Autonomous Landing of an Unmanned Aerial Vehicle on an Oscillating Platform

Undergraduate Project (UGP-II)

AE 471A

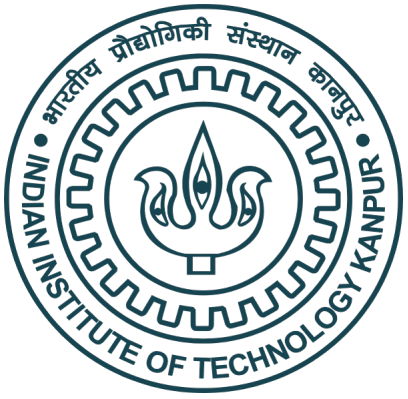
by

Rahul Rustagi

Under the Supervision of

Dr. Abhishek

Checked



DEPARTMENT OF AEROSPACE ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

April 2024

Certificate

This is to certify that the project entitled **"Autonomous Landing of an Unmanned Aerial Vehicle on an Oscillating Platform"** was submitted to the Department of Aerospace Engineering, IIT Kanpur, as an Undergraduate Project (UGP) is work done by **Rahul Rustagi**, under my supervision and guidance.

Dr. Abhishek

Professor

AE Department

Indian Institute of Technology Kanpur

Declaration

This is to certify that the report titled **"Autonomous Landing of an Unmanned Aerial Vehicle on an Oscillating Platform"** has been authored by me. It presents the undergraduate project conducted by me under the supervision of **Dr. Abhishek**.

Rahul Rustagi

Roll No. 200756

AE Department

Indian Institute of Technology Kanpur

Abstract

The objective of this project is to develop and demonstrate autonomous landing on a ship-deck platform using computer vision techniques. Such a task is primarily done by human pilots and which involves bias and accuracy difference. The work I show in this report involves a fundamental understanding of how vision can help achieve autonomy in this problem, what components are needed to implement it, and the results of integration of software with aircraft controls. The ship-deck motion dataset used is available for public use and meant for research purposes.

A quad rotor UAV with the required hardware is designed for the experimental testing of vision-based landing methods. A fiducial-based vision system is designed for detection and tracking of the moving ship-deck platform. A "tracking" controller is set up to always direct the UAV to hover directly above the landing position. During this stage, the UAV collects observations about attitude of the platform and its time-series. This data is used to decide optimal time for initiating landing protocol.

Experiments are conducted with static and sinusoidal motions to quantify the attitude tracking performance. The results show that it is possible to autonomously land on a ship-deck using computer vision alone. This type of vision-aided landing opens a window towards emulating the best of human training and cognition, without its burden of fatigue and divided attention.

Acknowledgements

I would like to express my deepest gratitude to my mentor Prof(Dr). Abhishek, for his constant support, guidance and assistance. He helped me in navigating through challenging topics and developing my research perspective. This experience and project would not have been possible without him.

I would also like to thank the lab members at Helicopter Laboratory for helping me with getting accustomed to the hardware used for experimentation. This project gave me an opportunity to work on my research interests in a proper oriented way.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Naval Helicopters

Ship deck landing is a risky, demanding helicopter maneuver which involves reaching the deck despite the heaving and rolling of the ship, airwake perturbances, and often stressful contexts. This motion could exceed the vertical agility of the operating helicopter, exceed the landing gear or engine torque limits, or increase the risk of a tail strike.

Attempts have been made by the government to install external devices on the ship that aid in landing of the helicopter or aircraft in general. We look at some of these methods that are currently in use to get a brief idea about touchdown on moving ship.

1.2 Advancements in Ship-Deck Landing

- **Advanced Arresting Gear (AAG)**: The U.S. Navy has implemented Advanced Arresting Gear (AAG) systems on its aircraft carriers. AAG uses an electro-magnetic energy absorption mechanism to provide a more controlled and adjustable landing experience for aircraft. Below is an image depicting the same. See Figure 1.1

- **Beartrap**: Helicopters are now a mainstay of naval operations, but only when the conditions on flight deck are favorable. In bad-weather or high sea-states, they remain grounded. In the late 1950s the Royal Canadian Navy invented a helicopter hauldown and rapid securing (HHRSD) device, known as beartrap. See Figure 1.2

FIGURE 1.1: F/A-18C landing on aircraft carrier, showing arresting cable

The bear trap system improves helicopter ship-deck landing in two ways:

1. It helps to keep the helicopter aligned with the center of the ship-deck when the helicopter is hovering over it.

2. It helps handling of the landed helicopter by securing it to the deck and preventing it from falling over

FIGURE 1.2: Helicopter being hauled down by a beartrap to land on the deck

Today, to enhance the safety and proficiency of ship-based aviation, there has been a growing emphasis on simulator-based training for pilots and deck crew, allowing them to practice complex landing maneuvers and emergency procedures in a controlled environment.

With the rise of unmanned aerial vehicles (UAVs), an autonomous ship-deck landing system is necessary for use of these aircrafts in naval aviation. These form the motivation behind the research to develop a vision-based autonomous system for vertical landing on ship-deck. The vision based approach to autonomously align the aircraft with the ship-deck seeks to replace the tethered wire approach of beartrap.

1.3 Can Computer Vision help?

In the field of Aerospace, Vision has found use in analysis of satellite images, inspection of aircraft parts, and in star trackers and earth sensors for satellite estimation determination and navigation.

Computer Vision algorithms analyse images and detects primitives such as color, lines and contours and uses them to detect more complicated shapes and objects. These detection techniques have been further improved over time to include Neural Networks (NN) for more complex detection at better accuracy. OpenCV is a common software that is used world-wide for executing and implementing computer vision algorithms on images. I have used the same software for image analysis.

Object Pose Estimation is one of the major revolutionary use-case that made Computer Vision so popular in Robotics.
Pose is a 6-degree state (3d position and 3d orientation) of an object denoted as [X,Y,Z,ϕ,θ,ψ].

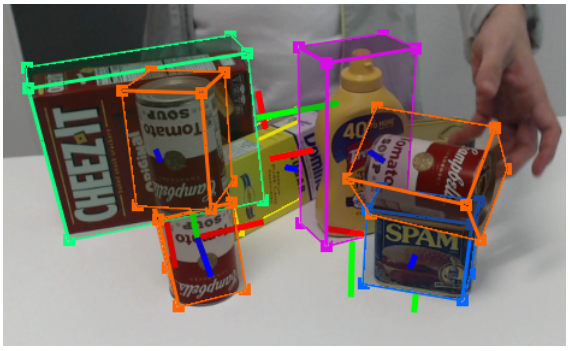


FIGURE 1.3: Detection and Pose Estimation of household objects

Observing the above image, one can see "bounding box" around each object that has been detected by the algorithm. 3 axes also emerge from the center of this box which essentially describes the 3d orientation of that particular object with respect to camera frame. Hence information about location and orientation is now known.

Below is an image of an ArUco marker (Refer Section 4 for more clarification) of id 203 (for unique identification) that can be easily detected and analysed for pose estimation due to its unique id and pre-defined orientation.

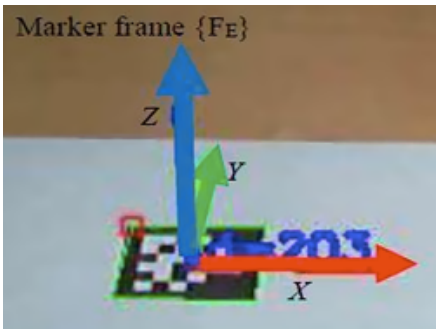


FIGURE 1.4: ArUco Marker ID: 203 Detected with XYZ Axes plotted

1.4 Vision for Ship-Deck Landing

The use of Vision for Landing is a popular area. As described above, by detecting objects and patterns in images, we can further analyse image to calculate pose of the detected feature. Once the pose is known, the UAV can be guided to that location by providing waypoints and landing can be safely achieved. The below figure depicts the same.

