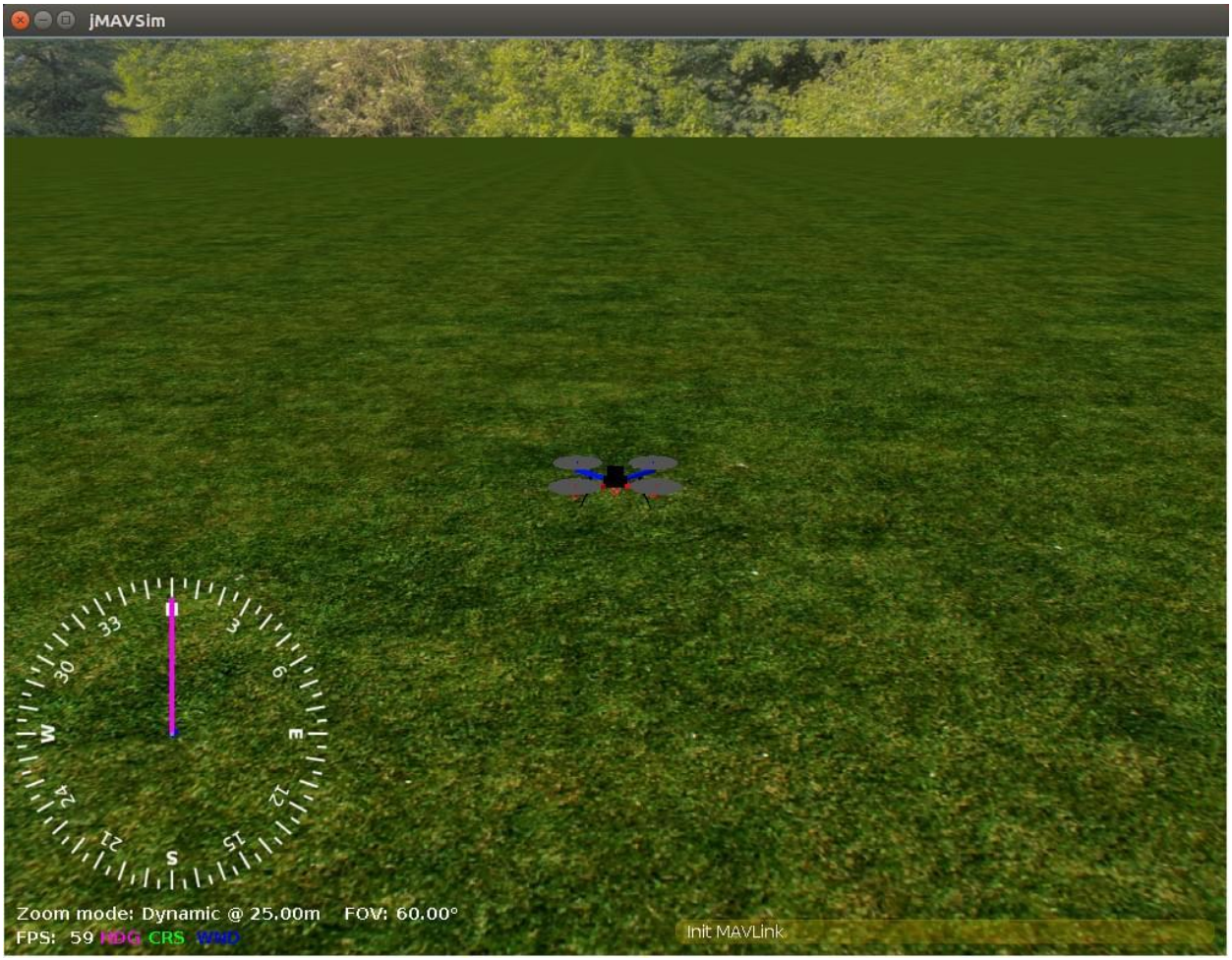


Figure 5.2: JMAVSIM Simulator connected to PX4 autopilot [30].

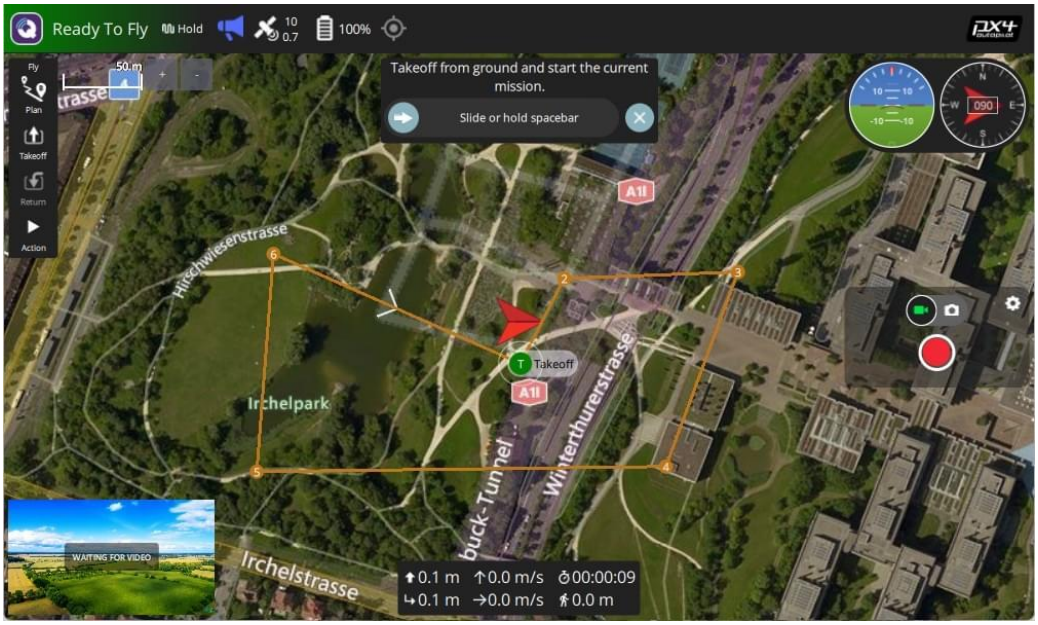


Figure 5.3: QGroundControl user interface with waypoint mission loaded onto vehicle [31].

5.4 HITL Test Plan

Before testing the navigation system in flight, the system is tested in a simulated environment to ensure that all communications are working correctly and minimize system errors. In practice simulation testing will not weed out all problems as many communication issues arise because of the testing environment and hardware limitations, however, simulation testing will serve as a benchmark and ensure that the designed failsafe works as intended. An important test will be ensuring that manual inputs override the autopilot commands. For example, if the autopilot commands are sending the drone off-course manual inputs from the RC Radio must override existing commands and allow the pilot to regain control over the drone. The simulator used in testing is JMAVSIM which is a simple multirotor simulator that uses PX4 autopilot and MAVLink to test takeoff, land, flight, and fail conditions for testing. JMAVSIM does not provide a camera model with high enough fidelity to validate the visual odometry module, so the simulator is used to validate control commands and fail conditions of the navigation module. JMAVSIM supports both Software-In-The-Loop (SITL) and HITL for MAVLink connection testing on development computers and PX4-specific firmware version on real hardware.

