4-30-2021

Automating Precision Drone Landing and Battery Exchange

Mia Scheider
miatschei@gmail.com

Automating Precision Drone Landing and Battery Exchange

By

Mia Todd Scheider

Approved by:

Mehmet Kurum (Major Professor)
Ali Gurbuz
Umar Iqbal
Jenny Q. Du (Graduate Coordinator)
Jason M. Keith (Dean, Bagley College of Engineering)

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Electrical and Computer Engineering
in the Department of Electrical and Computer Engineering

Mississippi State, Mississippi

April 2021

Name: Mia Todd Scheider

Date of Degree: April 30, 2021

Institution: Mississippi State University

Major Field: Electrical and Computer Engineering

Major Professor: Mehmet Kurum

Title of Study: Automating Precision Drone Landing and Battery Exchange

Pages of Study: 63

Candidate for Degree of Master of Science

As drones become more widespread throughout modern industry, the demand for drone automation increases. Drones are used for many applications, but their effectiveness relies heavily on their battery life. By designing, implementing, and evaluating an automatic drone landing and battery exchange system, drone missions can be more streamlined and efficient by eliminating the need for manual battery exchange. Previous projects within this topic rely on high-precision landing combined with a manipulator with low degrees of freedom for battery removal. This project offers a solution that allows less strict landing requirements to better fit drones of different sizes and shapes for a wide variety of applications. This autonomous drone landing and battery exchange system uses a robotic arm with 6 degrees of freedom for battery removal and on-board image processing to locate and land on a large, rotatable landing pad.

Key words: drone, UAV, precision landing, battery exchange, autonomous, robotic arm

DEDICATION

To my mom, who let me do the experiments in the back of my grade school science textbooks.

ACKNOWLEDGEMENTS

I would first like to thank the sponsors at ERDC for contributing the funds necessary to carry out this project. Additionally, I would like to thank my major professor Dr. Kurum and committee members Dr. Gurbuz and Dr. Umar for their patience and guidance in the completion of this thesis.

I would also like to thank my fellow NEST team members Austin Flynt, Nathan Goyette, and Spencer Barnes for their contributions to this project. Austin, in particular, was a huge help with testing the subsystems and Nathan provided the 3D modeling necessary for the custom end-effector.

I want to thank David Young and Mehedi Farhad for their assistance during drone testing and Dylan Boyd for helping with DroneKit based controls.

Finally, I want to thank my very good friend Bradley Humphreys who has provided me an incredible amount of emotional support throughout this process. Thank you for making sure I have groceries and for being patient with me during my hectic research hours.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS, ABBREVIATIONS, AND NOMENCLATURE

API	Application Programming Interface
GPS	Global Positioning System
MSU	Mississippi State University
PCB	Printed Circuit Board
RAM	Random-Access Memory
ROS	Robot Operating System
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
USB	Universal Serial Bus
θ	Angle of rotation about a given axis
x, y, z	Three-dimensional coordinate points
X, Y, Z	Three-dimensional axes
D	Displacement between two points
P	Point
R	Rotation matrix
t	Translation vector
A	Camera Matrix
T	Homogeneous Transformation

CHAPTER I

INTRODUCTION

1.1 Motivation

Also as unmanned aerial vehicles (UAVs) and remotely piloted aerial vehicles (RPAVs), drones are aircrafts characterized by their lack of an on-board human pilot. While general aviation development has contributed greatly to the technological advances of the modern world, drones bring with them smaller sizes and lower costs to fit a wider variety of applications [7]. The first drone developed is typically credited to Charles Kettering's "Kettering Bug" built in 1918 at the request of the US Navy [11]. Initially, drones were strictly intended for use by the military, but after the destruction caused by Hurricane Katrina in 2005, the Federal Aviation Administration (FAA) issued its first non-military certificates in 2006 for M7RQ military drones to be used over civilian skies for search and rescue purposes. With this development, the market was opened for further commercial and civilian drone technology advancements.

Presently, drones are widespread and used for a variety of applications. Within the military, drones prove useful by ensuring the safety of a potential on-board pilot when weaponized in hostile territory and by being able to easily transport sensor systems for surveillance and data collection. In the commercial market, drones are currently used for a wide variety of applications. Some commercial and civilian drone applications include film production, crop and land surveillance, animal population tracking, and power line inspection. Depending on the strength of the drone, they can also be used for transporting goods and delivering food or medical supplies [29].

Some drone missions, such as long-term data collection or surveillance, require long or repeated flights. The efficiency of these missions relies heavily on the battery life of the drone: how often the batteries need to be changed, and how long it takes to change them. Because a drone's battery life depends greatly on the drone's size and its battery capacity, it is more advantageous to optimize the battery exchange procedure.

Manual battery exchange can be difficult and time consuming, depending on the situation. For instance, the perimeter must be evaluated for safety before approaching the vehicle, the replacement battery must be located, the spent battery must be removed, and the replacement battery must be placed. After this, the mission can continue.

To streamline this procedure, the drone's flight, landing, and battery exchange should all operate autonomously to maximize performance. This way, no individual will be needed to intercept between missions. The incorporation of an automated drone battery exchange system would ultimately reduce the time cost of changing batteries by hand and increase overall efficiency.

1.2 Overview

To automate drone battery exchange, the method of exchange must first be decided. Because this system seeks to operate regardless of drone size and build, the OpenManipulator-P robotic arm will be used to access the drone's battery. This allows for the drone to land in any given angle orientation with respect to the arm so long as the drone lands within reach of the arm. To extend the reach of the arm, the arm will be attached to a 3-dimensional gantry so that the arm may access the landed drones and charged batteries more easily.

The drone batteries are installed into custom 3D printed battery casings that the robot arm's end effector is made to specifically lift, remove, and replace. The end effector also features a mount for a camera that is used to find an ArUco fiducial marker on the drone's 3D printed battery port. Using the image processing software OpenCV, the arm can be controlled using the Robot Operating System (ROS) to perform the battery exchange procedure.

To ensure that the drone lands in front of the arm, the drone's autonomous landing procedure is based around a rotatable landing pad that also features ArUco markers. Using a camera and Raspberry Pi mounted to the drone and connected to its flight controller, the position of the drone with respect to the landing pad can be found using OpenCV and then communicated to the drone using DroneKit-Python.

Once landed on the landing pad, the position of the drone on the pad can be determined given the ArUco markers seen by the drone's camera and communicated to a master computer. If the drone has landed too close to the pad's edge, the landing sequence can be repeated. However, if the drone has not landed too close to the edge, the landing pad can be rotated such that the drone is in front of the arm. Then, the arm can perform the necessary battery exchange before the next mission flight.

1.3 Objectives

The objective of this research is to further develop an autonomous drone landing and battery exchange system that is applicable to a wide variety of drone sizes and shapes. Discussed in

Chapter II, the design of the system is reliant on both previous work done in drone automation and on previous projects completed within the MSU IMPRESS lab. This defines not only the relevance of the study within its field, but also the existing requirements that the design must adhere to. Chapter III details the procedures, the hardware, and the software components used to implement each subsystem. The testing and evaluation of the system is conducted in Chapter IV, where the success rate of the given system is recorded and evaluated along with any major problems and solutions. Finally, Chapter V seeks to summarize the findings in designing, implementing, and evaluating this system and outline the future work to be done in the project's continuation.

CHAPTER II

BACKGROUND

2.1 Precedent in Drone Automation

As drones become more commonplace in everyday industry, the desire for autonomous control also increases. While developments have shown success in full drone autonomy, most drones are only partially autonomous or are controlled remotely on the ground. Fully automating drone flight is a highly complicated process and requires the consideration of environmental effects on the drone, potential obstacles, and path planning [8].

Several studies have been conducted since 2009 using machine learning tactics to completely automate drone flight through parameter tuning, adaptive control, real-time path planning, and navigation. While many significant advances have been made, these results are often limited to software simulation or indoor experimentation, which do not provide necessary outdoor variables such as wind. Additionally, results can be limited by small sample size, as drone flights are reliant on battery life and machine learning methods require incredibly large data sets [10]. Often, machine learning solutions for drone flight are also limited when used within a large target area due to poor performance in new environments. While there has been an algorithm proposed and tested to overcome this issue [22], the autonomous navigation of a drone within a small, controlled space for landing can surely be achieved with a much simpler and accessible solution.

In a study detailing the development of an autonomous drone for agricultural applications, the ground control station Mission Planner along with the Cube (PixHawk 2) flight controller is used to plan and execute the drone's flight using pre-determined waypoints and open a payload bay to release goods to farmers [18]. Mission Planner uses the MAVLink protocol to communicate with drone flight controllers [2], however, waypoint based pre-planned missions do not allow for any mid-flight correction.

Another project proposed an autonomous obstacle detection and avoidance system using LiDAR and the DroneKit-Python library with the Cube flight controller. DroneKit-Python is a Python library that generates and sends MAVLink messages to the drone's flight controller from written code. In this case, an Intel NUC processor is used as an on-board companion computer that processes the information from the LiDAR. The companion computer then uses this information to calculate an optimal path for the drone to follow. While early in the testing process, the results of this study show some success in controlling a drone with respect to outside influence through Python code [21].

Autonomous drone flight eliminates the need for a human pilot to control the drone, but still requires a human to manually exchange drone batteries for continuous flight. To automate this procedure, the batteries can either be charged directly on the drone or exchanged with fresh batteries and charged off the drone.

One proposal seeks to fully automate drone activity by incorporating wireless battery charging into a solar powered drone charging station. This project uses a resonant inductive coupling system with a series-series topology to allow more room for error in landing. From simulation results, the battery power to input power efficiency was shown on average to be 88.7% with an 83.4W output [30]. However, because wireless power transfer tends to lack in efficiency and can take long periods of time to complete charging, another possible drone charging method was proposed using a custom socket-plug system. After drone landing, a mechanical system is used to engage a socket with a plug connected to the drone's battery for charging. The socket-plug connection is designed to maintain low ohmic resistance even after several charging cycles [6]. Both proposed methods rely on the drone remaining in place during the charging procedure, which introduces more idle time in drone operation. This issue could be remedied by flying several different drones in rotation, but full battery exchange may be necessary to maximize activity for all drones in use.