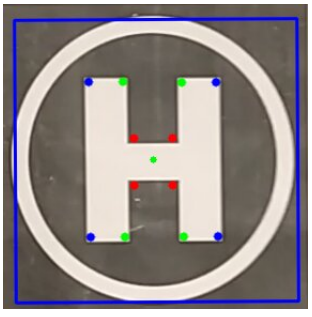


FIGURE 1.5: Helipad detection using Computer Vision for Landing

1.5 Autonomous Ship-Deck Landing

Now that I have established vision plays quite an important role for landing estimation, I further lay down the overview of the pipeline to be followed for initiating autonomous landing.

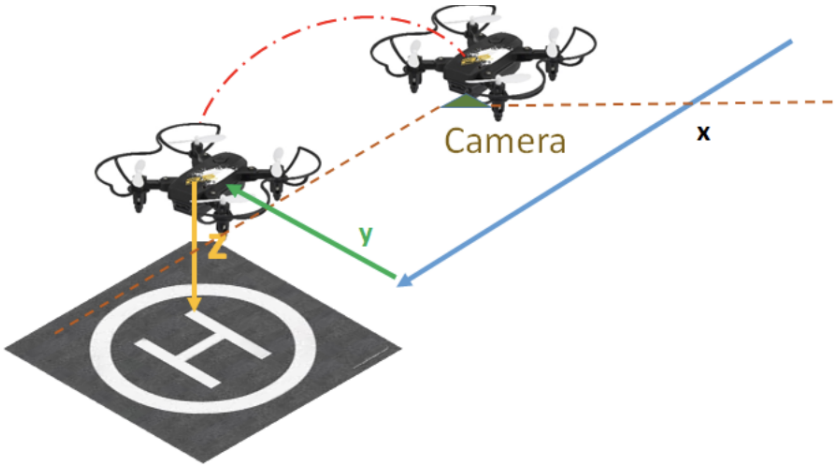


FIGURE 1.6: Protocol for Autonomous Landing

Issue arises due to continuously changing orientations and lateral motion of the landing platform. This would make touchdown impossible to execute safely as the platform with some angular velocity could disrupt safe landing and the UAV could topple. However, at one instant the orientations and motion would be within a certain "safe" threshold for a small duration during when touchdown can be achieved. The main idea is to initiate landing at a time when the landing platform is momentarily stationary or considered "safe" for landing.

It is quite important to decide this threshold for "safe" touchdown, the duration when landing can be achieved. This information can be extracted after analysis the motion of platform. This motivates us to know about sea conditions in which the ship is moving and the ship-wave dynamics involved. This is covered in the following section.

Chapter 2

Ship-Deck Motion

2.1 Types of Ship-Deck Motion

I provide a description of different types of ship-deck motion and sea-states, followed by a description of the deck motion dataset in this section.

Ship-deck motion is categorized into six types: 3 translations and 3 rotations. The translational motion along the longitudinal axis of ship is called surge, along the lateral axis is called sway and along the axis perpendicular to these axes is called heave, which is the raising or lowering of the ship-deck. Heave motion is of quite crucial in deciding landing. The rotational motions along the longitudinal axis is called roll, along the lateral axis is called pitch, and along the axis perpendicular to these is called yaw.

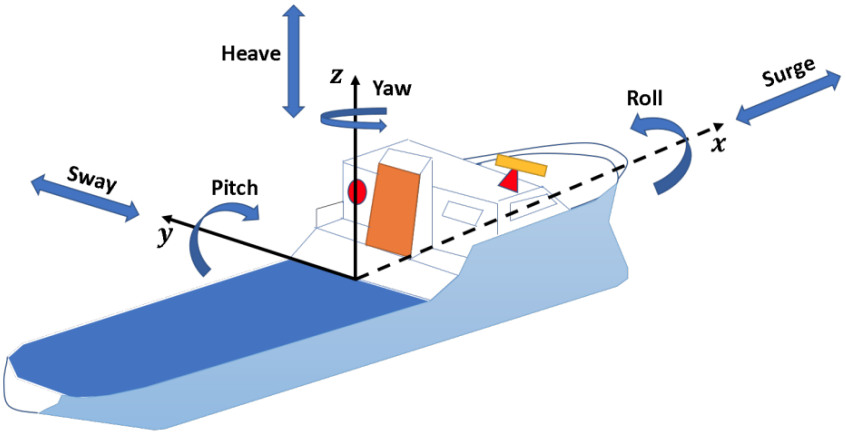


FIGURE 2.1: 6 Degress of Freedom in Ship-Deck Motion

2.2 Sea States

Sea States is a code from 0 to 9 that describes the roughness of sea taking into account wave height and wind velocity. The Douglas sea scale was devised to quantify the sea condition and help decide whether it is safe or unsafe for regular operations. Ship-deck motion is stochastic as the exact forces applied by sea-waves to the ship cannot be predetermined. However, statistical properties of these forces can be inferred from available datasets.

TABLE 2.1: Douglas Sea-scale

Sea State	Wave Height	Characteristic
0	0 m	Calm Glassy Sea
1	0-0.1 m	Calm Rippled Sea
2	0.1-0.5 m	Smooth Sea
3	0.5-1.25 m	Slight Sea
4	1.25-2.5 m	Moderate Sea
5	2.5-4 m	Rough Sea
6	4-6 m	Very Rough Sea
7	6-9 m	High Sea
8	9-14 m	Very High Sea
9	$>$ 14 m	Phenomenal Sea

2.3 SCONE Dataset

Systematic Characterization of the Naval Environment (SCONE) dataset is a publicly available dataset that models ship motion data. It is important to note that this dataset is prepared in a **simulated environment** using MATLAB and simulink. The SCONE data offer three levels of deck motion intensity in the heave and roll axes.

2.3.1 SCONE Data Description

Every file within the SCONE database contains a set of data which describe different ship deck motion with respect to roll intensity and heave rate intensity of the ship. The roll intensity and heave rate intensity of the ship are described as "low", "moderate" and "high" conditions. For each condition, five simulations (referred to as 'realizations') were executed with differing, random wave phases. Thus, there are five 30- minute time histories for each deck motion condition (sampled at 50Hz).

Each file are MATLAB files with 19 columns of data and a row for each time step. The contents in a file are explained below:

TABLE 2.2: SCONE Dataset Description

Row No.	Quantity	Dimension
1	Time, 0.0 at start of recording period	sec
2-4	X, Y, Z coordinate of the ship's flight deck reference point	ft
5-7	Roll, Pitch, Yaw angle - Euler rotation about the X, Y, Z axis	deg
8-10	Surge, Sway, Heave velocities in global coordinate system	ft/sec
11-13	Roll, Pitch, Yaw rates in ship-fixed coordinate system	deg/sec
14-16	Surge, Sway, Heave acceleration in global coordinate system	ft/sec^2
17-19	Roll, Pitch, Yaw acceleration in ship-fixed coordinate system	deg/sec^2

TABLE 2.3: SCONE Dataset Nomenclature

Wave Characteristics (m)	Douglas Sea State	Ocean Waves	SCONE levels
0.0 to 0.5	0,1,2	Low	1
0.5 to 4.0	3,4,5	Moderate	2
4.0 to 6.0	6	High	3

SCONE dataset files are named in the following fashion: "SCONE_DXR_Y.mat" or "SCONE_DXH_Y.mat". Here X varies from 1 to 5 showing different experiments conducted for making dataset and Y varies from 1 to 3 to depict the intensity level of either the H (Heave) or R (Roll).

2.4 Ship-Deck Hardware used for Landing

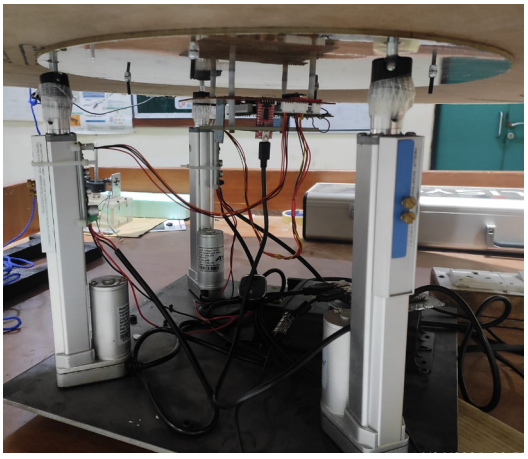


FIGURE 2.2: Under-actuated model of the Stewart Platform

A plate is attached to the end-effectors of these three actuators which moves as per the net motion imparted to it.