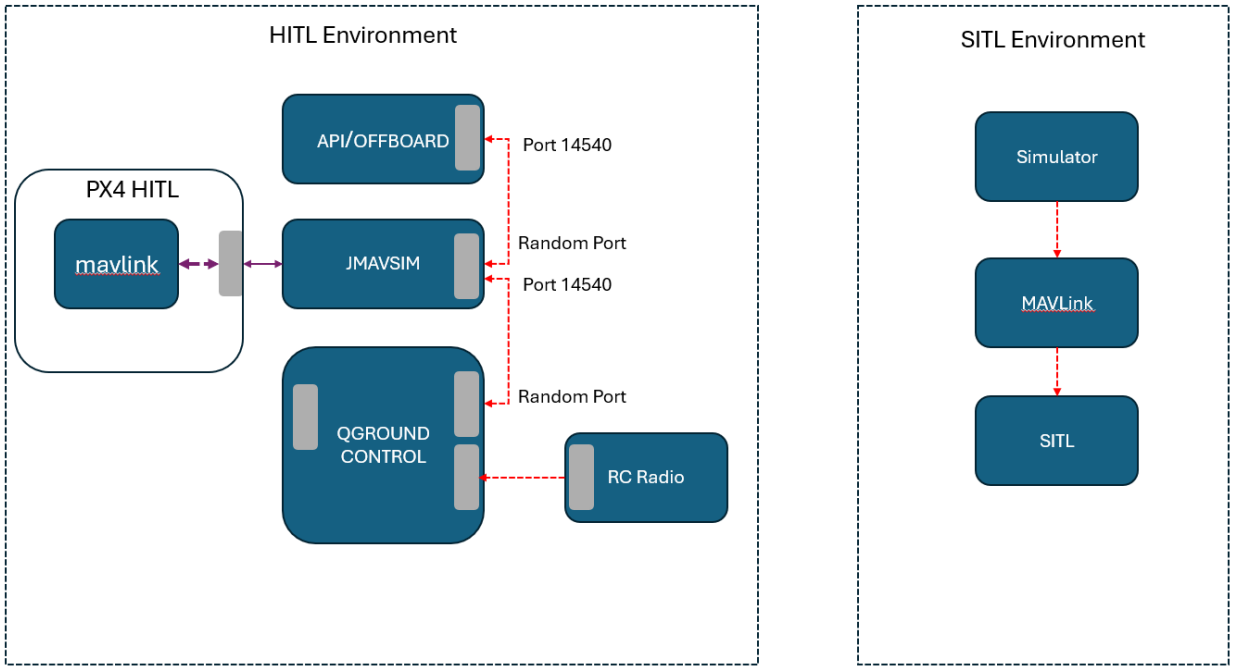


Figure 5.4: SITL and HITL Simulation Environments

To validate that the proper failsafe is working as expected, the system is tested using JMAVSIM, QGroundControl, pix32 FC, Telemetry Radio, RC Radio, and MAVSDK control module for HITL. By sending the simulator and FC a simulated trajectory to fly using the MAVSDK control module designed for the navigation system, the HITL test checks to ensure all the radio connections are established and can communicate with the GCS and autopilot with the same hardware and firmware that will be used during flight testing. The HITL test will send velocity commands to the drone using an MAVSDK module paired with the GCS, during the sent command user input from the RC radio will be sent. The test will be considered passed if the user input overrides the commands sent from the GCS, otherwise, the test will be considered failed and flight logs will be reviewed to determine the communications between the submodules. The following list shows the control hierarchy:

- RC Radio Manual Inputs (Highest Priority)
- Battery Critical Level
- Signal Loss
- Position Error Bounds
- Navigation System Inputs

In the control hierarchy, the RC Radio manual inputs are inputs from the pilot's joystick commands. The battery critical level is from the PX4 autopilot which monitors the drone's battery level. When the battery voltage is too low, the autopilot commands the drone to either land or return to a specified location (waypoint) and then land. The signal loss control returns the drone to the last known position when RX/TX communications are lost and can be programmed to either hold altitude or land at the location. The position error bounds are set range limits that command the drone to either hold altitude or return to an in-bound position. The position error bounds are set before flight in QGroundControl to attempt to prevent the drone from flying to positions that are out of range of communication signals, the bounds are hardware-specific, and estimates can be

found in manufacturer datasheets. The last priority on the control hierarchy is the navigation system developed in the paper as it is not developed with collision avoidance or position bounds, so it needs other features to ensure that it is safely enabled. The GCS monitors both vehicle status and sensor data, Table 5.2 shows the systems the GCS monitors. Performance metric will use flight logs from the PX4 autopilot software to validate:

- Position tracking
- Attitude information
- Packet loss
- Link quality
- Command latency
- System Stability

Table 5.2: GCS system monitoring for vehicle status and sensor data.

Vehicle Status	**Sensor Data**
Battery Voltage and Current	Attitude and Heading
Motor Outputs (PWM)	Position and Velocity
Flight Modes (manual, waypoint, VO-system)	Altitude
GPS Status	RC Signal Strength

During testing with flight hardware, the HITL testing does not guarantee mission success, factors such as signal obstruction, hardware degradation, and time synchronization will cause deviations from the simulated environment and connections. Risk mitigation strategies will include preflight checks, fail-safe verification, and system monitoring to ensure safe flight operations and mission success. Starting with simple tests such as a takeoff and hover test will ensure that the drone is operating as expected, and then boundary testing can be performed to assess the drone's capabilities. The following tests establish the drone's communication performance and failsafe reactions in a simulated environment.

Basic Controls Test:

- Arm Motors
- Climb to set altitude
- Hover for 30 seconds
- Validate position hold
- Land
- Disarm motors

Failsafe Verification Test:

- Arm drone
- Climb to set altitude
- Test return-to-home function
- Move the drone to the desired position
- Test position hold failsafe
- Test land failsafe
- Disarm drone

Waypoint Failure Verification Test:

- Upload waypoint trajectory
- Arm drone
- Climb to set altitude
- Move the drone to the desired position with waypoint control
- Test manual command takeover
- Resume waypoint trajectory
- Force failsafe condition (turn off manual controller)
- Automatically Return-to-Land
- Disarm drone

6. Results

For a set of 50 images taken at the test site, the feature extraction and matching algorithms are tested to evaluate the performance of each algorithm. The performance metric is the total number of features extracted, features matched, and the computation time using the following feature extractors ORB, SIFT, and XFEAT. Figure 6.1 shows a sample image from the dataset of the flying field where the navigation method was tested. Table 6.1 shows the results of the feature extraction testing on the dataset. Table 6.2 shows the results of the feature matching on the same dataset used in Table 6.2 against Figure 6.2 (constant) which is a Google Earth Satellite View of the flying field (at approximately 100 meters scale).

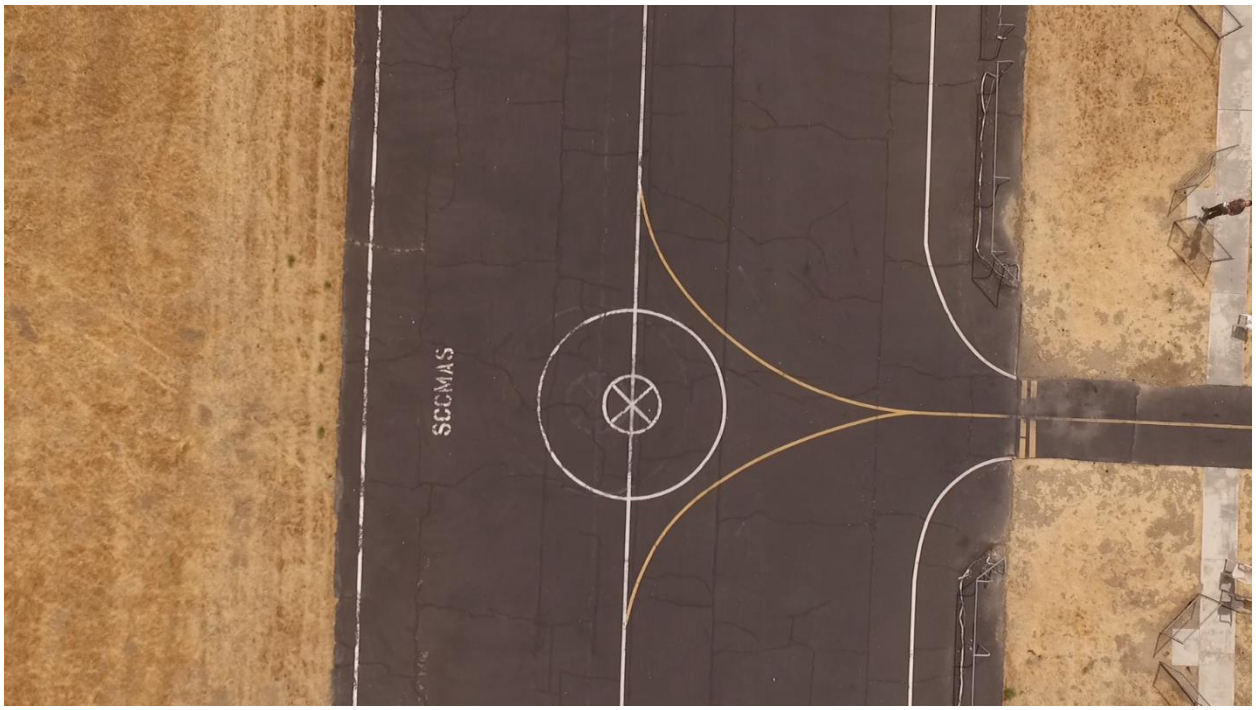


Figure 6.1: Sample image taken from a test flight of flying field mid-waypoint trajectory at 61 meters AGL.

Table 6.1: Feature extraction time and keypoint comparison between methods on fixed dataset.

Feature Matching Method	**Average points per Image**	**Processing Time (ms)**
ORB	2959.88	104.053
SIFT	3000	178.338
XFEAT	3000	176.25

Figure 6.2: Google Earth view of the flying field at approximately 100 meters AGL scale.

Table 6.2: Feature matching time and extraction comparison between methods on the fixed dataset.

Feature Detection Method	Average Keypoints Per Image	Average Matched Keypoints Per Image	Processing Time (ms)
ORB	2959.88	9.24	155.16
SIFT	3000	65.36	510.414
XFEAT	3000	21.46	488.72

Feature matching using ORB, SIFT, and XFEAT feature detection methods show that ORB processes the feature matching in the fastest time, but has the lowest number of detected and matched keypoints per sample image. SIFT feature matching features the longest processing time but produces the highest number of matched features with the sample dataset. Our results show from features matching that for the best case results (SIFT features) roughly 2% of the detected features are matched after performing outlier detection. Many of the rejected features that are removed during outlier detection are in the grass or on non-painted sections of the runway.