One project proposes a 3-part terrestrial power relay platform to fully automate battery exchange. This platform includes three modules for landing, battery charging, and battery exchange. Battery exchange uses a manipulator composed of a rigid beam attached to two linear actuators that allow movement both up/down and forward/back. The beam attachment includes an electromagnet for removing and placing batteries. Once a drone has landed on the landing module, the battery exchange manipulator removes the spent battery and places it in the battery charging module. A charged battery can then be picked up by the manipulator and deposited into the drone. This battery exchange procedure showed an 80% success rate after experimentation [13].

Another project, the Endless Flyer, uses a battery replacement system based on linear motion. As a new battery is pushed into place on the drone, the old battery slides out of the drone. As with the previous project, the Endless Flyer requires that the drone land in a particular position, however, the system uses L-shaped arms connected to servo motors to move and position the landed drone into the desired position and orientation. The drone's landing success rate was reported to be 90% with a 100% success rate of battery exchange upon successful landing [16]. Both proposals detail potential methods of physical battery removal from a drone, but both are also designed to only operate if the drone has landed in a particular position which requires the implementation of precise drone landing.

Precision drone landing often involves some form of image processing for navigation and correction. One proposal for autonomous precision drone landing uses a PixHawk flight controller connected to a Raspberry Pi companion computer running MAVROS. MAVROS is a ROS package that is used to send MAVLink commands to the drone's flight controller, similar to DroneKit-Python but uses ROS for control instead. To track the landing target image, the computer vision software OpenCV is used with the FindObject2D library. Using this setup, the drone was able to land close to the target with an error of about 1.5 m due to difficulty with wind resistance and low image resolution [19].

Another method for autonomous precise landing uses a hybrid fiducial marker where a small LentiMark marker is placed on a larger ARToolKit marker. The ARToolKit marker serves as a guide to the landing station from farther away, while the smaller LentiMark is used for higher accuracy landing control when the drone gets close enough to recognize it. Overall, this system produces a positioning error of around 50 mm or less [32].

Given the previously discussed studies on drone automation, battery exchange, and landing, autonomous drone control is a complex task to accomplish. Automating drone flight control requires a method of programmable control that is compatible with the drone's flight controller. Exchanging batteries autonomously relies on a mechanical system for battery manipulation as well as knowledge of the drone's position relative to the system. Finally, autonomous landing requires using computer vision software on a running camera feed to determine the position of the landing target relative to the drone.

2.2 Overview of Previous Projects

This project operates within the Mississippi State University (MSU) Information Processing and Sensing (IMPRESS) lab and is therefore influenced by previous related projects. The IMPRESS lab has two other projects that relate directly to drone battery exchange: the droNE_STation (NEST) and the Automatic Battery Exchanger (AuBex).

2.2.1 NEST: droNE_STation

The NEST is an ongoing project that seeks to serve as a ground station for fully autonomous drone operation housed by a trailer. A picture of NEST is shown in Figure 2.1. The NEST trailer includes two ArUco marker landing pads for two simultaneous drone missions. One pad can extend horizontally out of the back of the trailer through automatic doors while the other pad extends vertically through an automatic roof. The NEST's master computer connects to a server that is used to monitor the NEST's status and administer commands. The commands instructed are used to control the mechanical and software processes within the NEST, such as opening and closing the doors, instructing drones to take off, and storing any data gathered from drone missions. This study specifically defines the methods used for a proposed battery exchange system for the NEST.

The NEST operates on the principle that a wide variety of drones can be dispatched from its trailer, which means that the battery exchange and landing procedures must keep this in mind. While previous methods in drone automation focus on designing a system for one particular kind

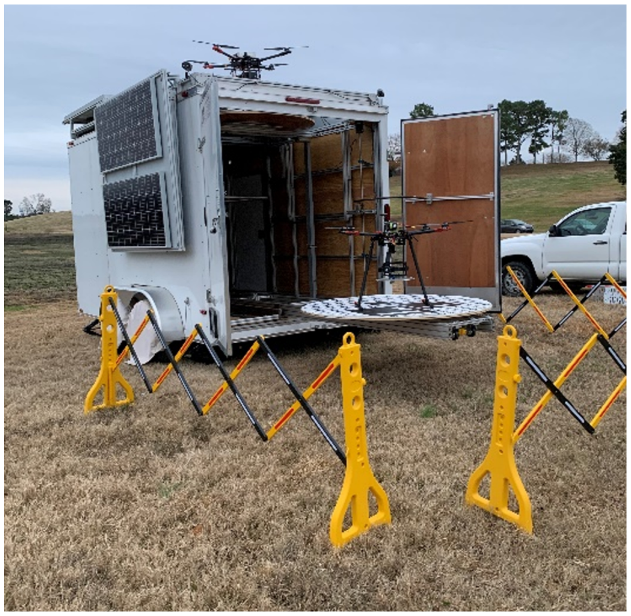


Figure 2.1

A picture of NEST deployed in the field.

of drone, the NEST's battery exchange system must be applicable to many shapes and sizes of drones. To accommodate this, the landing pads used are much larger than typical precision landing targets to allow larger drones. The pads' larger size, however, also requires an altered approach to using image processing. The landing pad design uses a spiral step-down approach where the landing target ArUco markers start out very large but then decrease in a spiral pattern. This allows the pad to be visible both from a distance and up close.

2.2.2 AuBEx: Automatic Battery Exchanger

AuBEx was an MSU Electrical and Computer Engineering (ECE) senior design team project that prototyped a similar battery exchange procedure with a manipulator and ArUco markers. The manipulator used was a 2-dimensional gantry system with a camera and custom 3D printed end-effector. The 3D printed end-effector and battery casings from AuBEx are used for battery exchange in this project, however, AuBEx operates on an already placed drone, and does not need to account for precision landing. This project aims to amplify the success of AuBEx by applying its principles on a larger scale.

CHAPTER III

METHODS

3.1 Hardware Overview

The hardware setup for overall drone flight automation begins with the drone itself. While this project has no designated drone for testing, the drone typically chosen is a small frame plastic drone that is lightweight and easy to transport. All drones available for testing are fitted with the Hex Cube Flight controller (formerly known as the Pixhawk2) [27] and FrSky X8R telemetry module. The transceiver used by the drone and ground control is the Holybro 433 MHz antenna and the Here2 GPS module is used for positioning. Much of the hardware mounted to the drones are a standard across the lab and are thus used due to availability. However, the hardware that is more specific to the project had to be chosen more deliberately.

The first project-specific area to address is the drone's ability to "see" the landing pad. To do this, a camera had to be chosen that is small, high performance, and easily interfaceable. The ELP USB 2.1mm lens camera is a reasonably affordable option that delivers this project's typical resolution of 480p at up to 120 fps for accurate pose estimation. This camera is also very small at 122x122 mm with holes in the PCB for easy mounting. Additionally, the USB interface provides versatile communication to the applicable methods of image processing [14].

To perform the necessary image processing on the camera's feed, an on-board computer had to be chosen that has a small form factor, network capabilities, and enough processing power to adequately communicate with the drone's ground control. The Raspberry Pi 4 is an affordable yet powerful option for image processing and ground control communication that remains small in build. The Raspberry Pi 2 was used initially to perform this task due to its availability in the lab, but it's communications with the drone software proved to be too slow to use. The Raspberry Pi 4 performed much quicker and reduced the communication latency significantly. Furthermore, the Pi 4 has built in Wi-Fi capabilities and does not require an external adapter unlike the Pi 2. The Pi 4 also has 4 GB of RAM with a quad-core processor, which allows adequate multitasking ability for image processing and drone control. The Pi is also compact enough to mount onto the smaller size drones without crowding or adding excessive weight.

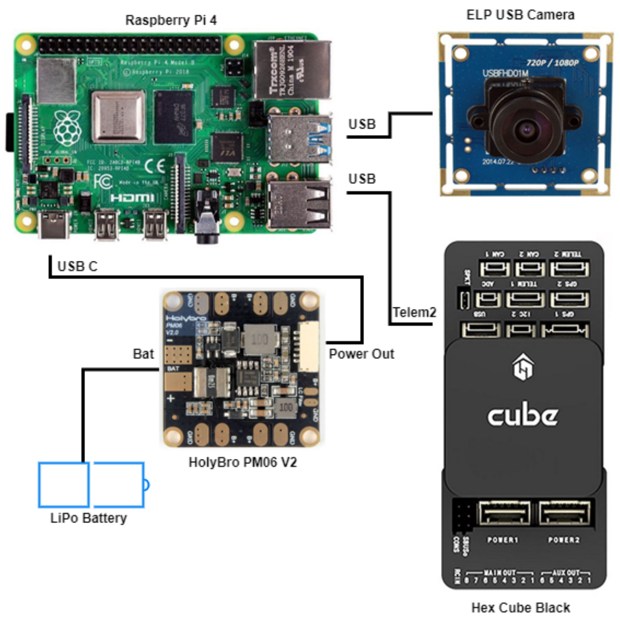


Figure 3.1

Schematic of Drone's On-Board Image Processing System

The Raspberry Pi 4, while small and powerful, also has high power requirements. The Pi 4's recommended power supply has an output of 5V at 3A [4], which could cut flight time drastically if the drone's power source was split between both the flight controller and the Pi 4. To remedy this, an additional battery is used with a Holybro PM06 V2 power module to step the battery voltage down to 5V at a maximum of 3A. This module can be used with any batter so long as the voltage falls between 7V and 42V [3]. The schematic of the on-board system is shown in Figure 3.1.