Another plate (big enough for the quadrotor to land) is attached on top of this plate. This custom "big" plate has a ArUco marker (Refer Section 3 for more clarification) attached on top of it. This marker is detected by the drone vision system and directs itself towards it for safe landing.

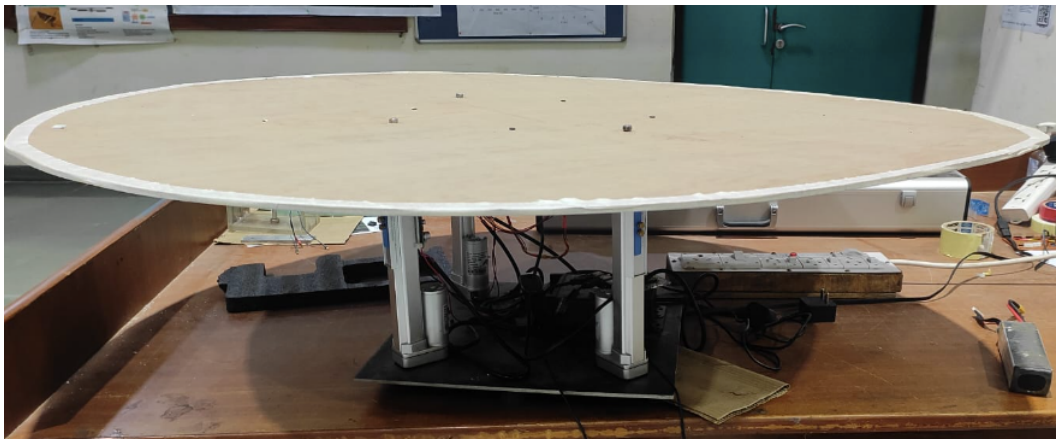


FIGURE 2.3: Drone Landing Platform built in the lab

The platform was provided to me to test my experiments. I have just used it and not designed it. I have performed my tests on this under-actuated (as only roll, pitch and heave motions are ideally possible) platform for demonstrating landing of my UAV autonomously using my algorithm while the platform simulates ship motion.

2.5 Hardware Limitations

Limitations currently in the platform are:

- The system as stated above is under actuated due to which 3 of the motions (surge, swell and rise) cannot be tested.
- Heave motion cannot be tested due to the limited stroke length of the actuator. Currently, these linear actuators have a 6 inch stroke meaning they can move a total of 15 cm. Heave coupled with roll and pitch simultaneously constraint the range of motion more in the z direction. Hence testing for heave platform could not be conducted.

Chapter 3

Specifications of the UAV Used

3.1 Overview

In this section, I discuss about the VTOL UAV I used to collect experimental results. The UAV was assembled and provided to me by the Helicopter lab members.

3.2 Vision System

The Quadrotor is attached with a Vision system to be able to take image feed as input for analysing it further. This vision system is a camera attached to the base of the drone.

The camera is Logitech C922 HD Pro Webcam. It is equipped with automatic light correction, it adjusts to the current lighting conditions, producing bright, contrasted images, even if the environment is currently a dim setting. Therefore, parameters like exposure and white balance would be tuned accordingly. Hence it fits for this project.

Below mentioned are the relevant specifications of the camera:

- Resolution: 1080p/30fps or 720p/30fps.
- Field of View: 78°

FIGURE 3.1: Vision System used in the drone

3.3 Companion Computer

A Onboard Computer is used on the vehicle that is primarily used for carrying out processor-intensive tasks like analysing images. The computer used is Raspberry Pi 4 model B. Specifications are as below:

1. RAM - 4GB DDR4
2. 1 USB 3.0 port
3. Operating System (OS): Linux Server

Its primary purpose is handle detection and pose estimation of markers. Its secondary purpose is data logging of flight variables (.rosbag) for debugging and post-processing. Different processes running on top of the OS are used for executing different tasks in parallel. These tasks are reading and processing the data from the camera sensor, executing algorithm commands, managing communication with the flight controller, and recording flight data to non-volatile memory for post-processing.

FIGURE 3.2: Raspberry Pi Computer used for Online processing of real-time data

3.4 Assembled UAV

In the picture attached, all the avionics and necessary components needed for flight are attached. Along with this the onboard computer and the vision system is also attached.

FIGURE 3.3: Assembled UAV

The camera is attached facing downward to capture the marker in head on position. Since previously I transformed the Marker Frame to coincide with the Image Frame, the frames coincide when the drone is stationary and aligned with the marker. This is useful in Roll Pitch estimation algorithm.

Some standard specifications:

- Flight Computer: PixHawk Cube Orange
- PX4 Build Version: 1.12.0
- GPS: here3
- Battery: LiPo 6S 3300 mAh
- Motor: T-Motors MN4006-23; 380kv (X4)

Chapter 4

Fiducial Marker & Vision System

In this section, I describe how the vision system is integrated with drone to take in the desired input feed as required. This is followed by a introduction to Fiducial markers which help in guiding the drone to itself by letting the drone extract its pose. The customisation of the library used here is a part of my estimation algorithm.

4.1 Camera System Used

As described above, the camera is Logitech C922 HD Pro Webcam operated at a resolution of 720p and 30fps.

4.1.1 Camera Calibration

Since the camera is operated to capture 3d objects but outputs a 2d image, this results in a mapping. Such a 3d to 2d mapping requires calibration of the lens parameters for correct and proportional projection of the input image onto image plane. This can be visually understand by looking at the figure below:

4.1.2 Integration with ROS

Since the camera is an independent system capturing the feed and the drone has ROS server running on its companion computer, they need to communicate through a common channel inorder to reduce latency.

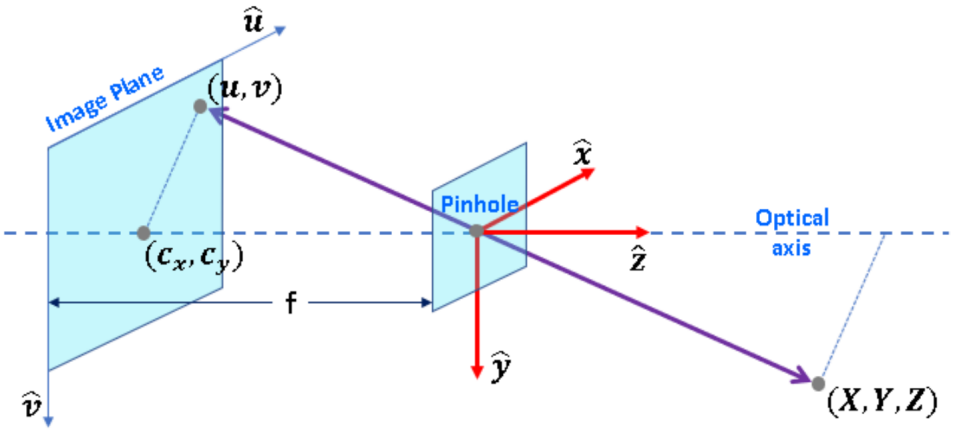


FIGURE 4.1: Pinhole camera model

I have used ROS (Robot Operating System) for calibrating the camera. ROS provides libraries to efficiently interact with the video feed system. In the backend, the video feed with the set parameters will be available on the ROS server for other processes to use (like OpenCV). Below is the code that takes in values of the necessary arguments as discussed above.

```
rosrun camera_calibration cameracheck.py
--size 9x7
camera:=/usb_cam
image:=/usb_cam/image_rect
```