The simulated environment for HITL testing allows for verification of system monitoring and control authority of the drone by using the same flight firmware and communications as the hardware. The results of the hover test show that all commands are received properly from the

controller and the PX4 firmware can achieve 0.2-meter positional accuracy with GPS and track yaw with a 2.0-degree accuracy. During the hover testing the results show drone can successfully complete the maneuver without triggering any failsafe's and there is now a benchmark for system performance as shown in Figures 6.3 - 6.5.

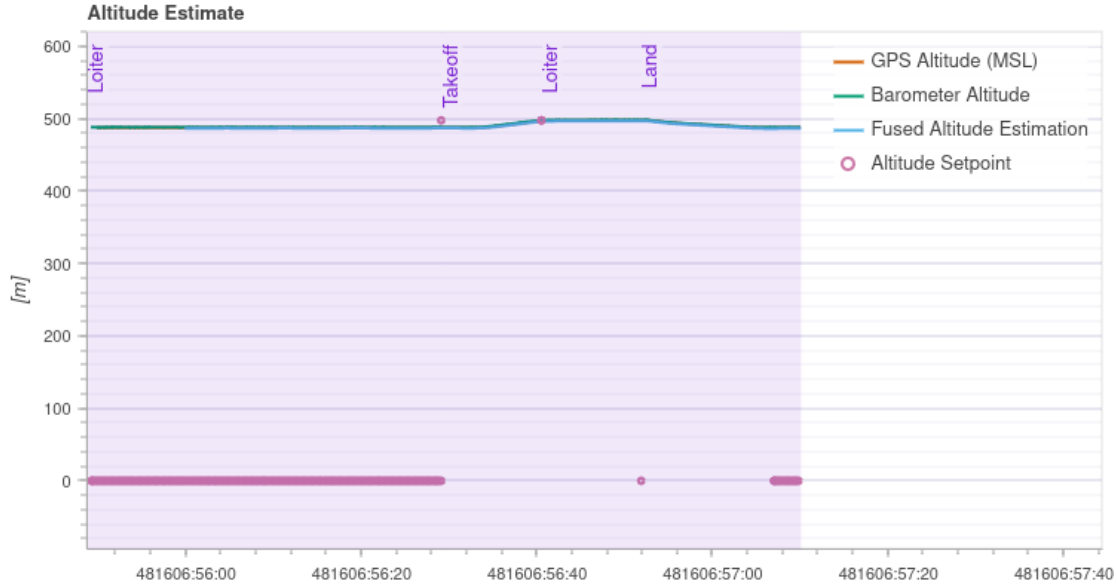


Figure 6.3: Altitude vs Time plot with GPS, barometer, and sensor fusion position for hover test.

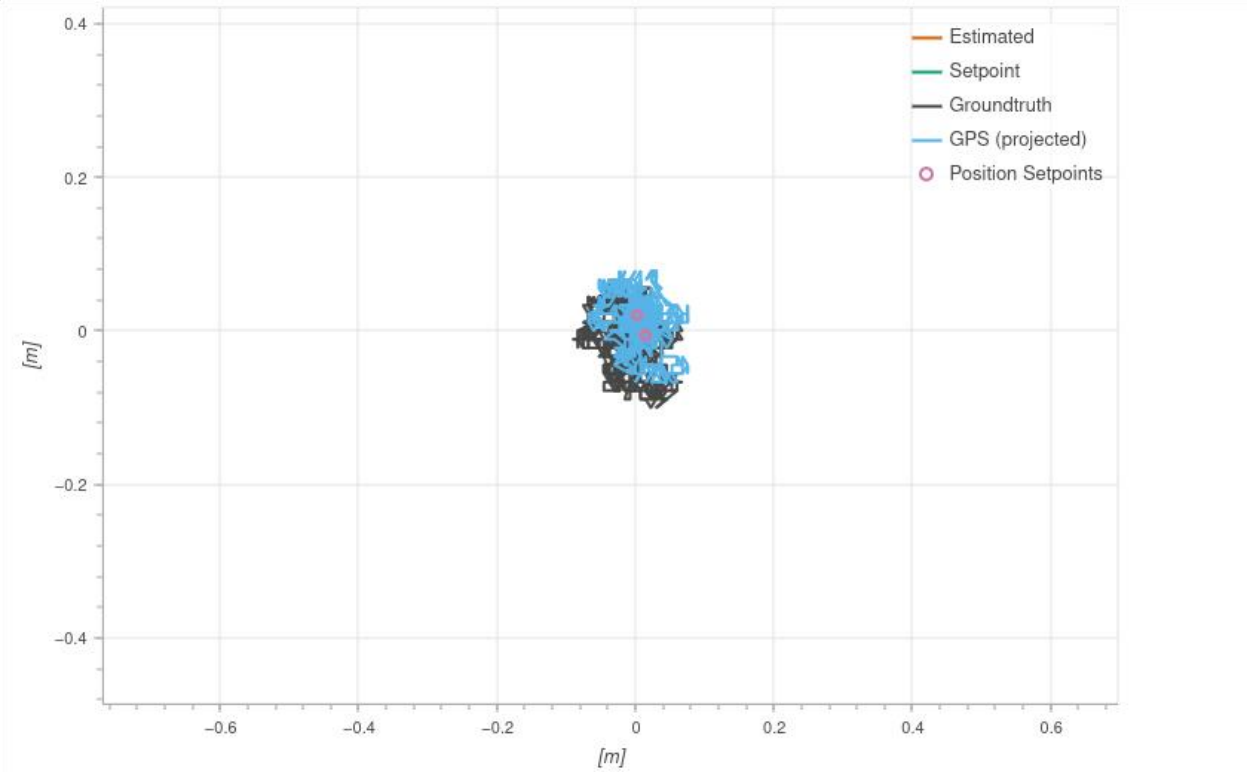


Figure 6.4: Position plot for hover test.

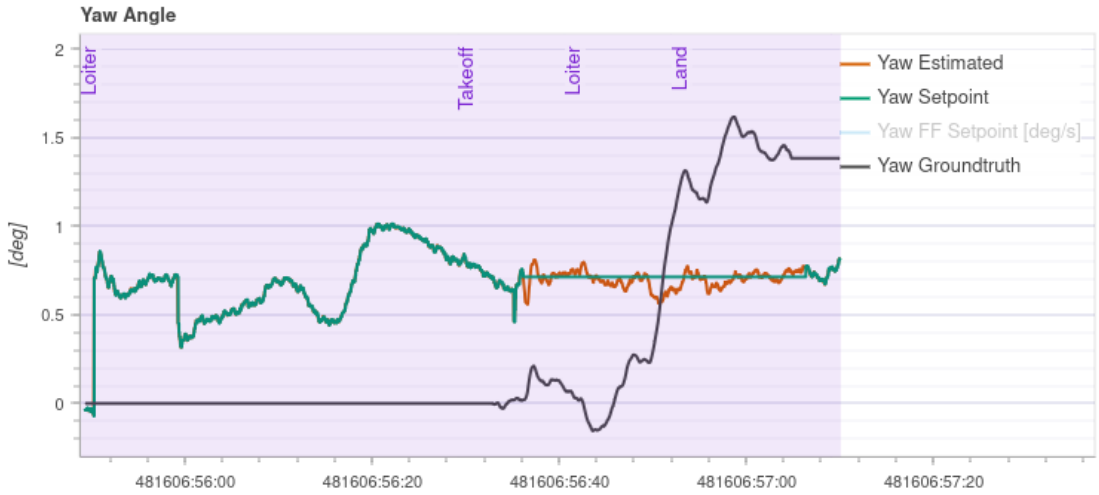


Figure 6.5: Yaw vs Time plot for hover test.

The results of the waypoint test show that all commands are received properly from the controller and the drone can fly a series of waypoints. The HITL simulation shows that the drone can traverse similar distances to what is expected during the real-world flight testing without triggering any of the failsafe conditions. This test ensures that during flight testing the expected trajectory will not reach any out of bound conditions for the communications hardware and signal will remain connected during flight operations. Results of the testing and transitions between flight modes can be seen in Figures 6.6 - 6.8.