To use on the drone, the on-board components must have a mounting method. This was done by cutting a plastic plate to mount the individual parts onto that could then be attached to a drone using threaded PCB standoffs. Figure 3.2 shows the hardware components mounted to the plastic plate from both the bottom and top, while Figure 3.3 shows an example of what the board looks like mounted onto an actual drone.

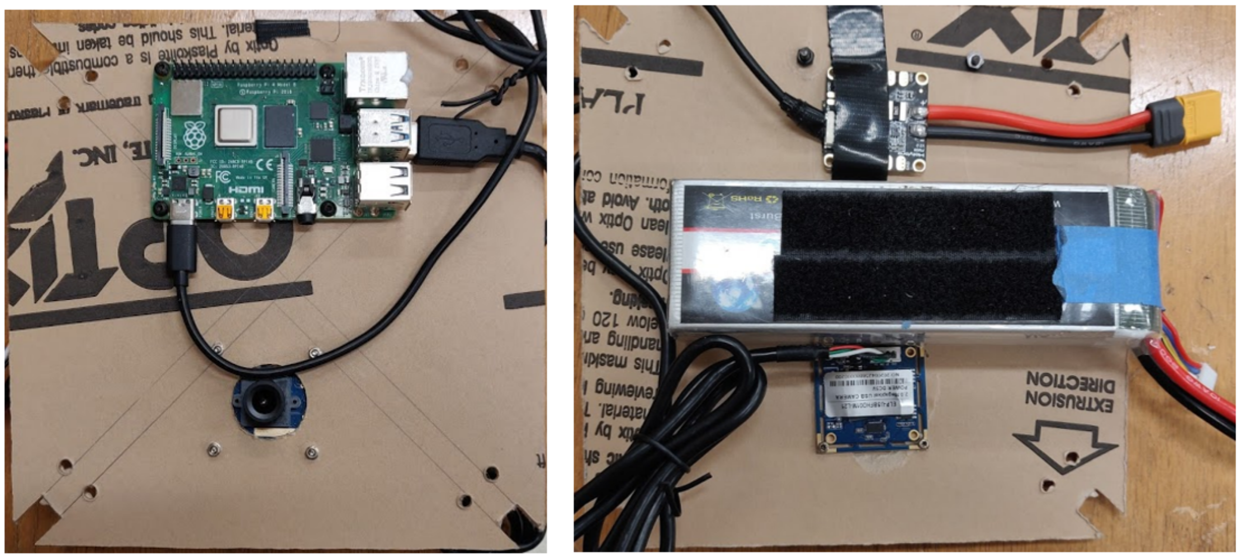


Figure 3.2

Bottom and Top View of Mounted On-Board Image Processing System

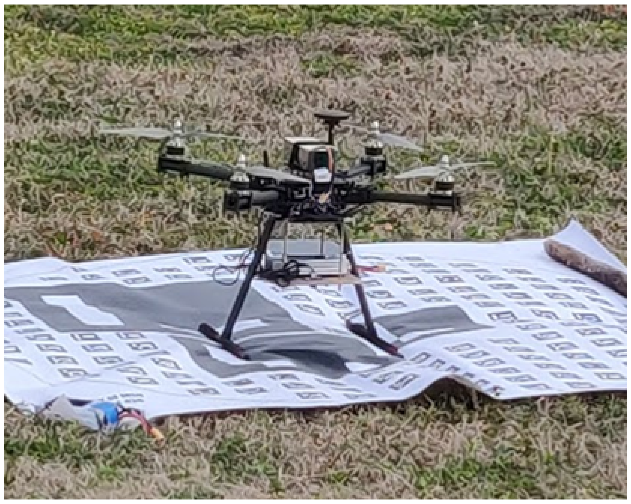


Figure 3.3

Example of Drone in Field with Attached Image Processing System

To access any data that the Raspberry Pi needs to broadcast, as well as execute the individual procedures, a remote computer is needed. For this, the requirements are only that the computer have Wi-Fi capabilities and the ability to execute Python scripts. The battery exchange procedure also requires a computer for controlling the robotic arm, however, the recommended software must be run on a Linux machine. Typically, lab executed battery exchange tests are run on a dedicated lab laptop with an Ubuntu 16.04 dual boot while drone flight executions are launched from a lightweight Windows 10 laptop that is easily transportable. In future work, both sequences will be executed from the trailer's master computer, which also runs Ubuntu 16.04.

The autonomous battery exchange procedure requires an extension that can easily access a drone at any given orientation in front of it. This extension must also be able to reach at least half (0.762 m) of the landing pad since the pad itself can rotate. The OpenManipulator-P robotic arm was purchased to satisfy the orientation requirements due to its 6 axes of rotation and extensive documentation and provided source code. However, the OpenManipulator-P can only extend to approximately 0.525 m, which does not reach the center of the landing pad. [31] To extend the arm's reach, the OpenManipulator-P will be mounted onto a custom-built gantry system within the trailer. The arm's end effector attachment is a custom 3D printed part made to specifically lift the 3D printed battery casings. This attachment also includes a space to mount a second ELP USB camera for seeking out the drone batteries.

3.2 Software Overview

To interface between the various pieces of hardware, the appropriate software must be written and incorporated into the system. For controlling the movements of the drone autonomously, the DroneKit-Python library is used. For image processing and pose estimation of the landing pad, the OpenCV Python library is used. Both Dronekit and OpenCV are used with Python 2 specifically for stability and compatibility. The OpenCV and DroneKit scripts are run using the Raspberry Pi 4 on the drone itself, while the OpenManipulator-P is controlled by the Robot Operating System (ROS) on the master computer using Ubuntu 16.04. To communicate between the two, a UDP server connection is established also using Python 2. This allows the Raspberry Pi to send post-landing position data to the master computer so that the battery exchange procedure can determine the drone's location. The battery exchange system also uses OpenCV for pose estimation of the battery ports, but uses the OpenCV C++ ROS package instead of Python.

3.3 ArUco Markers and Pose Estimation

For accurate positioning in both landing and battery exchange, the cameras used must have some point of reference for pose estimation. In this case, the original 5x5 ArUco fiducial marker library is used because it is freely available, highly documented, and provides distinct integer values per marker that are needed for positioning and battery retrieval purposes.

The ArUco markers themselves are comprised of an outer black border and an inner binary matrix represented by black and white squares [17]. Shown in Figure 3.4 is the original library ArUco marker representing the integer 1 with its corresponding binary matrix. This is a 5x5 marker, meaning the inner matrix is 5x5 bits.

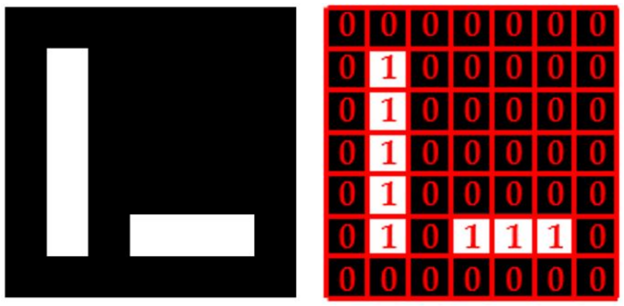


Figure 3.4

ArUco Marker ID 1 with its Binary Matrix Representation

To implement pose estimation, however, a computer vision software must be used to process the ArUco images the camera captures. The recommended companion software for ArUco, OpenCV, is a computer vision library available for both Python and C++ [28][9]. Besides ArUco based pose estimation, OpenCV also offers all necessary functionality for camera calibration. Because accurate pose estimation relies on a distortion-free image, camera calibration must be performed to calculate the necessary values for undistorting the camera image [24]. Figure 3.5 shows an example of an image with positive radial distortion before and after undistortion where the corrected image shows straight lines instead of the original's curved lines.

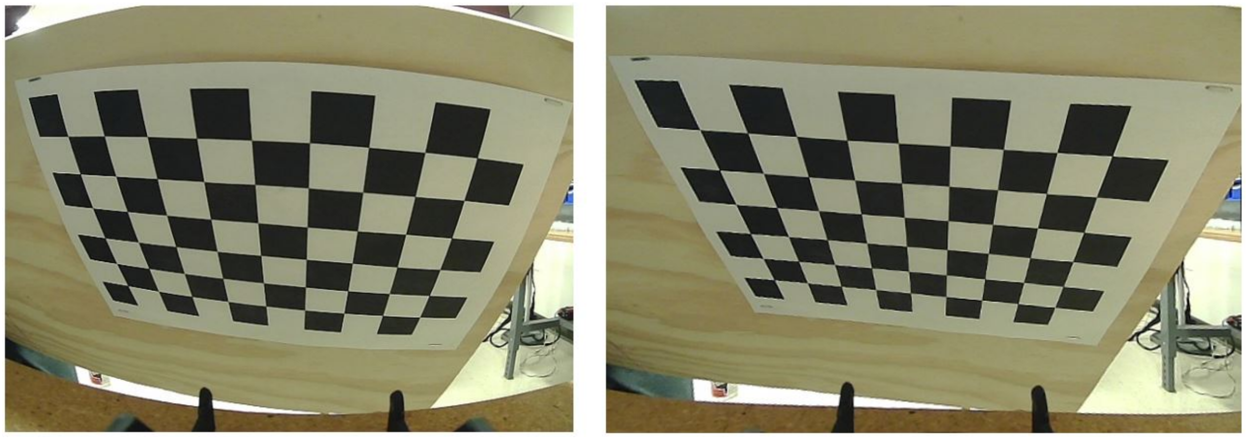


Figure 3.5

Chessboard Calibration Image before and after Undistortion

OpenCV's calibration process seeks to correct both radial distortion, where straight lines are shown as curved lines, and tangential distortion, where objects appear closer than expected. This process is performed by photographing a chessboard reference image several times from several angles and processing the images through a Python script using the OpenCV calibration functions. 30-40 images are taken of the reference board before processing to ensure the most accurate results. The software displays a pattern on each reference image where the user can either accept it for processing or discard it. Figure 3.6 shows examples of both acceptable and unacceptable calibration patterns. On the left, the acceptable test image shows each grouping of dots placed correctly on the inner corners on the board. The unacceptable test image on the right shows the dots placed inaccurately and unevenly. Once all images have been evaluated, the camera matrix and distortion coefficients are calculated for pose estimation [23]. The Python scripts used for collecting reference snapshots and calibrating the camera are open-source and found on GitHub [15].