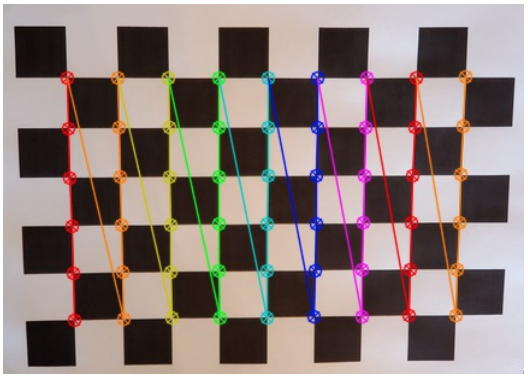


FIGURE 4.2: Checkerboard calibration

After calibration, we get the **Camera Matrix** that specifies where the coordinates of the optical centre (in pixels) and the focal length in X and Y directions (in pixels). The camera model assumed is pinhole which is not the case generally, hence we also obtain a distortion matrix to cancel the effect of distortion on the final image analysis.

```
image_width:1280
image_height:720
camera_matrix: !!opencv-matrix
 rows: 3
 cols: 3
 dt: f
 data: [9.497770800818170756e+02,0.00000000000000000e+00,6.740236625926454508e+02,     [fx  0 Ox]
        0.00000000000000000e+00,9.523650293344279589e+02,3.704231968017428471e+02,     [0  fy Oy]
        0.00000000000000000e+00,0.00000000000000000e+00,1.00000000000000000e+000]    [0   0  1]
distortion_coefficients: !!opencv-matrix
 rows: 1
 cols: 5
 dt: f
 data: [1.213969947315009096e-01,-4.901896702003197964e-02,-5.746435130429767939e-03,
        1.501020385820848492e-02,-2.230235807979541185e-01]
```

FIGURE 4.3: Obtained Camera Parameters

4.1.3 Camera Focus

Different autofocusing methods exist for many cameras today. The camera used for this project has an autofocus feature. Autofocus can indeed pose challenges in robot vision applications, especially in dynamic environments or when dealing with varying distances between the camera and objects of interest. Here are some common problems associated with autofocus in robot vision:

1. Speed: Traditional autofocus mechanisms may not be fast enough to keep up with rapid changes in the scene, leading to delays in image acquisition and processing.

2. Accuracy: Autofocus algorithms may struggle to accurately determine the optimal focus point, particularly in scenes with complex or cluttered backgrounds. This is one of the critical reason for turning off autofocus as it hinders with clear input images which an lead to possible poor detection (due to loss of resolution in blurry images) and eventually failing of the pipeline.

Due to the above reasons, autofocus is turned off.

4.2 Fiducial Markers

4.2.1 ArUco Marker

ArUco markers are visual markers used in computer vision for augmented reality applications. These are square or rectangular patterns with a black border and a unique binary

interior pattern. They are simpler in design compared to fractal markers but still offer good detection reliability. ArUco markers are widely used for their simplicity and ease of detection. They can be quickly recognized in images or video frames using standard computer vision algorithms, making them suitable for real-time applications.

While ArUco markers offer good detection reliability under normal conditions, they may be more susceptible to occlusion and partial obstruction due to their simpler design. As the drone starts to lower its height during landing on the platform, the visibility of ArUco marker reduces and if even one corner goes outside frame then detection fails which leads to loss in waypoint generation. Hence, until the drone lands, the marker should be visible all time. This is where fractal markers help.

4.2.2 Fractal Marker

Fractal markers are a new concept of marker, which is composed of several fiducial square markers of different size inside. Unlike traditional fiducial markers, the structure of this marker can be detected from a large number of distances, as well as solve problems of partial or total occlusion of the marker.

The features of this marker, together with the tools developed make it a powerful tool for camera pose estimation in a large number of applications such as robots, unmanned vehicles and augmented reality.

Fractal Marker is integrated inside ArUco's libraries, allowing a fast, robust and precise detection of the markers. ArUco is a widely used OpenSource library for detecting squared fiducial.

4.3 Pose Estimation

Fractal Marker System uses ArUco to perform markers detection. The process of calling ArUco is completely transparent to the user, and carried out using the FractalDetector class.

If the extrinsic parameters of the camera are known (camera matrix and distortion coefficients obtained by calibration), then we can calculate the relative position of the fractal marker and the camera. The system will give you the rotation and translation vectors

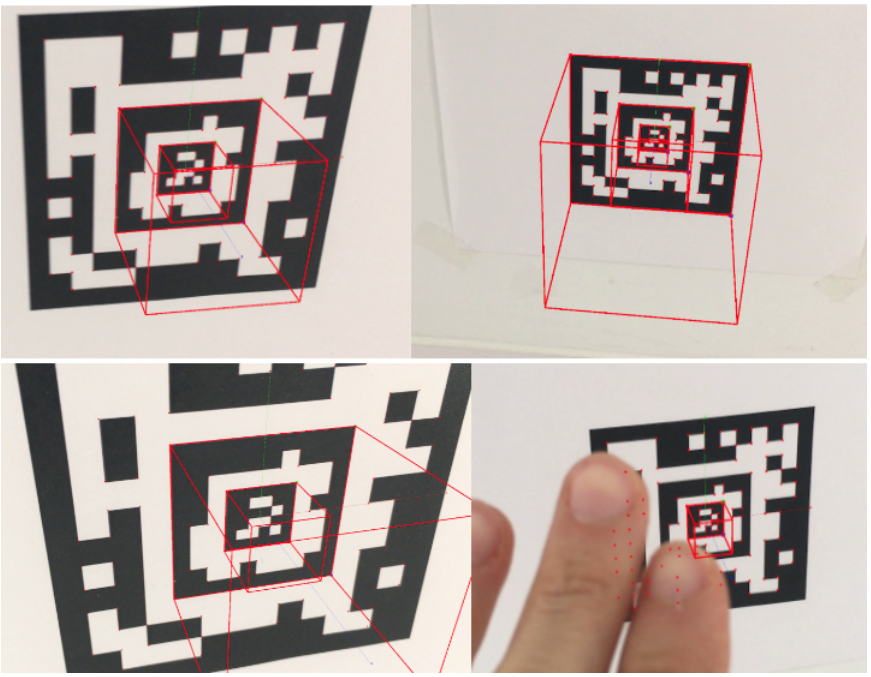


FIGURE 4.4: long-range stable detection under occlusion

(**Rvec** and **Tvec**), which represent the transformation matrix from the marker frame to the camera frame.

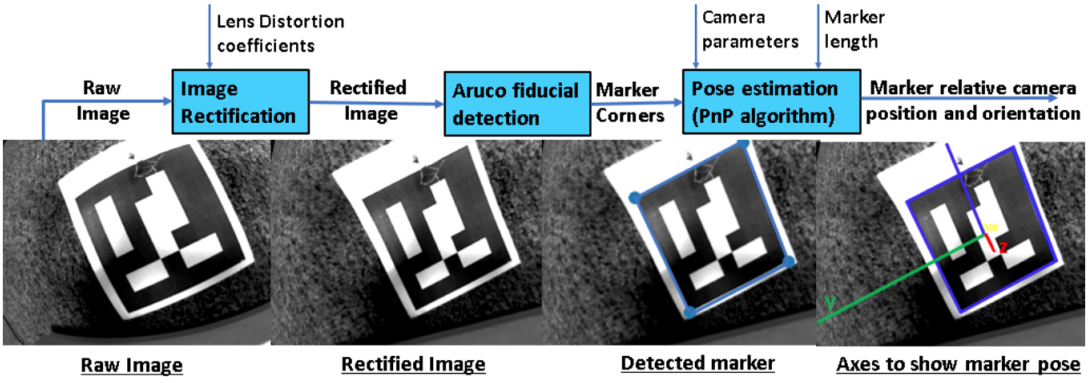


FIGURE 4.5: General pipeline of pose esitmation for an ArUco marker

4.3.1 Understanding Rvec and Tvec

rvec refers to the rotation vector, which is a compact representation of a rotation in three-dimensional space. Instead of using matrices to represent rotations (such as rotation matrices or Euler angles), the rotation vector uses a single vector. Commonly known as

the axis angle representation.

$$r = [a_1, a_2, a_3]; \sqrt{{a_1}^2 + {a_2}^2 + {a_3}^2} = \theta$$

Rotation Angle (θ): This is the magnitude of the rotation, typically measured in radians. It indicates how much the object is rotated about the axis of rotation.