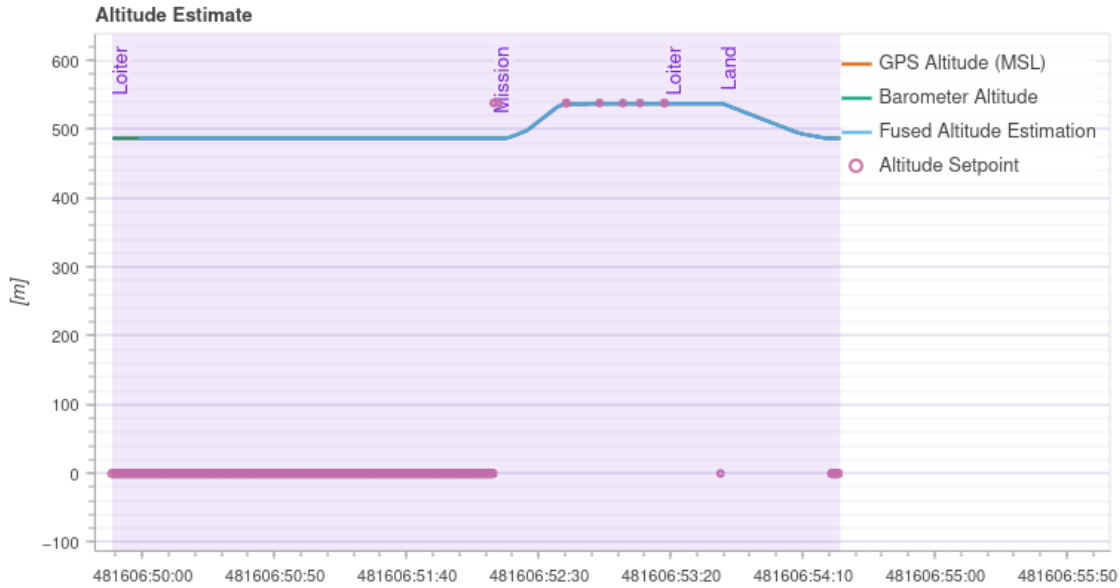


Figure 6.6: Altitude vs Time plot with GPS, barometer, and sensor fusion for waypoint test.

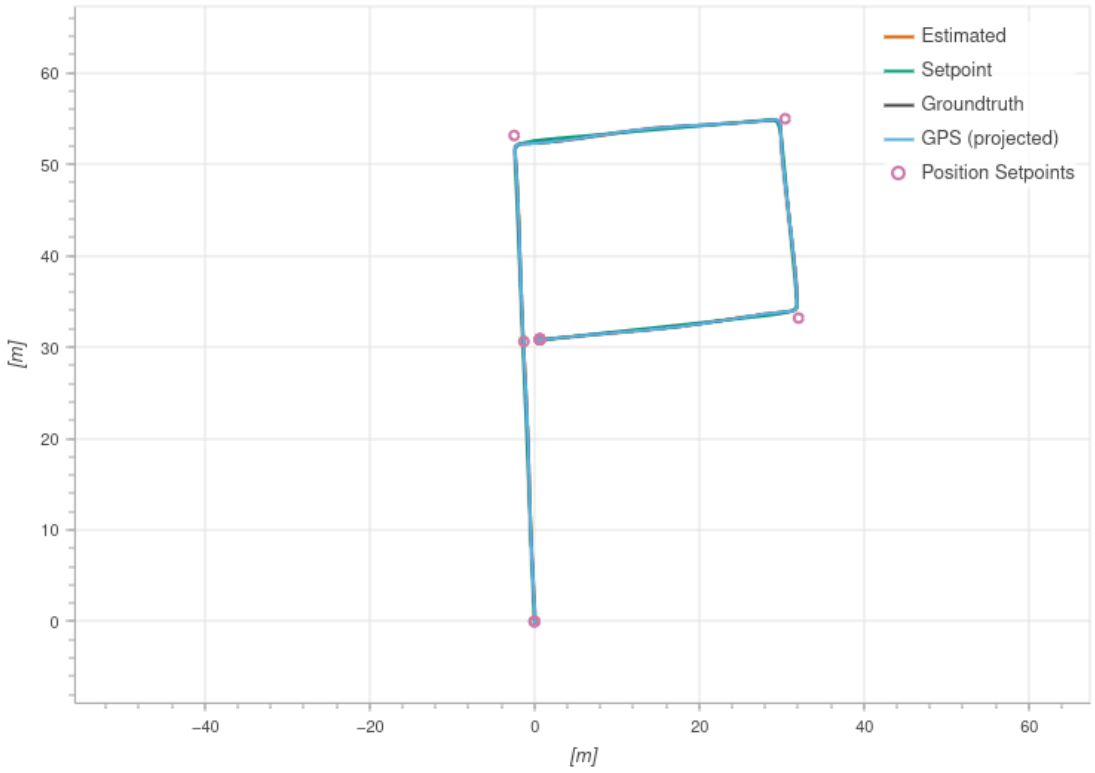


Figure 6.7: Position plot for waypoint test.

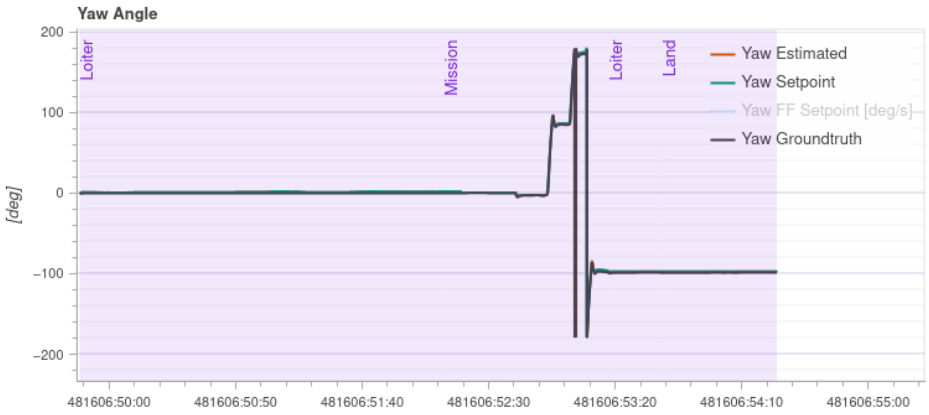


Figure 6.8: Yaw vs Time for waypoint test.

The results of the manual takeover and Return to Land testing demonstrate the ability of the drone to recover during hazardous scenarios. During the manual takeover the user can gain control of the drone and interrupt its waypoint trajectory without delay. The result is a drop in altitude and translation as the user inputs manual controls before stabilizing in hover. The results can be seen in Figures 6.9 - 6.11 during the highlighted “Manual” stages of flight. The drone is

then able to resume its waypoint trajectory during the second “Mission” stage of flight before the signal is lost and the “Loiter” and “Return to Land” commands are sent to the drone. After the signal is lost and the “Return to Land” command is engaged the drone returns to its home position and safely lands at its takeoff altitude.

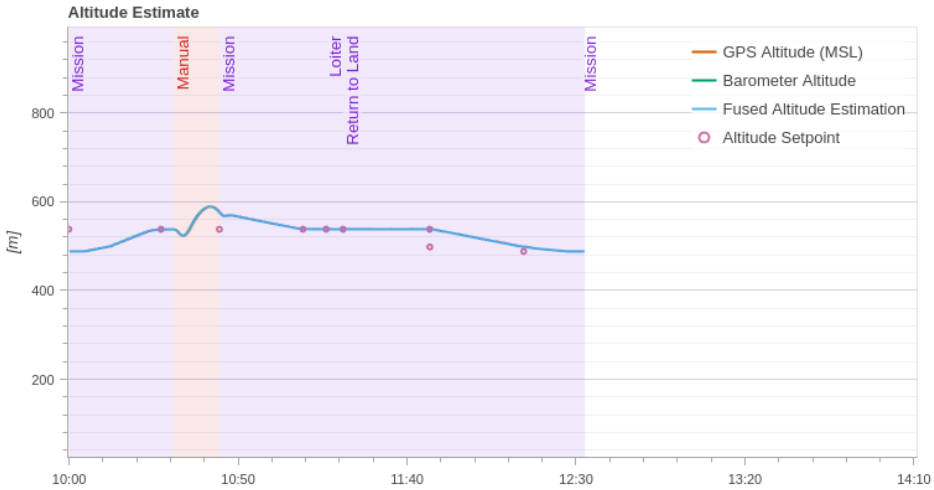


Figure 6.9: Altitude vs Time for failure test.

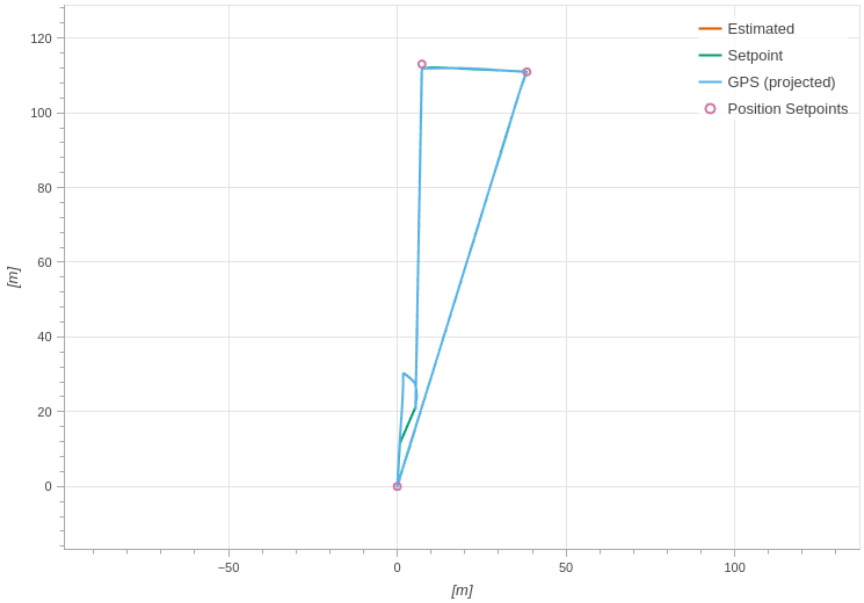


Figure 6.10: Position plot for failsafe test.