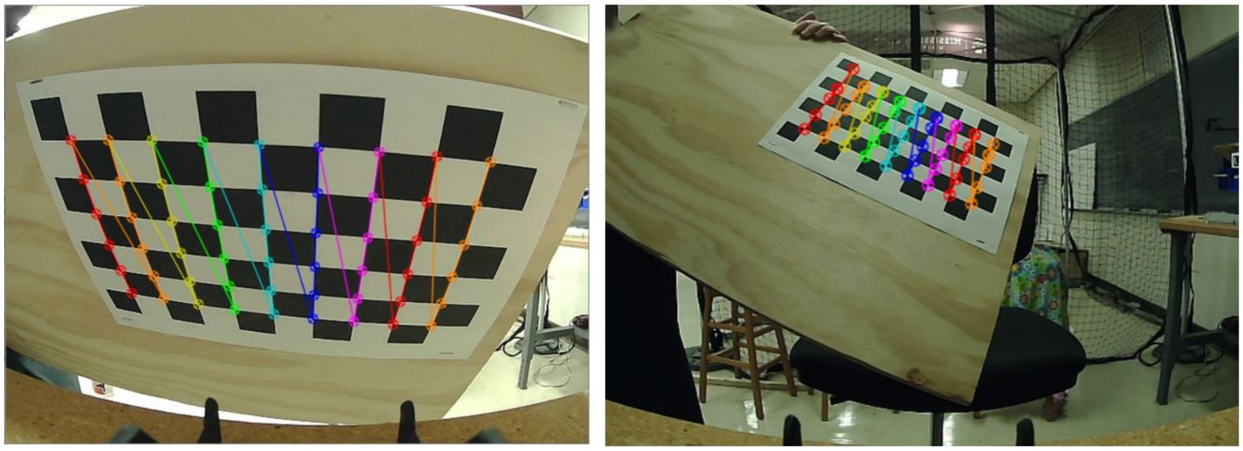


Figure 3.6

Example of both Acceptable and Unacceptable Test Images

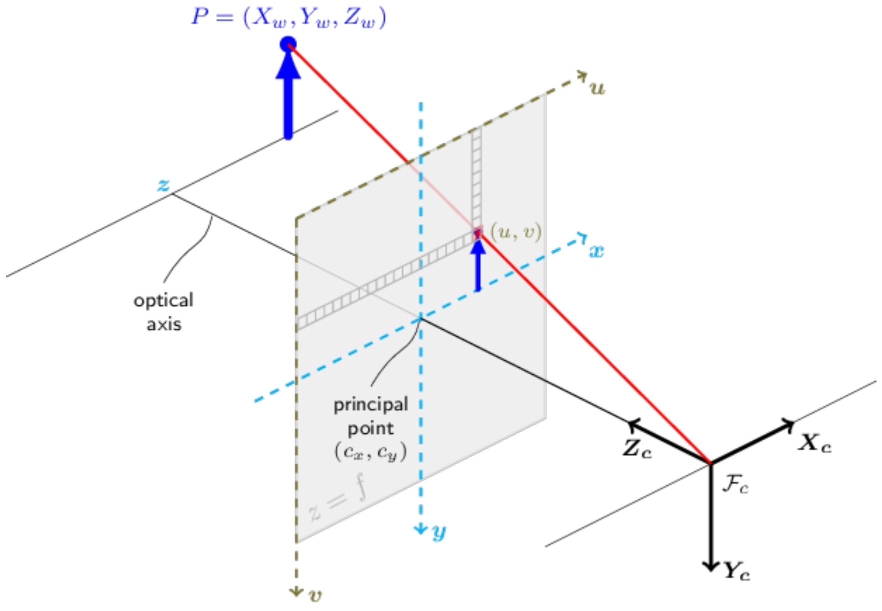


Figure 3.7

The Pinhole Camera Model [24]

Pose estimation, in this case, seeks to track the 3D displacement of an ArUco marker with respect to the camera module. Shown in Figure 3.7, this 3D reconstruction is implemented by OpenCV using the pinhole camera model. This model works by projecting a 3D point P onto the 2D pixel-based camera frame. This model is represented fundamentally by the following equation:

$$s\,p = A[R|t]P_w \tag{3.1}$$

Here, P_w represents the 3D point with respect to the world axis and p represents the projected 2D pixel point of P_w onto the camera image. A is the camera matrix of intrinsic values, R and t refer to the rotation and translation of the world frame to the camera frame, and s represents an arbitrary scaling factor. For drone navigation and battery exchange, the main areas of interest are the rotation matrix and translation vector which are represented as such:

$$[R|t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \tag{3.2}$$

where

$$R = R_z R_x R_y \tag{3.3}$$

which is further defined by the following transformations:

$$R_z = \begin{bmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

$$R_x = \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) \\ 0 & 1 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) \end{bmatrix} \tag{3.5}$$

$$R_y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) \\ 0 & \sin(\theta_x) & \cos(\theta_x) \end{bmatrix} \tag{3.6}$$

The rotation matrix is used to calculate the angle orientation of the marker with respect to the camera while the translation vector reports the marker's distance displacement from the camera [24]. The angles $\theta_x\theta_y\theta_z$ are calculated as follows:

$$\theta_x = \text{atan2}(r_{32}, r_{33}) \tag{3.7}$$

$$\theta_y = \text{atan2}(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}) \tag{3.8}$$

$$\theta_z = \text{atan2}(-r_{21}, r_{11}) \tag{3.9}$$

However, if r_{21} and r_{11} are both close to or equal to 0, θ_x must then be calculated using:

$$\theta_x = \text{atan2}(-r_{23}, r_{22}) \tag{3.10}$$

where

$$\theta_z = 0 \tag{3.11}$$

These calculations yield the roll, pitch, and yaw angles of the marker with respect to the camera [12].

3.4 Landing Procedure

Success of the drone landing procedure is predominantly dependent on two major variables: controlling the drone's flight and processing the landing pad's position. For drone flight control, DroneKit-Python will be used to connect to and communicate with the drone's flight controller while OpenCV and ArUco markers will be used for tracking the landing pad position.

Of the drones available in-lab, all are fitted with the Cube flight controller using the PX4 firmware which communicates using the MAVLink protocol [20]. To control the drone autonomously, the software package used must also interface with the Cube using MAVLink. DroneKit-Python is an open-source Python API that uses simple function calls to communicate with vehicles using MAVLink. DroneKit is typically executed by an on-board companion computer over a low-latency link or by a ground station application using higher-latency telemetry control [5]. Because the Raspberry Pi 4 is already in use for ArUco detection aboard the drone, connecting it directly to the Cube flight controller for drone communication will further streamline the process. All DroneKit code is written in Python 2 for best stability and compatibility and executed from the Raspberry Pi.

The drone's landing pad is designed in such a way as to be easily visible both from up close and from far away. Shown in Figure 3.8, the pad displays the ArUco marker IDs 1 through 156 in decreasing size. Marker IDs 1, 2, and 3 represent the largest, second largest, and third largest respectively, ID 4 represents the center of the pad, and marker IDs 5 through 156 make up the grid of small markers. The pad itself is 1.524 meters across with the largest marker measuring 50.75x50.75 cm and the smallest measuring 5x5 cm. By having markers that decrease in size in this way, the drone can see the largest marker from far away for initial placement and from up close

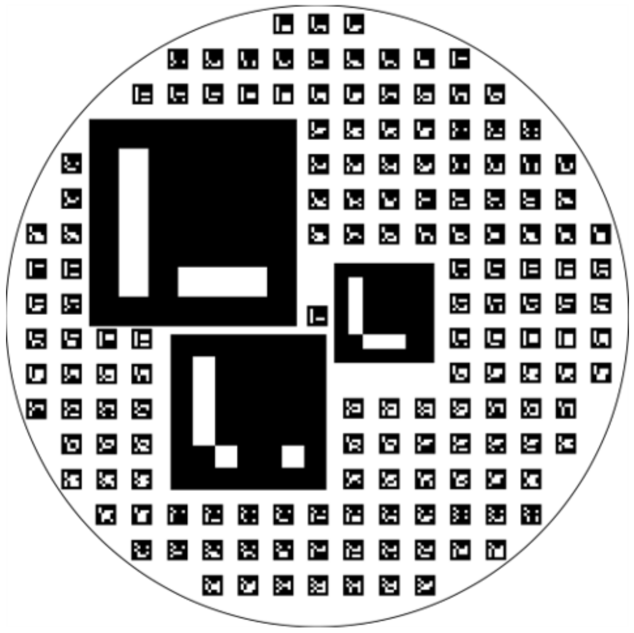


Figure 3.8

The Pad Design Used for Autonomous Drone Landing

for more precise landing. As the drone descends, the ArUco detection script switches to look for the most appropriately sized marker.