Rotation Axis (e): This is a unit vector (a vector with a length of 1) that represents the axis of rotation. The direction of this vector determines the direction of the axis, and its magnitude gives the rotation in radians.

rvec requires only three parameters compared to the three angles needed for Euler angles, making it more compact and efficient for storage, computation, and transmission. rvec representations are less prone to singularities like gimbal lock, ensuring numerical stability and robustness.

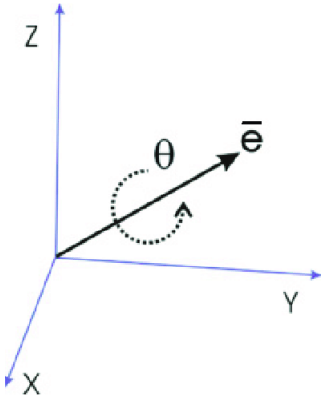


FIGURE 4.6: Axis-Angle representation

4.4 Euler Lock and Quaternions

4.4.1 Gimbal Lock and its prevention

Gimbal lock is a mathematical problem that arises only when Euler angles (Roll/Pitch/Yaw) are used to represent 3D orientation. More specifically, it occurs when the X-axis of points straight up or straight down, i.e. Pitch = +-90 deg. In this orientation, the Roll and Yaw angles will become mathematically unstable. The phenomenon is called gimbal lock because the Roll and Yaw axes of a gimbal are "locked" together in this orientation; they are both affected by a rotation around the X-axis of the rotating body.

ϕ, θ and ψ are euler angles which is the input to our platform. Hence, I was converting marker orientation from quaterion to euler since, I was able to directly map the difference and determine if platform is suitable for landing. However dealing with them is not advisable, so I wrote the algorithm to work in axis-angle notation.

4.5 Transformation from Marker Frame to Image Frame

In OpenCV, the image has a ESD (East-South-Down) frame orientation. However the Fractal library which detects the marker, presents the marker pose in ENU (East-North-Up) frame orientation.

However for accurate and simple conversions, I wrote a function in the FractalDetector class to change the orientation of the marker from ENU to ESD (Image frame). This would help understand the intuition behind Roll Pitch estimation algorithm.

The algorithm calculates the orientation axes using detected points in order. However, if we tweak the ordering of the detected points that go into the *SolvePnP* algorithm, we can get the desired orientation.

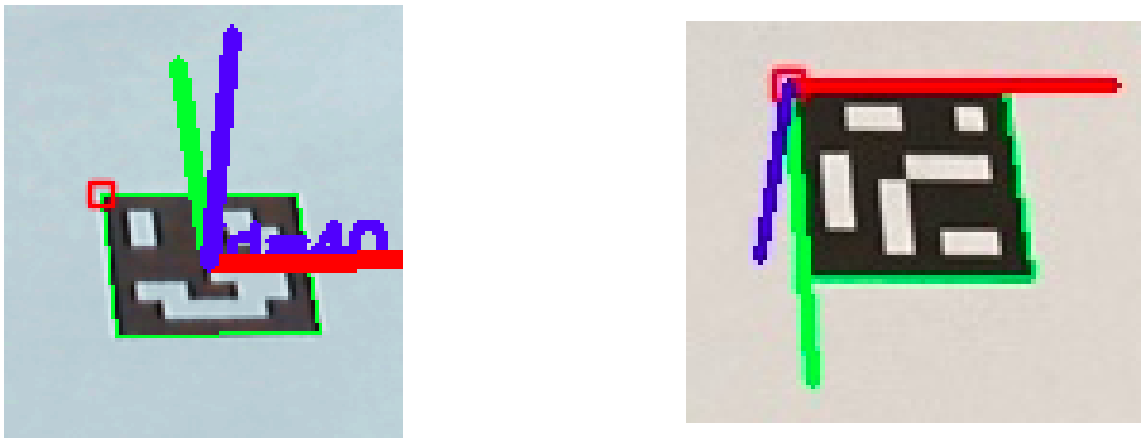


FIGURE 4.7: [Left]: Axes in ENU frame before conversion. [Right]: Axes in ESD frame after conversion

4.6 Discussion

Below are some plans that need to be done for improving overall performance:

- Upgrade to a better camera system with 720p/60fps so that the frequency of the pipeline increases.
- Currently, the camera is rigidly attached to the drone. Attach the camera to a gyroscope so that any perturbations faced by the drone mid-air do not induce error in analysis. As can be seen in the figure below, even a small jerk in camera (due to random instability in mid-air) will magnify the position estimate by a magnitude of Z times the perturbation. Where Z is the current distance between object and camera (assuming camera is downward facing (nadir)).

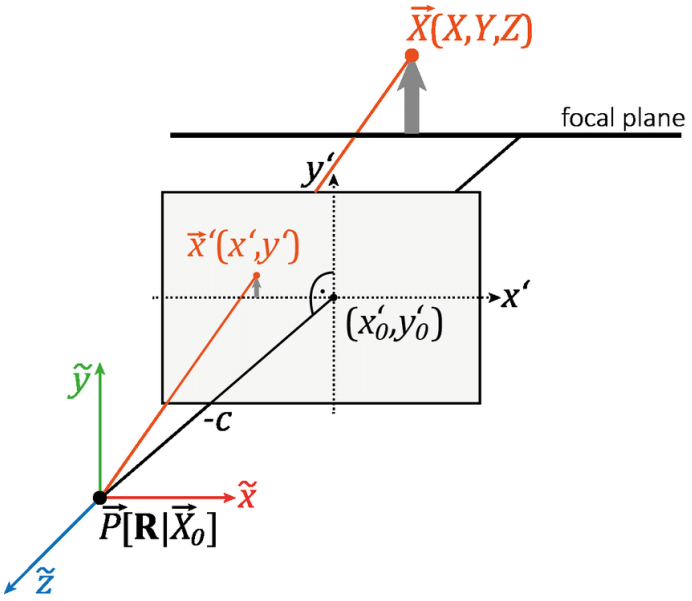


FIGURE 4.8: Magnification is proportional to the distance between image and object plane

Chapter 5

Roll and Pitch Estimation

5.1 Motivation

I present a method to estimate the roll and pitch motion of a vessel using solely vision equipment without requiring any vessel-specific parameters or models. The motivation is that many marine applications require precise and safe landing for the vessel to avoid any damage to the aircraft. Here I have not introduced heave motion and tested only on roll motion (due to actuator limitation)

5.2 Algorithm for Roll and Pitch Estimation

To talk about the algorithm, it is quite straightforward. The UAV is hovering just above the platform and we want it to land on. After getting a few observations, the algorithm decides when to land such that it does not topple.

At each iteration of algorithm, given the detection is successful, the Fractal library extracts pose of the detected marker and conveys it to the algorithm. The algorithm calculates the angle deviation of the platform with respect to the UAV and outputs the angle. This angle if falls within a threshold directs that the platform is suitable at this moment to land and so the UAV moves down. This is repeated at each time step and if the angle calculated falls above the threshold, then the UAV is not commanded to lower down instead just hover at the location.

FIGURE 5.1: [Left]: UAV hovering above the marker to collect observations [Right]: Camera feed conversion

Algorithm 1 Roll Pitch landing psuedo code

Require: Marker Detection is successful in current frame
Ensure: UAV is stable and camera facing completely down
while $Marker\_Pose$ is updated **do**
$Rot \leftarrow Marker\_Pose.Rvec$
$Tran \leftarrow Marker\_Pose.Tvec$
$Angle \leftarrow \sqrt{Rot[0]^2 + Rot[1]^2 + Rot[2]^2}$
if $Angle \leq abs(Threshold)$ **then**
$Intiate\_Landing \leftarrow True$
else
$Intiate\_Landing \leftarrow False$
end if
end while

The threshold for toppling is set to around **12**° through experimentation for this particular frame.

This algorithm reduced to a simple angle check as during detection I have already transformed the marker axes to coincide with the camera image axes. Due to this modification, I just need to calculate the angle from the **Rvec** vector and check if that falls below the threshold.