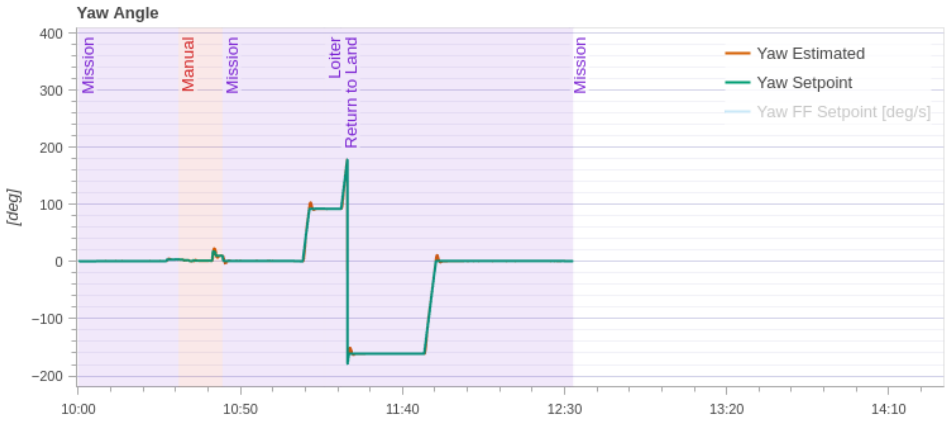


Figure 6.11: Yaw vs Time for failsafe test.

The results of the visual-waypoint navigation show that the navigation algorithm can estimate its global position to an average of 4.30-meter accuracy at an altitude of 25 meters with a standard deviation of 2.16 meters. At an altitude of 50 meters the global position estimate improves to an average of 2.81-meter accuracy with a standard deviation of 0.47 meters. The results of the global estimation using visual waypoints can be seen in Table 6.3 and the true position of the user specified visual waypoints can be seen in Figure 6.12.

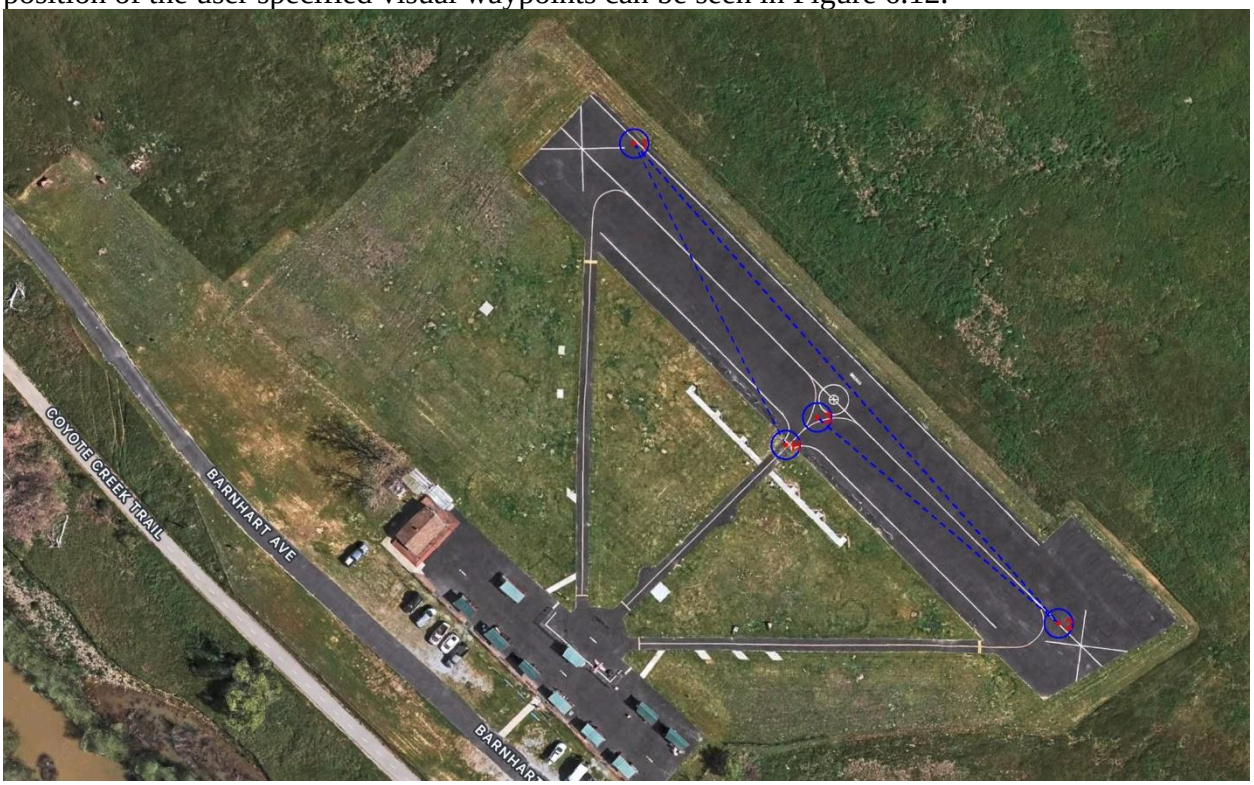


Figure 6.12: Ideal trajectory produced from user-defined waypoints.

Figure 6.13: SIFT feature matching sample first waypoint between map and UAV camera stream at 25 meters.

Figure 6.14: Navigation algorithm position estimate and true state of the first waypoint.

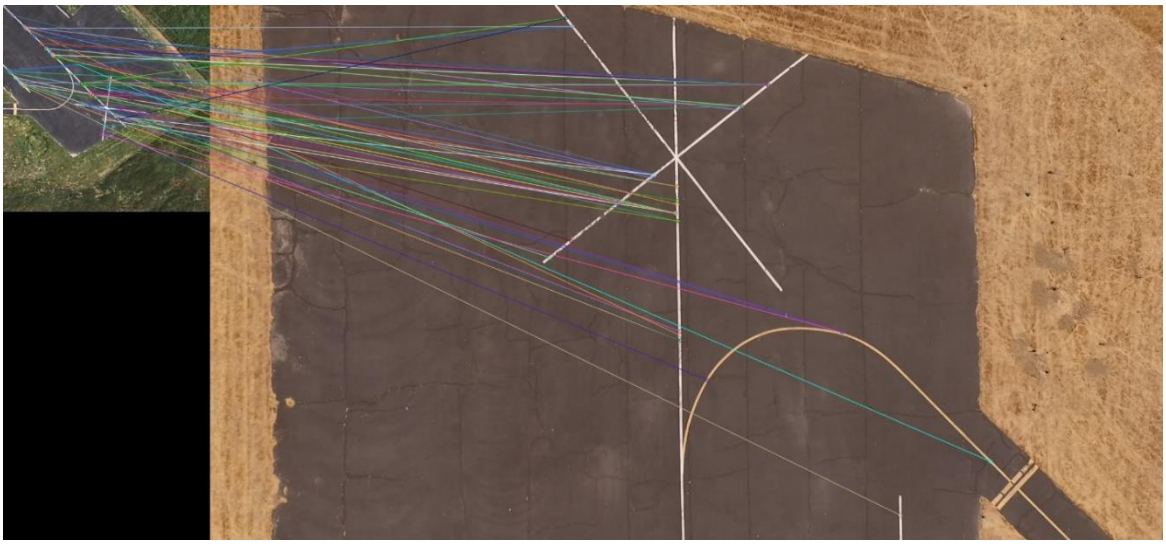


Figure 6.15: SIFT feature matching the second waypoint between map and UAV camera stream at 25 meters.