Once the landing procedure has begun, the drone cannot descend until its error with respect to the center of the current marker is low enough. This error is calculated using the example shown in Figure 3.9 where Δx, Δy, and Δz are the cm displacements of the drone with respect to the marker as reported by OpenCV. The angle θ_e is the calculated descension error angle using the following:

$$\theta_e = \text{atan2}(\sqrt{\Delta x + \Delta y}, \Delta z) \tag{3.12}$$

This allows the error to be calculated proportionally to the current height of the drone. By doing this, the drone is allowed more room for error the higher its altitude. This works because the battery exchange procedure does not rely on any particular landing orientation and only requires

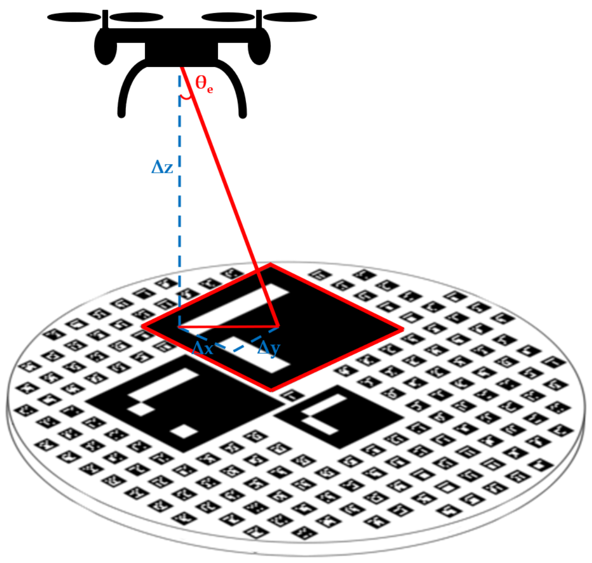


Figure 3.9

Drone Descension Error Calculation Model

that the drone lands safely on the pad. If the error is not low enough to descend, however, the drone must be instructed to correct its position.

Drone flight instructions are typically given in GPS coordinates, which requires that cm correctional distance be converted to latitude and longitude degrees. Given that Δx and Δy refer to the cm offset of the drone with respect to the marker and θ_{yaw} refers to the yaw angle of the drone with respect to directional North, the cm displacement of the marker with respect to the drone expressed in the North and East directions are calculated as such:

$$D_N = \Delta x \cos(\theta_{yaw}) - \Delta y \sin(\theta_{yaw}) \tag{3.13}$$

$$D_E = \Delta x \sin(\theta_{yaw}) + \Delta y \cos(\theta_{yaw}) \tag{3.14}$$

Now, given that D_{lat} and D_{lon} represent the displacement of the marker with respect to the drone expressed in latitude and longitude radians and R_{Earth} represents the approximate radius of the Earth in meters, the following can be calculated:

$$D_{lat} = \frac{D_N(0.01)}{R_{Earth}} \tag{3.15}$$

$$D_{lon} = \frac{D_E(0.01)}{R_{Earth}\cos(lat_0\frac{\pi}{180})} \tag{3.16}$$

$$lat_1 = lat_0 + D_{lat}(\frac{180}{\pi}) \tag{3.17}$$

$$lon_1 = lon_0 + D_{lon}(\frac{180}{\pi}) \tag{3.18}$$

Where lat_0 and lon_0 represent the drone's current latitude and longitude and lat_1 and lon_1 represent the marker's calculated latitude and longitude. The marker's calculated latitude and longitude are then sent to the drone as an instruction before evaluating for error again [15]. The full procedure in detail is shown Figure 3.10.

3.5 Post-Landing Procedure

After the drone's mission has run and the drone has successfully landed on the pad, the master computer must be informed where the drone is located on the pad and what direction it is facing. To do this, another Python script is executed on the Raspberry Pi to look at the markers below the drone and report the calculated cm position of the drone on the pad. A diagram detailing the basic geometry of this task is shown in Figure 3.11.

From this camera feed, the script selects any arbitrary marker that the camera can see and calculates the drone's position based off of the drone's angle and cm displacement as reported in

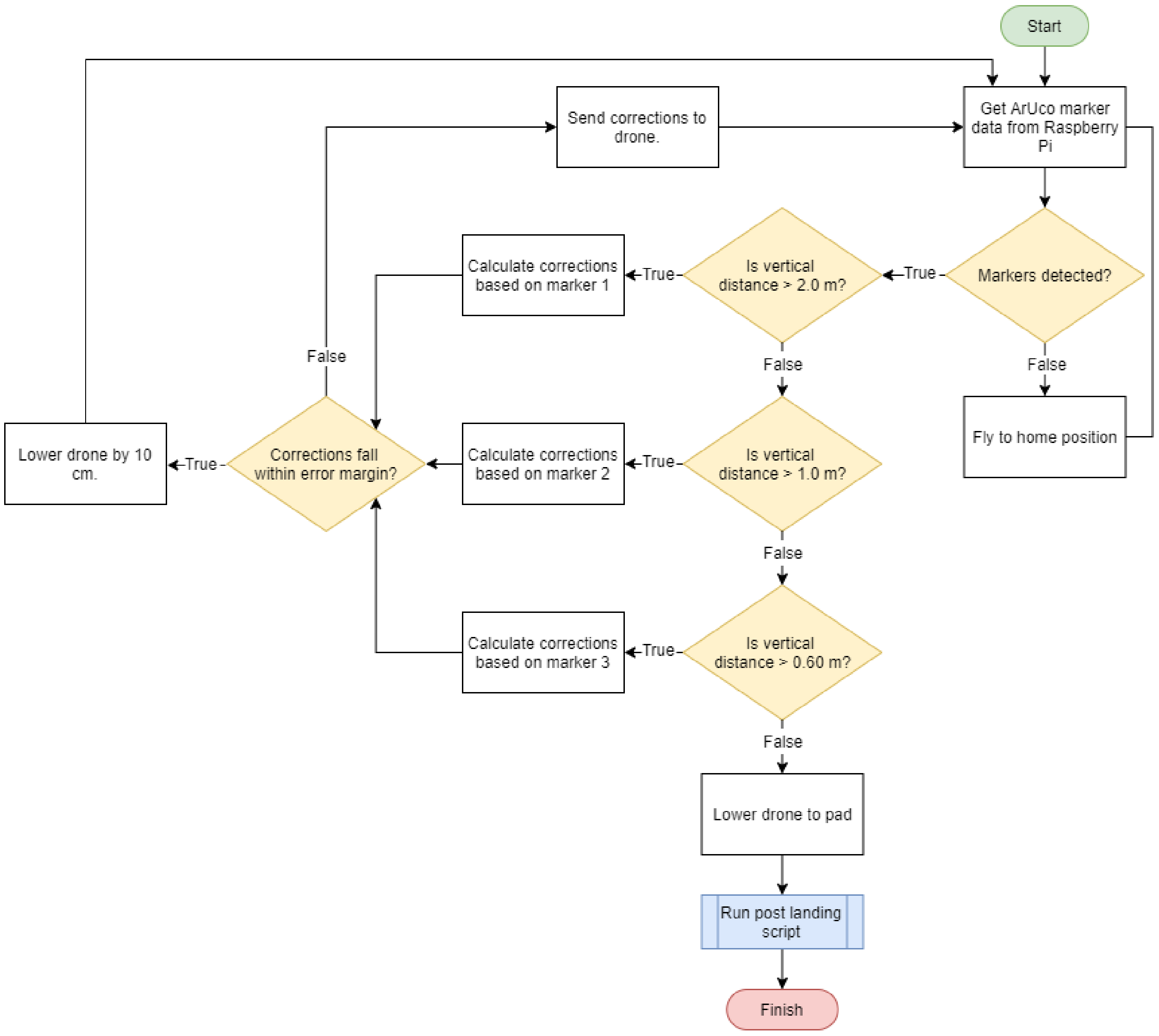

Start
Get ArUco marker data from Raspberry Pi
Send corrections to drone.
Markers detected?
True
False
Fly to home position
Is vertical distance > 2.0 m?
True
False
Calculate corrections based on marker 1
Is vertical distance > 1.0 m?
True
False
Calculate corrections based on marker 2
Is vertical distance > 0.60 m?
True
False
Calculate corrections based on marker 3
Corrections fall within error margin?
True
False
Lower drone by 10 cm.
Lower drone to pad
Run post landing script
Finish

Figure 3.10

Flowchart for Autonomous Drone Landing Procedure

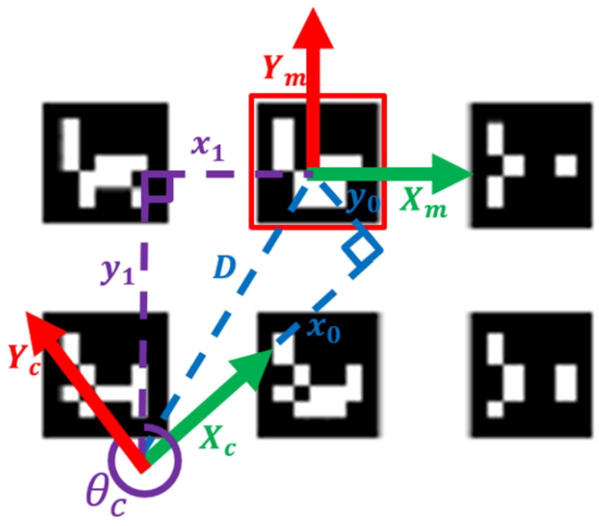


Figure 3.11

Diagram of Post-Landing Positioning Geometry

the ArUco marker's rotation matrix and translation vector. θ_c represents the z-axis yaw angle of the drone with respect to the marker. This angle is measured from the marker's positive y axis. D is the diagonal distance between the points P_c and P_m, the cm positions on the landing pad of the camera and marker respectively. x_0 and y_0 are the cm displacements given by the marker's translation vector and describe the marker's position with respect to the camera's axis while x_1 and y_1 are the cm displacements that describe the marker's position with respect to the marker's axis. x_1 and y_1 can be calculated using the following transform:

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} \cos\theta_c & -\sin\theta_c \\ \sin\theta_c & \cos\theta_c \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{3.19}$$

which expands to:

$$x_1 = x_0\cos\theta_c - y_0\sin\theta_c \tag{3.20}$$

$$y_1 = x_0\sin\theta_c + y_0\sin\theta_c \tag{3.21}$$

Then, the point P_c can be calculated using the following:

$$p_{cx} = p_{mx} - x_1 \tag{3.22}$$

$$p_{cy} = p_{my} - y_1 \tag{3.23}$$

where

$$P_c = \begin{bmatrix} p_{cx} \\ p_{cy} \end{bmatrix} \tag{3.24}$$

$$P_m = \begin{bmatrix} p_{mx} \\ p_{my} \end{bmatrix} \tag{3.25}$$

The cm positions of each marker on the pad is defined in a Python dictionary where they can be easily accessed for these calculations. After calculation, the drone's position on the pad is displayed using the Matplotlib Python library to plot the drone's radius and heading orientation. An example of this is shown in Figure 3.12.