5.2.1 Proof of Concept

I am now going to state down reasons why this algorithm would not fail for any pattern of roll and pitch signals given.

Assuming that currently the platform is exhibiting only roll and pitch motions, we can make a few deductions based on this:

1. At any instant, when the pose is calculated and returned in axis-angle format, then the axis will always lie in the same plane as that camera frame (given camera is horizontal and stable throughout).
2. As the axis lie in the plane parallel to the camera frame, the angle that is calculated is precisely the deviation of the platform from the horizontal plane. This is what we need to set the threshold on.

Hence, the algorithm works in this case. During touchdown the angular velocity with which the platform is exhibiting motion can also be checked to ensure safe and less impactful landing. However, the ship data that I used for demonstration is of quite low-frequency; around 0.2 to 0.3 Hz. The command to initiate landing runs at 25 Hz, nearly 100 times faster. Hence, there should not be any problem.

5.3 Protocol for Safe Landing

The UAV is lowered by generating a waypoint with Z coordinate reduced by 0.02m keeping X and Y same. As the flight controller module runs at approximately 50Hz, reducing height by 0.02m results in a smooth execution.

This method is straightforward. I am calculating the position setpoint and sending it to the UAV from the companion computer.

However, here the accuracy can be improved further by leveraging the communication protocol of the UAV system. I have discussed this point in Sec 5.5.

Below, is the algorithm for landing once $Initiate\_Landing \leftarrow true$

Algorithm 2 Landing Protocol

Require: $Initiate\_Landing \leftarrow True$
Ensure: Current Position is fetched from UAV
  $Set\_Pose \leftarrow 0, 0, 0$
  $Curr\_Pose \leftarrow 0, 0, 0$
  while $UAV.Current\_Pose = Updated$ **do**
    $Curr\_Pose \leftarrow UAV.Current\_Pose$
    $Set\_Pose.X \leftarrow Curr\_Pose.X$
    $Set\_Pose.Y \leftarrow Curr\_Pose.Y$
    $Set\_Pose.Z \leftarrow Curr\_Pose.Z - 0.02$
    $Publish(Set\_Pose)\ to\ UAV$
    if $Landed = true$ **then**
      $EXIT$
    else
      $Continue$
    end if
  end while

5.4 Results

I tested my algorithm for different orientations of the platform as described below.

5.4.1 Experiment 1: Sudden Discrete Change in Elevation

FIGURE 5.2: [Left]: Platform at 22° elevation from horizontal [Right]: Platform at 0° elevation from horizontal conversion

This is a simple experiment where the platform is initially at an elevation of 20 - 25 ° from the horizontal. The UAV starts hovering above the marker and takes in observations. As the current angle is above the threshold set, the UAV would not land and rather hover i.e. $Initiate\_Landing \leftarrow false$. Immediately I reduce the elevation to 0 °. Now the algorithm sets $Initiate\_Landing \leftarrow true$ and drone starts to land. Below are the plots demonstrating the same.

Below are the results attached:

Here we see that $Marker\_Visible$ variable value changes from 0 to 1 as soon as marker is visible.

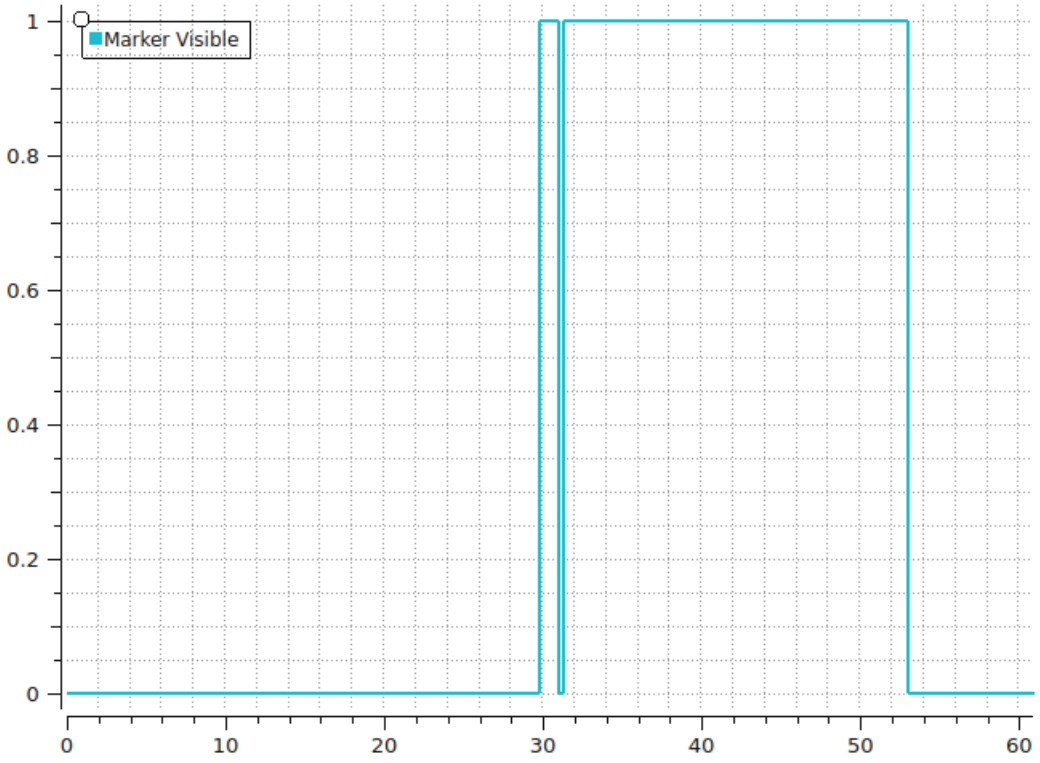


FIGURE 5.3: Initiate_Landing variable

Here we see the angle recorded by the drone. Notice the 5° offset.

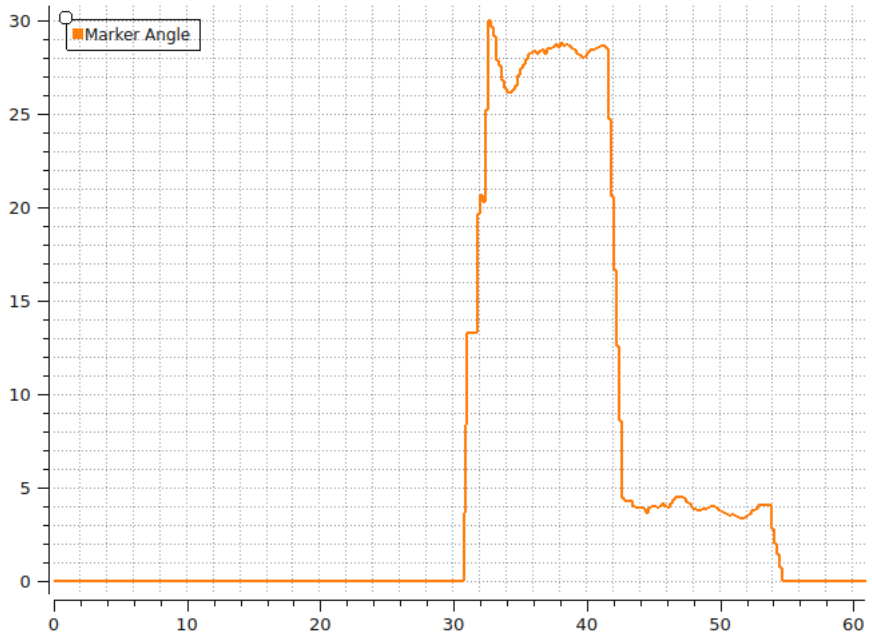


FIGURE 5.4: Platform Orientation as recorded by Drone

Here we see that as soon as percieved angle changes from 30° to 5°, the drone starts landing and we can see the change in z position (green curve). Note that the Z value is in negative as the UAV follows NED notation in which Z is positive downwards. Hence ascent results in a negative value and descent in a positive.