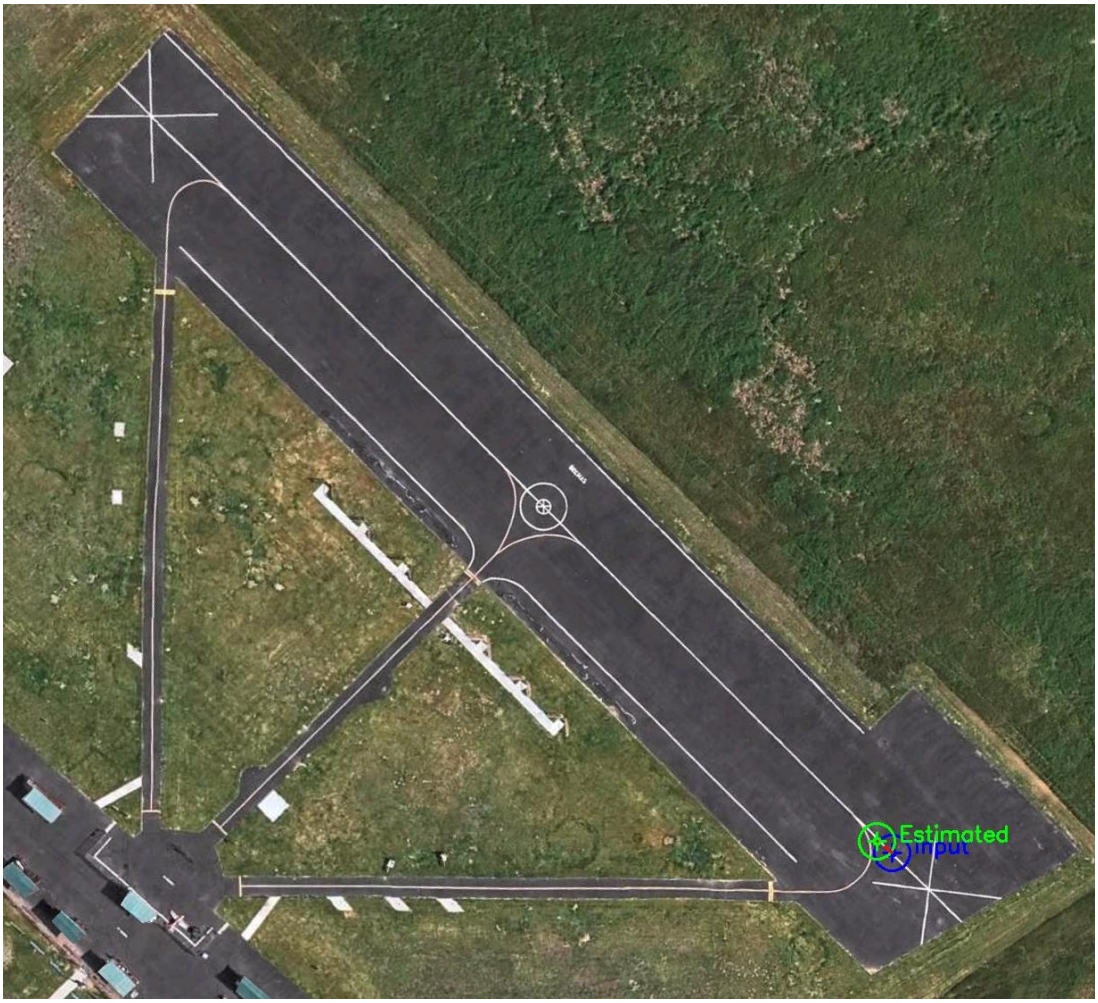


Figure 6.16: Navigation algorithm position estimate and true state of the second waypoint.

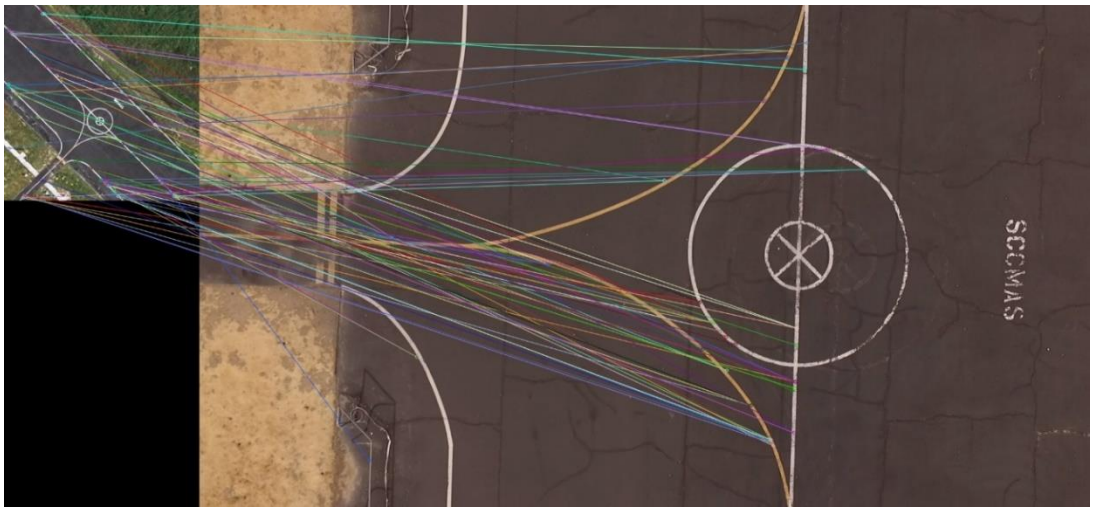


Figure 6.17: SIFT feature matching of the third waypoint between map and UAV camera stream at 25 meters.

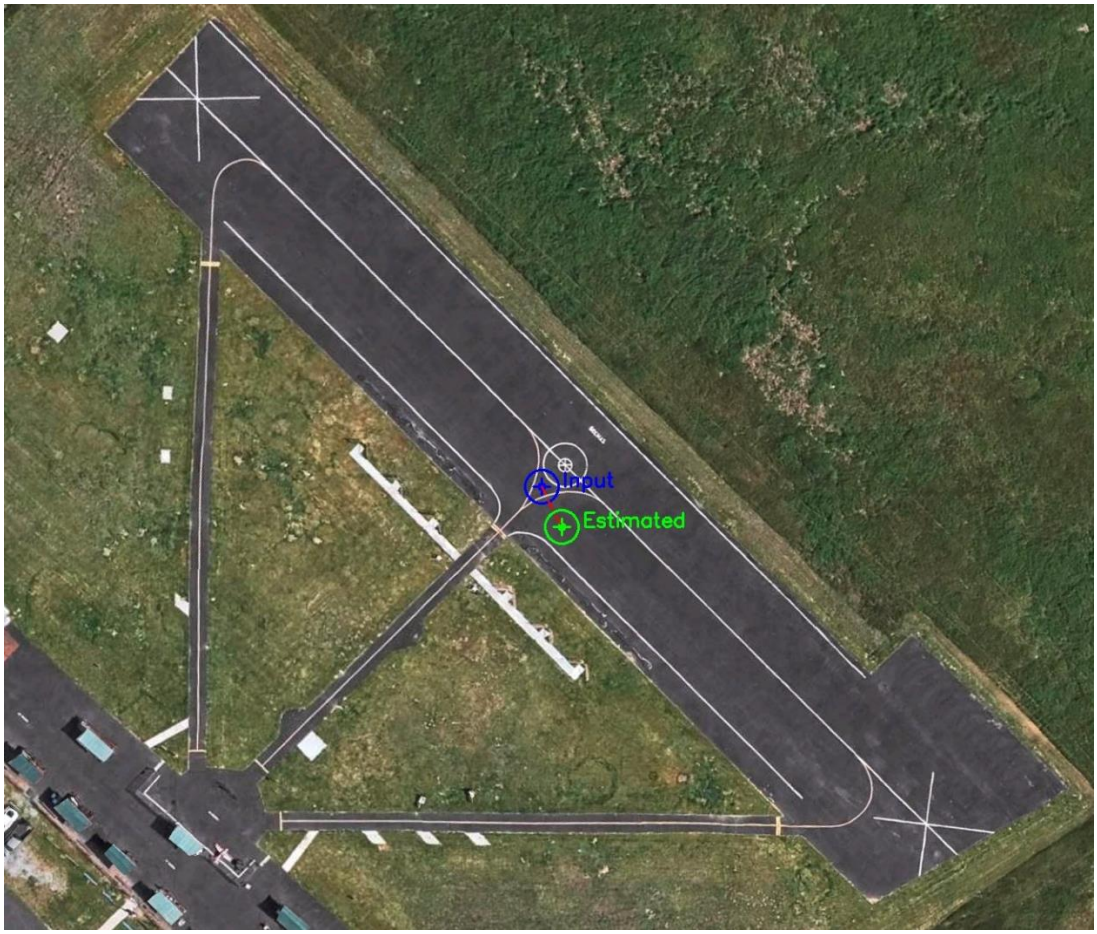


Figure 6.18: Navigation algorithm position estimate and true state of the third waypoint.

Table 6.3: Error in estimated vs true state of visual waypoints.

Waypoint	Altitude (m)	X Positional Error (m)	Y Positional Error (m)	Euclidean Distance (m)
1	25	2.99	6.00	6.71
2	25	1.99	1.56	2.53
3	25	2.86	2.25	3.65
1	50	2.42	2.31	3.34
2	50	1.81	1.82	2.56
3	50	2.24	1.12	2.51

The VO algorithm can detect, track points, and provide state estimates in real-time. Figure 6.19 shows the heading animation from the output rotation measurements from the decomposed homography matrix converted to degrees for an intuitive display for users. Figure 6.20 shows the heading tracking over time for the commanded yaw, measured yaw (homography matrix output), and the estimated yaw (EKF estimated output). Figure 6.20 shows that the developed EKF can filter the measurement noise while maintaining accuracy to the commanded yaw (ground truth). During sharp rotations the accuracy of the measurement begins to degrade and gaps between the measured and commanded yaw begin to form and propagate.