The visualization script shows the drone's radius as a green circle if the drone has landed safely, or as a red circle if the drone has landed too close to the edge. This is calculated by checking if the drone's radius falls completely within a user-defined "safe-zone." If the drone has landed safely, the landing pad will pull back into the trailer for battery exchange. However, if the drone has not landed safely, the landing procedure must be repeated. Additionally, the drone's angle orientation is shown by the blue arrow within the circle.

The Raspberry Pi 4 broadcasts the drone's position over a UDP server connection to the master computer. The battery exchange procedure can then operate using the drone's position as a reference. The full post-landing procedure flowchart is shown in 3.13.

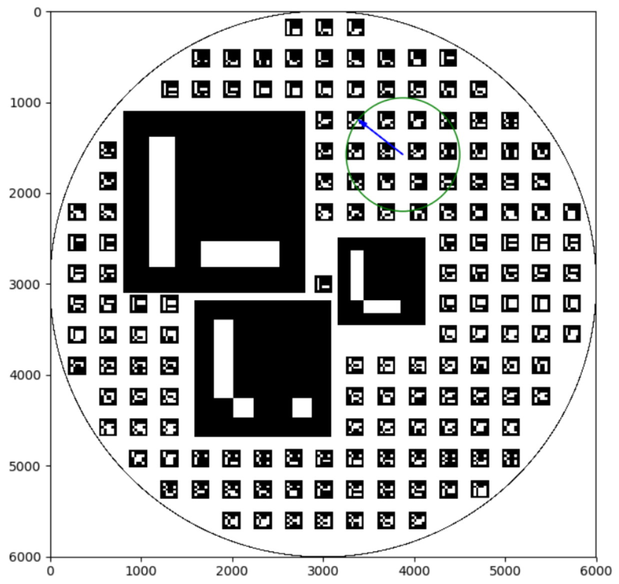


Figure 3.12

Post-Landing Positioning Visualization

3.6 Battery Exchange Procedure

Once the drone's position on the pad has been recorded and broadcast to the master computer, battery exchange must take place. To remove and replace the batteries, the OpenManipulator-P robotic arm is used along with the Robot Operating System (ROS) for control. Shown in Figure 3.14, the drone's battery will be deposited into a custom 3D printed port that displays an ArUco marker for alignment. The battery itself is held in a 3D printed casing with two holes in the bottom to serve as contact points for the battery. The robot arm also has a custom end effector with a camera mount that is built to retrieve and deposit the battery casing. The battery casing and end effector are shown in Figure 3.15 where the battery casing is shown on the left and the end effector on the right. Additionally, 3.16 shows how the two parts interact together in battery exchange.

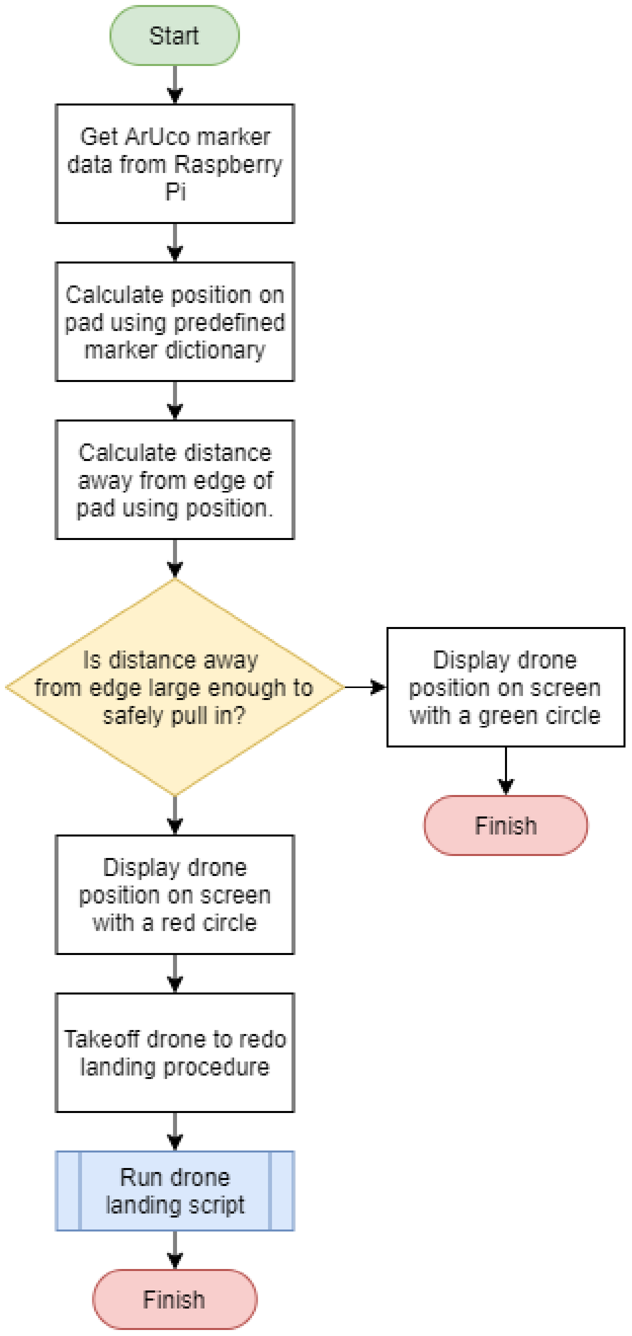

Start
Get ArUco marker data from Raspberry Pi
Calculate position on pad using predefined marker dictionary
Calculate distance away from edge of pad using position.
Is distance away from edge large enough to safely pull in?
Display drone position on screen with a green circle
Finish
Display drone position on screen with a red circle
Takeoff drone to redo landing procedure
Run drone landing script
Finish