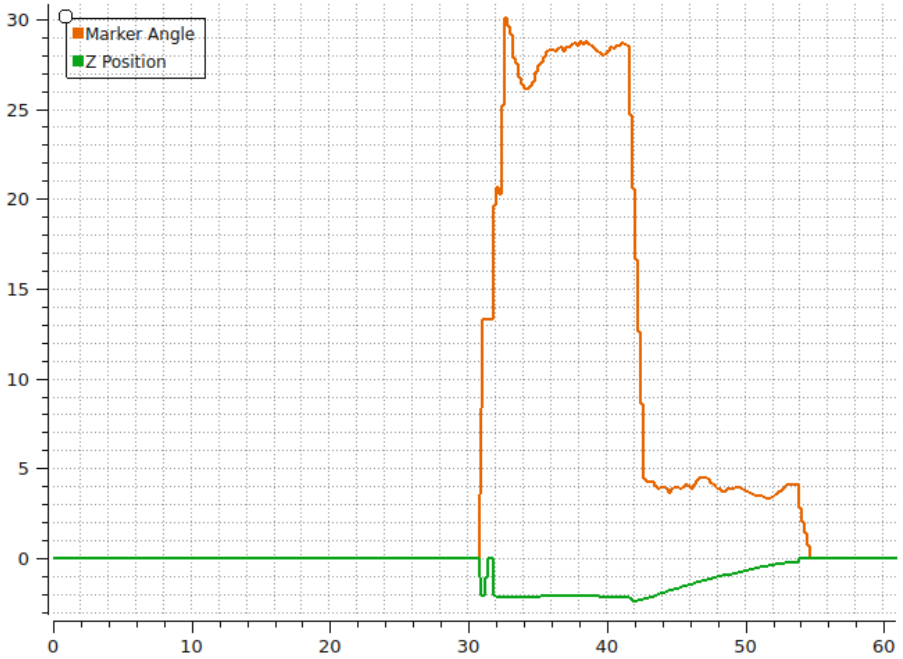


FIGURE 5.5: Landing trajectory as marked by the green curve

5.5 Experiment 2: Ship Roll motion on the platform

This experiment would be having continuous ship data simulated on it. Here, the ship roll data would be added onto the platform. Note that the maximum roll is around 10°. Unfortunately while hovering the drone crashed brutally breaking the leg and propeller of the drone and due to lack of available hardware, experiment was not performed. This experiment would hopefully be done until the presentation date and I would have the results for this by then.

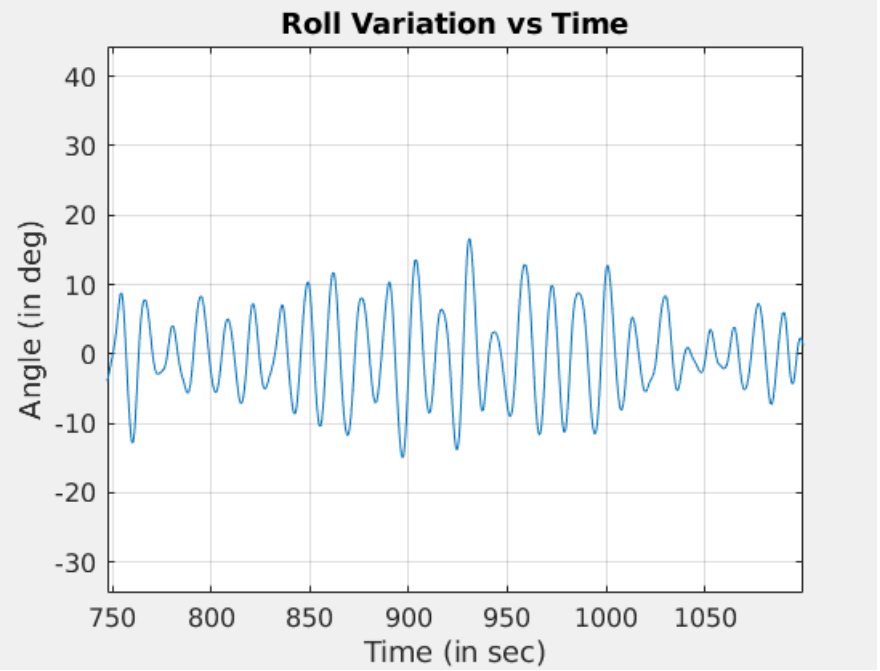


FIGURE 5.6: SCONE_D2R_3 File of Dataset

However, I have set the landing threshold for angle to be around 5° considering the roll rate and angle effect during touchdown.

Below is the input to the platform:

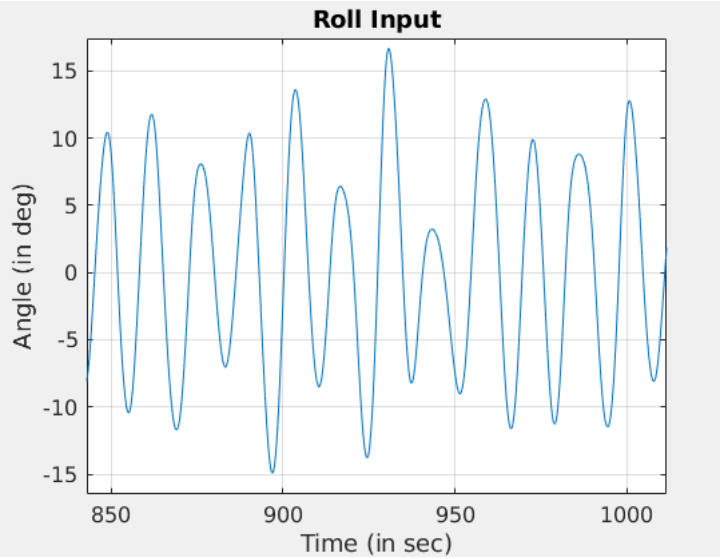


FIGURE 5.7: Input Roll to the Platform

I was able to collect what the drone is perceiving the ship motion as so as to give us an idea of efficiency. Here is that plot:

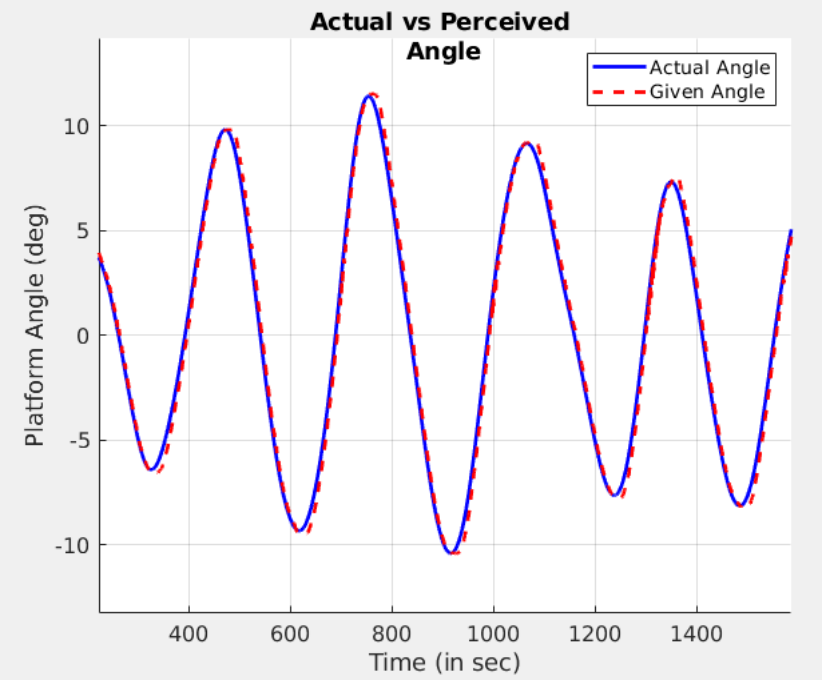


FIGURE 5.8: Comparison between Actual and Perceived Platform Angle

5.6 Improving Performance

This is section is quite important to run the algorithm at high frequency and so is optimal for execution of the pipeline. I discuss here the issues in above implementation with respect to UAV communication framework and how latency introduced can be reduced by building code at the firmware level instead of running it in the companion computer.