Figure 6.19: Navigation algorithm running in real-time with heading tracking and trajectory tracking with EKF.

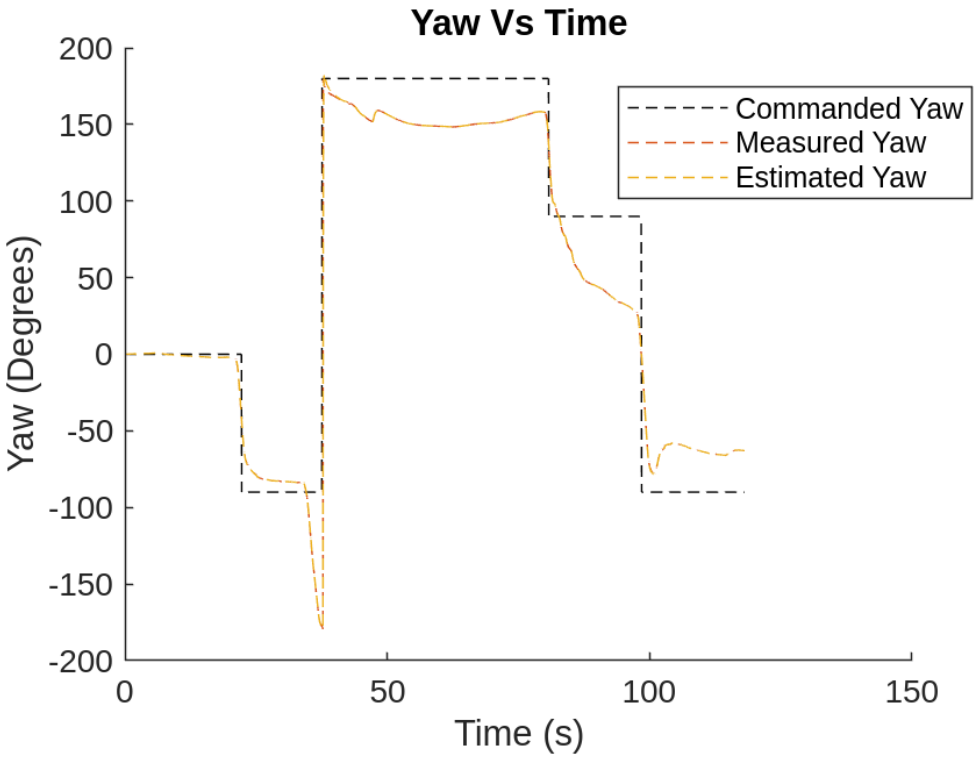


Figure 6.20: Plots of EKF corrections against commanded yaw and measured yaw during flight.

7. Conclusion

The monocular VO navigation algorithm developed in this paper achieves a global position accuracy of 4.30-meters without EKF corrections at an altitude of 25 meters and an accuracy of 2.81-meters at an altitude of 50 meters. The results at a 25-meter altitude show a higher standard deviation and lower accuracy with respect to the 50-meter altitude. The higher altitude allows for a wider field of view from the lower altitude, allowing the feature matching to have more landscape to compare against the satellite view of the preloaded map. Additionally, the higher altitude more closely mimics the satellite's view and resolution, allowing features to remain in frame of the camera for a longer period between frames. The higher altitude allows for larger features to become more prominent and for smaller features (such as grass patterns or shadows) to be filtered out as a result of the resolution changing with altitude. Filtering out the smaller features increases the algorithm's robustness against seasonal changes and lighting variations by providing a more complete view of more distinct features (such as the runway). Testing the algorithm at higher altitudes proves effective at increasing the accuracy of the global estimation by providing natural filters and more closely representing the satellite view's scale. Most satellite navigation requirements for aircraft by the FAA are around 7.0-meter accuracy 95% anywhere on or near the surface of the earth [32]. The results surpass the FAA's satellite navigation requirement of 7.0-meter accuracy (95% confidence) for aircraft near or on Earth's surface. The algorithm demonstrates the ability to use visual waypoints to localize itself but struggles to accurately track its trajectory using VO. Trajectory tracking is an inherent issue with monocular visual odometry due to scale ambiguity from the single camera. Common ways to resolve scale ambiguity in 3-Dimennsions are Perspective-n-Point, integrating measurements from an IMU, or using stereo vision. Implementing a hybrid solution is common industry practice for navigation and autonomy needs.

The results of the navigation system highlight tradeoffs between hardware and performance. For the feature extraction and matching algorithms, the XFEAT neural network developed by Google features the largest number of detected features for the dataset and completes extraction and matching faster than SIFT features. ORB features detect the least number of features with the fastest detection and matching time which is the anticipated result from the literature review. However, these results were achieved on a computer with a dedicated GPU and optimized drivers for the hardware. Running the algorithms on the flight hardware (RP4) shows that the non-CNN algorithms perform much faster and can consistently produce results. The flight hardware streams the camera data at 30 FPS (like human optical perception rates), and the rate will serve as a benchmark for visual processing operations. Running the XFEAT algorithm on the desired image resolution and sizing resulted in degradation to the navigation algorithm FPS to less than 1 FPS which does not allow for real-time use. Using ORB and SIFT algorithms degraded the navigation algorithm to around 17 FPS on flight hardware. To support XFEAT for visual processing the image dimensions or quality could be reduced to speed up processing time or a more powerful compute platform could be used for flight hardware. Alternative hardware to GPUs could be Tensor Processing Units (TPU) or Neural Processing Units (NPU) which greatly speed up computation times for ML and DL tasks. However, the processing units or more powerful computer platforms (such as the NVIDIA Orin series) will require larger voltage inputs, higher power draw, and larger size which are all detrimental to the aircraft's payload sizing and weight requirements. Therefore,

due to cost, weight, and sizing for the flight hardware using SIFT features for feature detection and matching was determined to be the most efficient path forward for the navigation algorithm.

The results of the feature matching experiment further validate the choice of using SIFT features where SIFT features yield the highest average of matched features per image. It should be noted that the test environment is sparse as the location is in the middle of a field with a runway. The sample map from Google Earth was also recorded during a different season during testing which caused issues in feature matching due to lighting and foliage coloring between seasons. The runway provides some unique features with lettering, markings, and cracks in the asphalt, but the grass areas provide no distinctly unique features. The grass often yields a high rate of false positive detections, which, when performing outlier detection results in a reduced number of detected features, as seen in Table 6.2. The feature sparse environment shows near worst-case scenario testing and shows that the algorithm is robust to operating conditions. Due to FAA Part 107 regulations, the algorithm was not able to be tested in urban or suburban areas which would result in more feature-dense environments (cars, houses, gardens, etc.). Testing using sample images gathered from urban environments shows higher feature-matching averages, however, due to testing restrictions, it was not used to develop the algorithm. Testing in urban environments would also require some form of collision avoidance due to object density to ensure safe operations.

The simulator HITL testing shows that all communications work as intended and that the control hierarchy is validated as defined in Section 5.0. The HITL testing shows that the user can enter manual inputs and regain control of the aircraft during all phases of flight and that the return to land features work as intended for the aircraft. The stock PX4 estimator shows some errors in observing the yaw with roughly a 2.0-degree error over the flight test trajectories. The HITL sensors were calibrated by PX4 documentation to ensure that the system would simulate the hardware noise and bias of the sensors. Improved hardware and sensors could reduce the error in the system, but for the sake of experiments the hardware and firmware have been characterized for testing. Testing of the visual odometry algorithm will use the PX4 sensor fusion estimator as a benchmark to compare against. During the manual takeover test switching from “Mission” to “Manual” flight modes causes the drone to lose altitude while the user stabilizes the throttle to regain altitude. The HITL testing shows realistic behavior of the altitude loss in GPS denied environments using PX4 hardware due to position hold requiring a GPS lock.

While the VO algorithm faces challenges in positional trajectory tracking, the heading performance when aided by an EKF demonstrates strong accuracy. The EKF maintains good tracking of the measured states during the constant yaw commands and shows minimal drift during steady-state periods. However, during rapid heading changes the measurement accuracy degrades impacting the algorithm's performance which can be seen in Figure 6.20 with the estimated state overshooting the measured state. To mitigate tracking errors, a viable solution would be to limit the UAV’s yaw rate to create smaller angle changes in the drone’s maneuvers or perform additional filter tuning to optimize performance. The adjustment improves estimation accuracy by aligning the UAV’s behavior with the EKF’s small angle approximation. The EKF shows good smoothing, settling time, and noise filtering while preserving the trend. Due to the sparsity of unique features in the test site, the visual feature tracking is degraded during rotations which cause the measurement discrepancies from the commanded states. To improve the feature tracking between frames, investigating motion blur and lighting effects could help identify limitations in the system.

Increasing the camera's frame rate, coupled with constraining the drone's angular rates have the potential to improve measurement accuracy through reducing motion blur and decreasing the distance features move between frames. Implementing the constraint would improve the performance of the system's ability to track the heading during all flight maneuvers.