Figure 3.13

Flowchart of the Drone Post-Landing Procedure

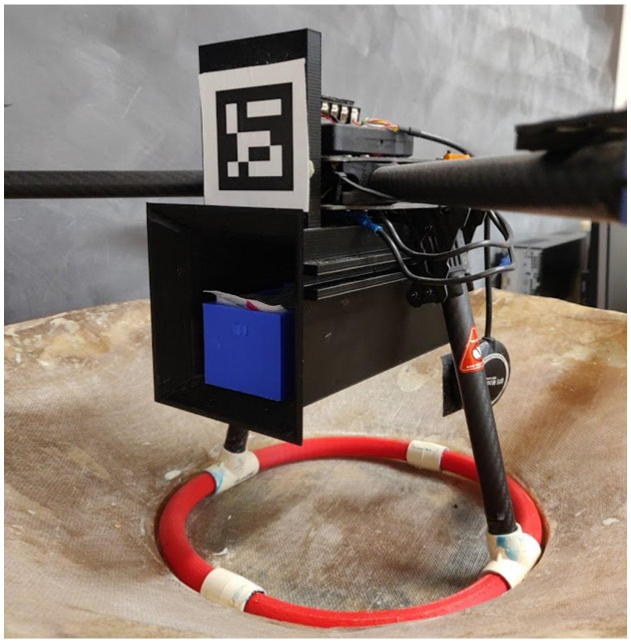


Figure 3.14

Custom 3D Printed Battery Port with ArUco Marker for Battery Exchange

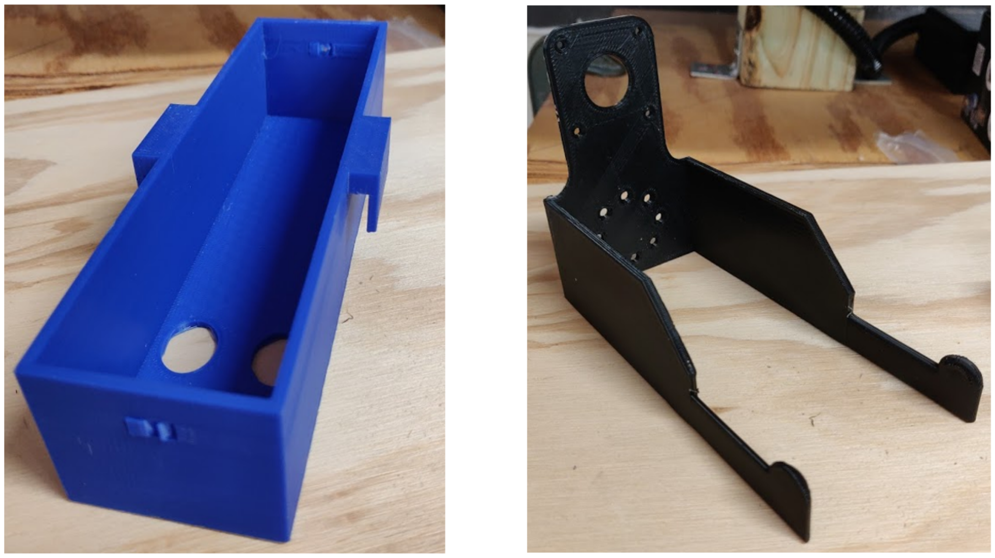


Figure 3.15

Custom 3D Printed Battery Casing and Robot Arm End Effector

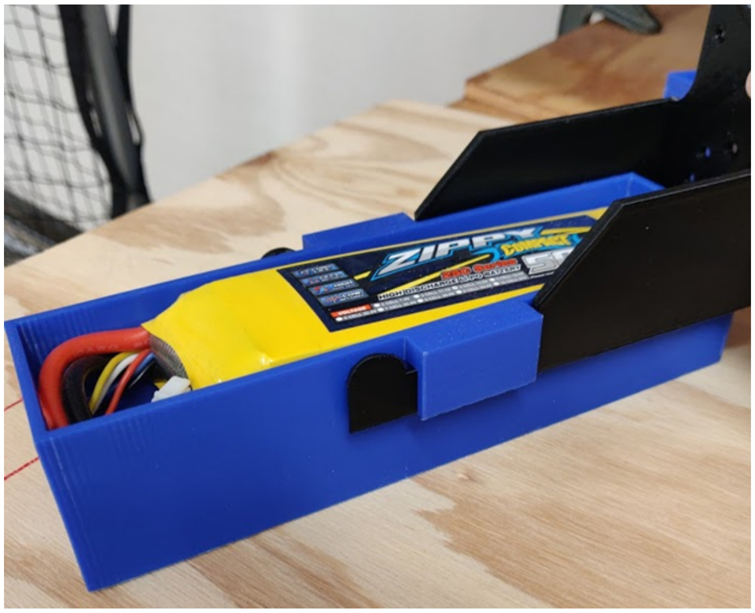
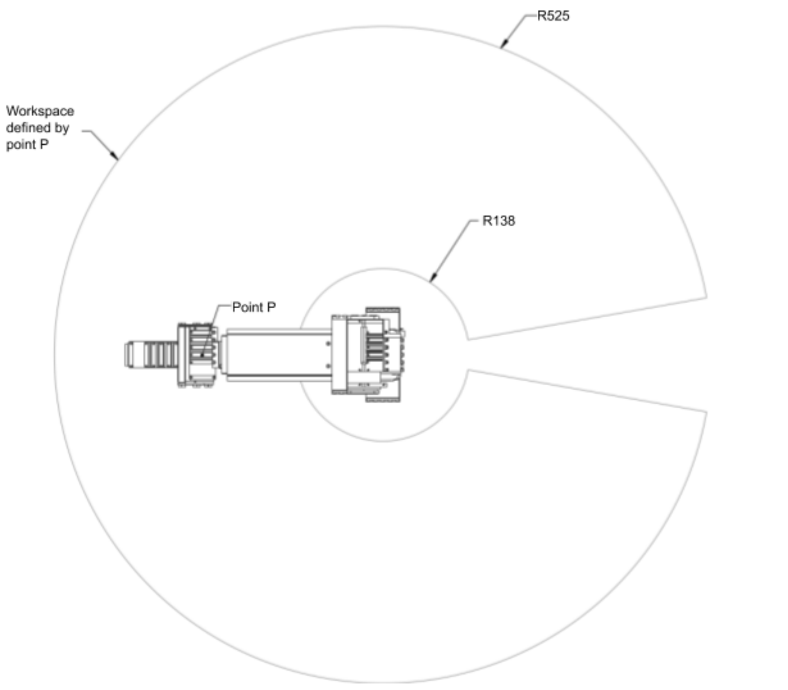


Figure 3.16

Demonstration of How Battery Casing is Carried by Custom End Effector

Much like the previous procedures, battery exchange also relies heavily on ArUco detection and pose estimation. Using the data obtained from the post-landing procedure, the position of the drone relative to the pad is known, but the position of the drone relative to the arm must be calculated and optimized. Shown in Figure 3.17 is the OpenManipulator-P's volumetric workspace from the top and from the side [31]. The workspace represents each point that the arm can reach [25]. From this, The maximum length that the arm can outstretch is 0.525 m. However, for battery exchange purposes, the arm must be able to reach at least the radius of the landing pad. The pad itself measures 1.524 m with a radius of 0.762 m. Because this lies outside of the arm's workspace, the arm will be attached to a custom gantry system to supplement the needed reach.

The full system setup is shown in Figure 3.18 where the arm is a distance D from the center of the landing pad. The drone's position P can be expressed by the vectors P_{pad} and P_{arm} in reference



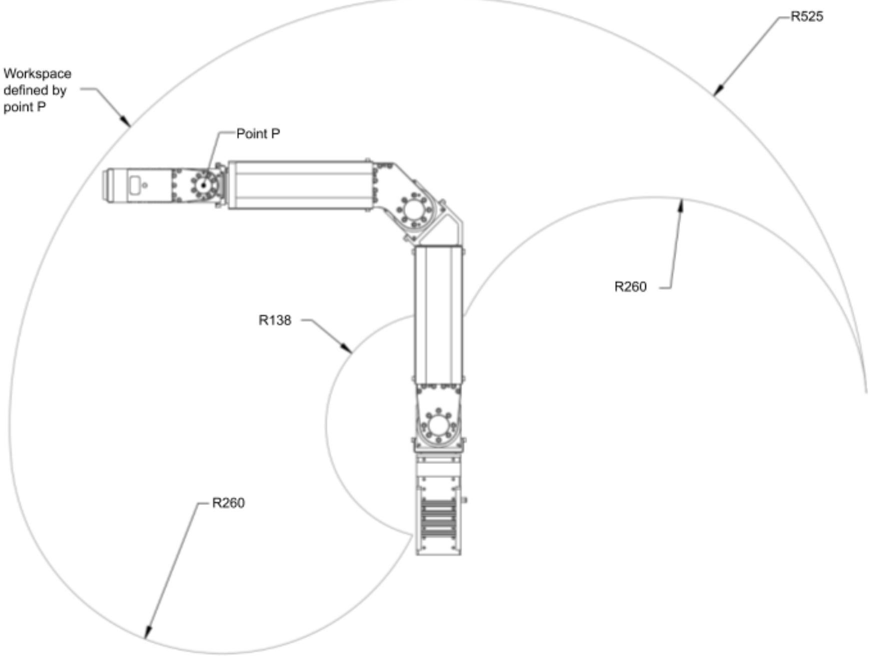


Figure 3.17

Diagram of the OpenManipulator-P's Workspace from both the Top and Side

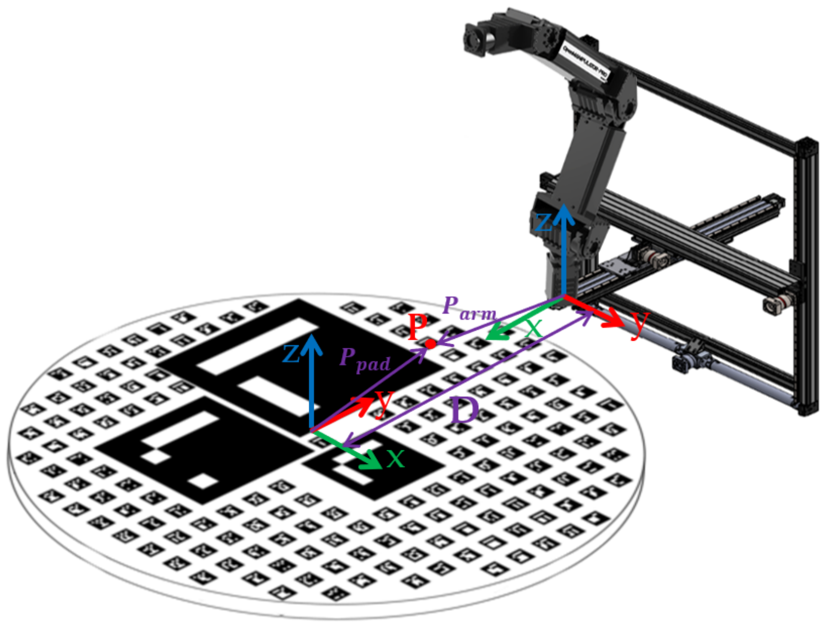


Figure 3.18

Diagram Showing the Battery Exchange System Given the Drone's Position P

to the origin of the landing pad's frame and the robot arm's frame respectively. Additionally, the frame of the landing pad shown has a 90° z-axis rotation in relation to the arm's frame. The vector P_{pad} is supplied by the post landing script while P_{arm} must be calculated using the following transform:

$$\begin{bmatrix} P_{arm} \\ 1 \end{bmatrix} = T_{arm\ pad} \begin{bmatrix} P_{pad} \\ 1 \end{bmatrix} \tag{3.26}$$

Where $T_{arm\ pad}$ is the homogeneous transformation of the arm's frame in relation to the pad's frame [26]. If $D = [d_x \ \ d_y \ \ d_z]^T$ $T_{arm\ pad}$ can be further define by the following:

$$T_{arm\ pad} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & d_x \\ \sin\theta & -\cos\theta & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.27}$$

Here, θ represents the the angle rotated about the z axis of the pad in relation to the arm's axis. Given that $P_{arm} = [p_{ax} \ \ p_{ay} \ \ p_{az}]^T$ and $P_{pad} = [p_{px} \ \ p_{py} \ \ p_{pz}]^T$, the coordinates of the drone as seen by the arm, P_{arm}, is calculated as such:

$$\begin{bmatrix} p_{ax} \\ p_{ay} \\ p_{az} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & d_x \\ \sin\theta & -\cos\theta & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{px} \\ p_{py} \\ 0 \\ 1 \end{bmatrix} \tag{3.28}$$

Where, given the nature of the reported drone positioning, $p_{pz} = 0$.

In a case where the drone does not land directly in front of the robot arm, the landing pad must be rotated to place the drone in a reachable position. This is done by finding the angle that the coordinates of the drone make with the center of the pad and then using the pad's radius to find the edge in front of the drone. From this point, the closest marker can be determined using the Python dictionary with each marker's coordinates. Then, by aiming the OpenManipulator-P's end effector downward towards the pad's edge, the pad can be rotated until the closest marker to the edge of the pad in front of the drone has reached the camera's center. Now, knowing the distance between the pad and the arm, the pad's angle of rotation with respect to the arm, and the drone's position with respect to the pad, the drone's position with respect to the arm can be easily calculated using the above transform equation.