5.6.1 Bottlenecks in performance

Firstly, the image analysis code has to run on the companion computer. However, once the *Initiate_Landing* signal is set true, the new waypoint as described above is generated and published to the flight controller, which then commands the navigation module of the UAV to follow it.

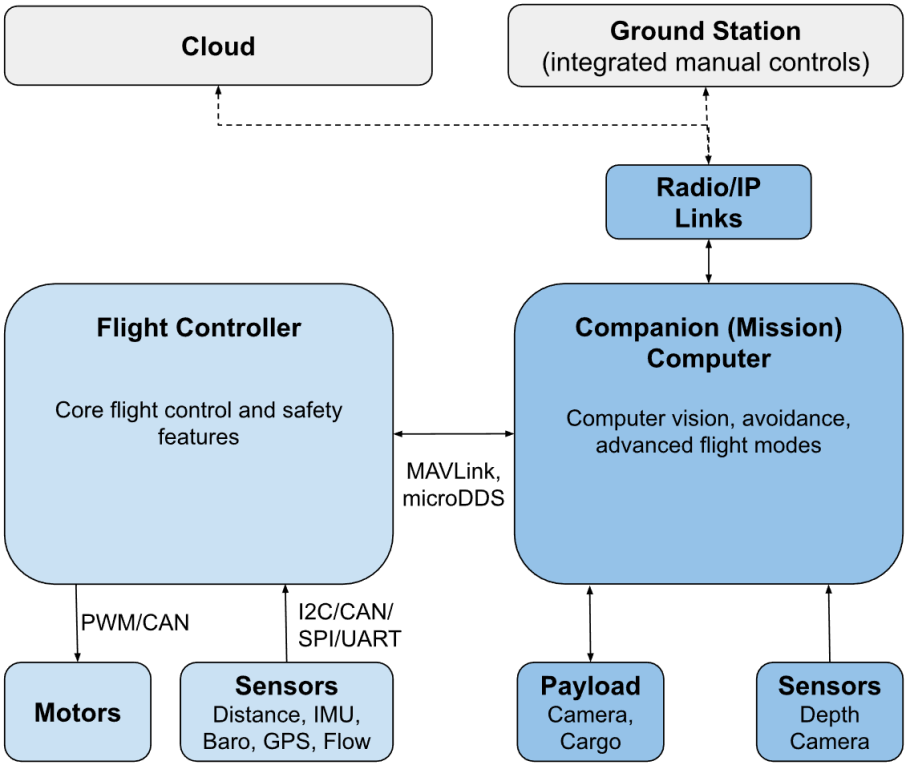


FIGURE 5.9: Possible architecture for an unmanned vehicle

Since the UAV communication system uses a serial communication protocol, messages are serially queued and sent to the modules. There is serial communication (MAVLink protocol) between Pixhawk and the offboard computer ass can be seen in the image attached

above. This introduces a lot of latency as the commands are generated in the onboard computer and sent to the controller from outside.

5.6.2 Generating Landing Commands in Firmware Code

Latency is introduced due to the serialised nature of communication. To resolve that, I build the code at the firmware level. The idea is that the flight controller is working on Pixhawk computer which has a very powerful microcontroller and processor (as it handles many calculations for stabilising the UAV). The flight controller is a controller that is calculating next waypoint based on current inputs, observations and the set reference value. Currently, the reference value is set and communicated by mission computer. This can be instead written in the firmware of the flight controller to switch the reference if it receives that $Initiate\_Landing$ is set to true.

Note that the results plots generated above are considering the improvements.

5.6.3 Always hovering above Marker

The UAV is first manually made to hover above the marker, and then allowed to reposition itself above the center of the marker. Here, I talk about the problem of matching X and Y coordinates of the UAV with that of the Marker incase the UAV might sway due to wind.

The X and Y position of the Marker obtained from the **tvec** vector is fed to the flight controller for repositioning. One inaccuracy that might seep into this is due to GPS error which can be large at times. Hence instead of allowing drone to get its feedback position from the GPS and then redirect to another X Y position (that of the marker), I directly set the X and Y position of the UAV relative to the marker (when the marker is visible only then). Hence, now the control problem has changed from redirecting drone from (X_{UAV}, Y_{UAV}) to (X_{marker}, Y_{marker}) to redirecting drone from $(X_{relative\_pose\_from\_marker}, Y_{relative\_pose\_from\_marker})$ to (0,0). It is essentially the same problem but better as GPS inaccuracies in postion feedback is removed.

Chapter 6

Discussion

6.1 What is this section?

In this section, I want to discuss the error and difficulties faced during implementation of the project. As my role was predominantly building the software and trying to execute it on hardware, I faced some integration issues in getting the expected accuracy.

6.2 Motion Capture and Pixhawk Integration

As the drone requires an positioning system for localising itself, we use GPS to enable it to estimate its own pose. When the drone is powered on, the GPS module tries to "lock" itself with as many satellites it can so as to ensure better and stable pose feedback. However, such hardware testing is unfeasible to be conducted outside everytime. Indoors there is no GPS lock due to obstruction. Hence an external indoor positioning system is used in the laboratory for conducting experiments which has better accuracy than GPS feedback.

I used the Qualisys Motion Capture system present in the laboratory to provide pose feedback to the drone. However, even after carefully following the documentation for integration of pose feedback from external vision, the drone was not able to get a position feedback for more than 90 seconds.

This problem was occurring with the Pixhawk autopilot and not with ardupilot system. This took a lot of time for debugging and I was able to increase the maximum flight time in position mode from 90 sec to 200 sec, but the problem persited.

6.3 Hardware Actuator Limitation

This was a major bottleneck for continuing the experiments. The linear actuators attached to the platform were having 2 major issues:

1. There was no z-feedback provided to the actuator control system during roll and pitch motion. This resulted in the actuators failing and falling down to minimum stroke position.
2. The stroke length of the current actuators is 15cm. This restricts performing any possible experiment of estimating heave motion as 15 cm (when ideal) is a lot smaller than actual conditions of even the calm sea. Hence non-dimensionalising the parameters like angular velocity and acceleration reduces them by a very big factor. Hence actuators of bigger stroke length are needed.

New actuators were ordered two weeks ago and they are yet to arrive. This is why heave testing and implementation were not possible.

6.4 Drone Frame Optimisation

1. The legs of the drone are too long which increases the chances of slippage and toppling during landing on a non-stationary platform. Hence, the drone is not optimised with respect to the frame. Along with this, a lot of extra wires and empty space that affect the positioning of the CG and add on extra weight.

Chapter 7

Conclusions

Here I discuss the summary of the project. A new feature-based vision system was been developed and experiments were performed to quantify performance of the vision system to understand its practicality. Based on the results shown following key conclusions are drawn:

1. The fiducial based vision system can make precision landing possible on a roll pitch platform with good accuracy. The only requirement is that the platform should have well lit area such that marker is visible.

2. The Fractal markers can tolerate 80%-90% occlusion instead of the ArUco fiducial system which cannot tolerate more than 10% occlusion. Fiducial-based vision system is accurate in its measurement of position and orientation of the landing pad.

3. Roll or Pitch landing is possible and can be implemented on a large scale by using a fiducial marker system. Hence any perturbations in sea that result in change of orientation of ship deck can be handled using the landing protocol logic and is implementable by using a fiducial based system.

Hence, it is possible to automate and achieve precision landing of an Unmanned Aerial Vehicle on an oscillating platform with a good accuracy.

Bibliography

[1] C. I. Abhishek Shastry, Datta Anubhav, "Towards autonomous vertical landing on ship-decks using computer vision," ., vol. ., pp. 1–45, 2022.

[2] O. H. Mehling, Tim Halbe, "Piloted simulation of helicopter shipboard recovery with visual and control augmentation," *AIAA Scitech 2021 Forum*, vol. ., pp. 1–28, 2021.

[3] A. Britcher, Victoria Gessow, *Exploration of Feature-Based Algorithm for Autonomous Ship-Deck Landing under Visually Degraded Conditions.* Thesis: University of Maryland, College Park, 2020.