The work presented in the paper demonstrates the feasibility of using a monocular VO algorithm for UAV navigation using low-cost hardware. The algorithm demonstrates the ability to operate in GPS-denied environments using only camera data and a downloaded map. However, further testing to evaluate edge cases, extreme environments, and hardware degradation to evaluate reliability should be done to further characterize and evaluate the performance. Optimization of the navigation algorithm could focus on hardware improvements and resolving scale ambiguity. For example, testing hardware with a dedicated GPU and larger storage would allow for the integration of CNNs into the algorithm and potentially produce more accurate or faster results. To address the feature sparsity, custom-trained NNs could be developed to handle feature sparse environments or seasonal changes to environments (currently limiting feature matching). Additionally, incorporating inertial data through sensor fusion with IMU and camera data would be the recommended approach to mitigate issues with scale ambiguity. A common method for sensor fusion for visual and inertial data is Visual Inertial Odometry (VIO) which compensates for scale ambiguity by directly providing the system with inertial data. Adding multiple cameras to enable stereo vision would be another method to resolve scale ambiguity. Using stereo vision would directly allow for integration of collision avoidance algorithms, creating depth maps, and enable 3D reconstruction capabilities. Extensions that would allow the algorithm to be implemented in real-world applications would be to implement collision avoidance, develop failure detection modes, and recovery systems (such as return to land using visual data rather than GPS data). The above enhancements could improve the navigation algorithms' performance and increase their application scope for UAVs.

References

[1] R. Abdelfatah, A. Moawad, N. Alshaer and T. Ismail, "UAV Tracking System Using Integrated Sensor Fusion with RTK-GPS," *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, Cairo, Egypt, 2021, pp. 352-356, doi: 10.1109/MIUCC52538.2021.9447646.

[2] Karlsson, Niklas & Bernardo, Enrico & Ostrowski, Jim & Goncalves, Luis & Pirjanian, Paolo & Munich, Mario. (2005). The vSLAM Algorithm for Robust Localization and Mapping.. 24-29. 10.1109/ROBOT.2005.1570091.

[3] D. Nister, O. Naroditsky and J. Bergen, "Visual odometry," *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, Washington, DC, USA, 2004, pp. I-I, doi: 10.1109/CVPR.2004.1315094.

[4] Gao, Xiang, et al. *14 Lectures on Visual SLAM: From Theory to Practice*. Publishing House of Electronics Industry, 2017.

[5] "Feature detection and description," *OpenCV* Available: https://docs.opencv.org/4.x/db/d27/tutorial_py_table_of_contents_feature2d.html.

[6] David G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, 60, 2 (2004), pp. 91-110.

[7] K. Qiu, T. Liu and S. Shen, "Model-Based Global Localization for Aerial Robots Using Edge Alignment," in *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1256-1263, July 2017, doi: 10.1109/LRA.2017.2660063.

[8] Restrepo, Carolina, and Andrew Johnson. "Terrain Relative Navigation Basics." Presentation at Workshop on Lunar Mapping for Precision Landing, March 2, 2021, with contributions from ALHAT, SPLICE, Mars2020, and OSIRIS-Rex. NASA Goddard and NASA JPL.

[9] Guan, Tianrui, et al. "AGL-NET: Aerial-Ground Cross-Modal Global Localization with Varying Scales." 2024, arXiv, doi:10.48550/arXiv.2404.03187.

[10] H. Oleynikova, M. Burri, S. Lynen and R. Siegwart, "Real-time visual-inertial localization for aerial and ground robots," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, 2015, pp. 3079-3085, doi: 10.1109/IROS.2015.7353802.

[11] Majdik, Andras, et al. "Air-ground Matching: Appearance-based GPS-denied Urban Localization of Micro Aerial Vehicles." *Journal of Field Robotics*, vol. 32, 2015, doi:10.1002/rob.21585.

[12] M. Blösch, S. Weiss, D. Scaramuzza and R. Siegwart, "Vision based MAV navigation in unknown and unstructured environments," *2010 IEEE International Conference on Robotics and Automation*, Anchorage, AK, USA, 2010, pp. 21-28, doi: 10.1109/ROBOT.2010.5509920.

[13] M. -M. Gurgu, J. P. Queralta and T. Westerlund, "Vision-Based GNSS-Free Localization for UAVs in the Wild," *2022 7th International Conference on Mechanical Engineering and Robotics Research (ICMERR)*, Krakow, Poland, 2022, pp. 7-12, doi: 10.1109/ICMERR56497.2022.10097798.

[14] Arora, Dhruv. *TAROT FY690S HEXACOPTER with DJI NAZA LITE*. YouTube, 20 May 2014, www.youtube.com/watch?v=rUEOCXNb5Hk. Accessed 21 June 2024.

[15] D. Scaramuzza and F. Fraundorfer, "Visual Odometry [Tutorial]," *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 80-92, Dec. 2011, doi: 10.1109/MRA.2011.943233.

[16] OpenCV, "Camera Calibration," OpenCV, Available: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html. Accessed: Jul. 14, 2024.

[17] OpenCV. "Stitching Module." *OpenCV Documentation*, 3.4, https://docs.opencv.org/3.4/d1/d46/group__stitching.html. Accessed 31 July 2024.

[18] D. Rose, "Rotation Quaternions, and How to Use Them," May 2015. [Online]. Available: https://danceswithcode.net/engineeringnotes/quaternions/quaternions.html. Accessed: Aug. 2, 2024

[19] Kalman, Rudolf E. "A New Approach to Linear Filtering and Prediction Problems." Journal of Basic Engineering, vol. 82, no. 1, 1960, pp. 35-45.

[20] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." *Advances in Neural Information Processing Systems*, vol. 25, 2012, https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[21] He, Kaiming, et al. "Deep Residual Learning for Image Recognition." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770-778.

[22] LeCun, Yann, et al. "Gradient-Based Learning Applied to Document Recognition." *Proceedings of the IEEE*, vol. 86, no. 11, 1998, pp. 2278-2324.

[23] Patel, Amit. "Deep Learning Course — Lesson 5: Forward and Backward Propagation." Medium, 31 Aug. 2020, medium.com/@nerdjock/deep-learning-course-lesson-5-forward-and-backward-propagation-ec8e4e6a8b92.

[24] Simonyan, Karen, and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." arXiv, 10 Apr. 2015, https://arxiv.org/pdf/1409.1556.

[25] Ren, Shaoqing, et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.* arXiv, 11 Dec. 2015, https://arxiv.org/pdf/1512.03385.

[26] Zhang, Aston, Zachary C. Lipton, Mu Li, and Alexander J. Smola. "Forward and Backward Propagation." Dive into Deep Learning, 2024, d2l.ai/chapter_multilayer-perceptrons/backprop.html.

[27] "MAVLink Developer Guide." MAVLink, MAVLink Developer Team, 2024, mavlink.io/en/. Accessed 25 Oct. 2024.

[28] "MAVSDK Documentation." MAVSDK, MAVSDK Development Team, 2024, mavsdk.mavlink.io/main/en/. Accessed 25 Oct. 2024.

[29] "PX4 User Guide." PX4 Autopilot, PX4 Development Team, 2024, docs.px4.io/main/en/. Accessed 25 Oct. 2024.

[30] "jMAVSim with SITL." PX4 Development Guide, PX4 Development Team, 2024, docs.px4.io/main/en/simulation/jmavsim.html. Accessed 25 Oct. 2024.

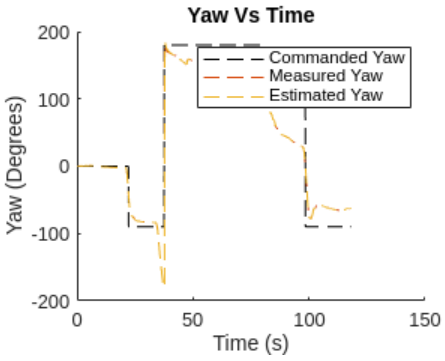
[31] "QGroundControl User Guide." QGroundControl, QGroundControl Development Team, 2024, docs.qgroundcontrol.com/master/en/. Accessed 25 Oct. 2024.

[32] "How GPS Works." Federal Aviation Administration, U.S. Department of Transportation, www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/gps/howitworks. Accessed 2 Dec. 2024.

Appendix A: Matlab Plotting Code

```
clc; close all; clear all;
Array=csvread('state_estimation.csv',1);
Array2=csvread('Untitled spreadsheet - state_estimation.csv',1);
time = Array(:,1);
measured_yaw = Array(:,7);
estimated_yaw = Array(:,16);
commanded_yaw = Array2(:,20);


hold on;
plot(time,commanded_yaw,'color', 'black','LineStyle','--')
plot(time, measured_yaw,'LineStyle','--')
plot(time,estimated_yaw,'LineStyle','--')
xlabel('Time (s)')
xlim([0 150])
ylabel('Yaw (Degrees)')
title('Yaw Vs Time')
legend('Commanded Yaw','Measured Yaw', 'Estimated Yaw')
hold off;
```



Appendix B: Code Access

https://github.com/cjmih/visual_waypoint_nav