Given the drone's position and orientation with respect to the arm, the arm can then be instructed to seek out the drone and align for battery retrieval. To ensure the best safety for the drone however, an effective "dead space" must be implemented to keep the arm from attempting to move "through" the drone. This is done by using the stored radius and height of the drone to place a boundary where the arm cannot pass. If the arm attempts to move through this "dead space," the arm will move upwards to avoid the object.

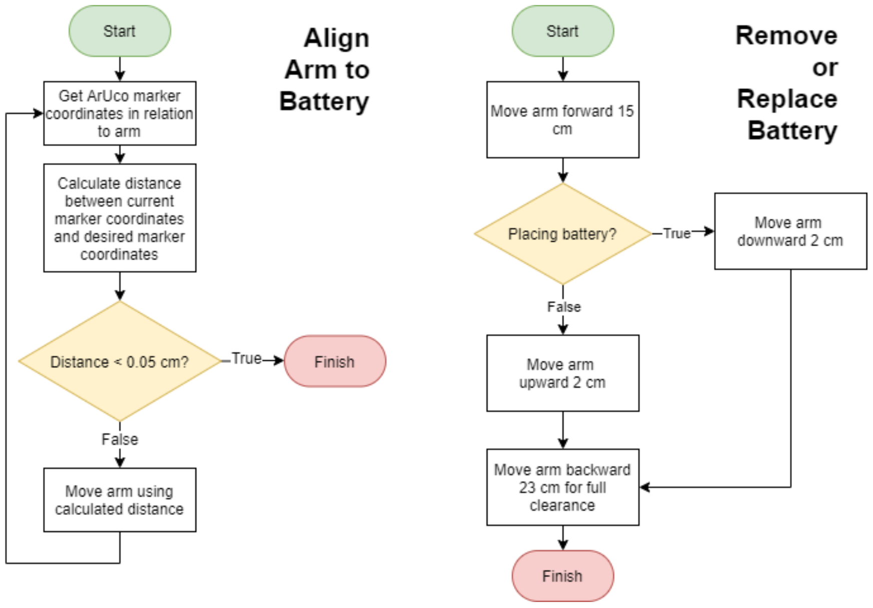

Start
Align Arm to Battery
Get ArUco marker coordinates in relation to arm
Calculate distance between current marker coordinates and desired marker coordinates
Distance < 0.05 cm?
True
Finish
False
Move arm using calculated distance
Start
Remove or Replace Battery
Move arm forward 15 cm
Placing battery?
True
Move arm downward 2 cm
False
Move arm upward 2 cm
Move arm backward 23 cm for full clearance
Finish

Figure 3.19

Flowcharts of the OpenManipulator-P's Alignment and Battery Removal or Replacement Procedures

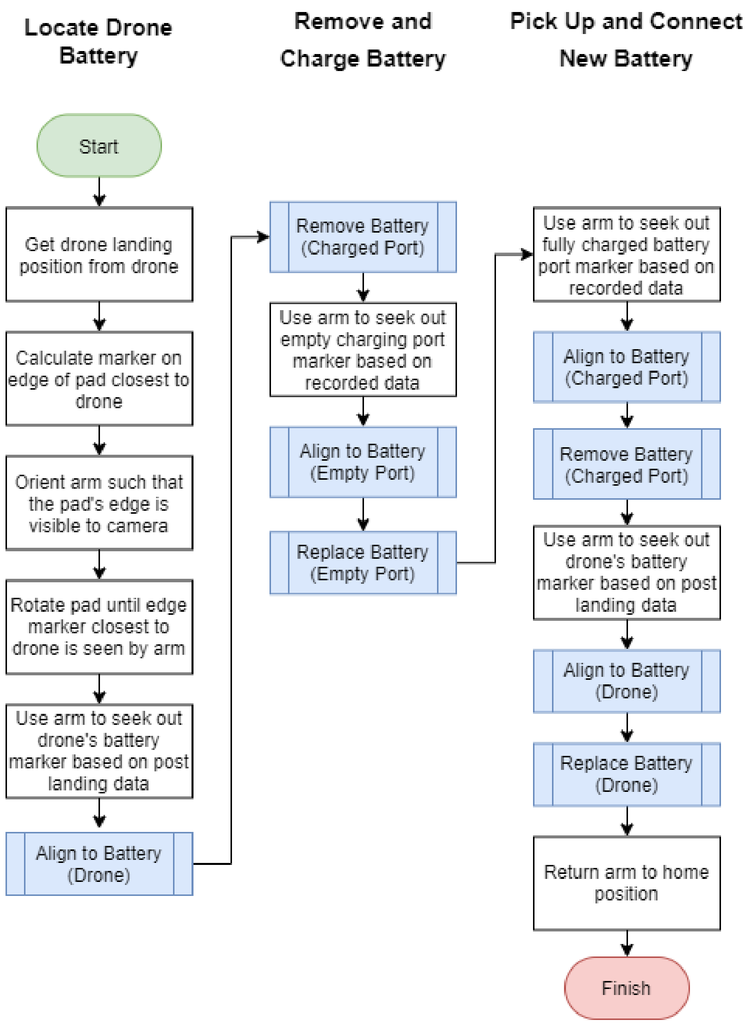

Locate Drone Battery
Remove and Charge Battery
Pick Up and Connect New Battery
Start
Get drone landing position from drone
Calculate marker on edge of pad closest to drone
Orient arm such that the pad's edge is visible to camera
Rotate pad until edge marker closest to drone is seen by arm
Use arm to seek out drone's battery marker based on post landing data
Align to Battery (Drone)
Remove Battery (Charged Port)
Use arm to seek out empty charging port marker based on recorded data
Align to Battery (Empty Port)
Replace Battery (Empty Port)
Use arm to seek out fully charged battery port marker based on recorded data
Align to Battery (Charged Port)
Remove Battery (Charged Port)
Use arm to seek out drone's battery marker based on post landing data
Align to Battery (Drone)
Replace Battery (Drone)
Return arm to home position
Finish

Figure 3.20

Flowchart of the OpenManipulator-P's Battery Exchange Procedure

The main method of control uses the OpenManipulator ROS controller with C++. This package is used to control the arm using cm coordinate instructions within the arm's workspace and roll, pitch, and yaw angles of the arm's end effector with respect to the fixed world frame. The fixed world frame refers to "starting axis" of the arm's configuration which does not change as the arm moves or rotates [26]. The ROS node that controls the arm must also receive ArUco marker information. This is done by using the ArUco detect package within the ROS fiducials package [33]. A subscriber within the arm control node is used to access the ArUco detection information that is published as a ROS topic. Once an ArUco marker has been located, the arm can align itself and remove or replace a battery as shown in the flowchart in Figure 3.19. Once the battery has been extracted from the drone, the arm must move to place the battery in an empty charging port. Each port has a distinct ArUco marker to distinguish it from the others. If available, a charged battery is then removed from a charging port and the arm places the new battery in the drone. This full procedure is further detailed in the flowchart in Figure 3.20.

CHAPTER IV

IMPLEMENTATION

4.1 ArUco Detection

Before testing with other systems, the accuracy of position estimation using ArUco markers must first be evaluated. As with any vision-based sensing, the effectiveness of ArUco marker pose estimation relies on the camera's ability to clearly see the marker. This means that the main variables to assess are the lighting conditions present and the distance between the camera and the marker.

Because this system will mainly be used outdoors during daylight, some level of light will always be present. However, this light level can vary with the time of day and amount of cloud cover. To simulate this within a lab setting, varying amounts of light are used for evaluation. Shaded lighting is accomplished by only lighting the half of the lab space opposite to the testing space, normal lighting uses all overhead lights in the lab, and bright lighting incorporates the use of a flashlight to increase brightness. Additionally, as the drone descends or the robotic arm moves towards the drone battery, the visibility of the marker will increase in clarity. Shown in Figure 4.1, a drone was hung from a telescoping mast to assess ArUco marker visibility from far away. However, while the camera was able to clearly see the markers in daylight from approximately 4 m away, this did not prove to be an effective setup to evaluate accuracy due to the hanging of the drone causing it to swing.

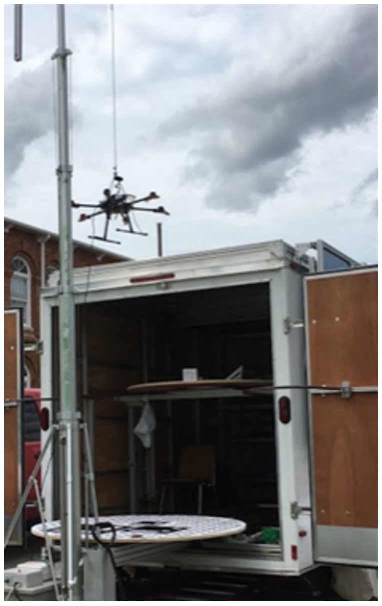


Figure 4.1

Drone Hung from Approx. 4 m on Telescoping Mast for Visibility Testing

A better setup to assess ArUco marker accuracy as affected by height is to re-purpose the 2-dimensional gantry system from the Automatic Battery Exchanger senior design project. By mounting the ArUco vision system to this gantry, it can be controlled to move up to approximately 0.4 m off the ground. Any required precision movement from the robot arm will operate within this margin, as will any position assessment from post drone-landing. Furthermore, if the accuracy of ArUco marker detection is reasonably high from a 0.4 m distance, the corrections generated at a higher distance during drone landing will most likely serve the purpose of general guidance to the landing pad.

Accuracy of pose estimation will be determined by placing an ArUco marker directly under the camera used for detection such that the x and y displacement values are equal to zero. The h value is determined by the height supplied by the 2D gantry system. Accuracy error is calculated by finding the cm distance between the expected coordinates $[0, 0, h]$ and the reported coordinates by OpenCV $[x, y, z]$. The formula used for error calculation is as follows:

$$error = \sqrt{(0-x)^2 + (0-y)^2 + (h-z)^2} = \sqrt{x^2 + y^2 + (h-z)^2} \tag{4.1}$$

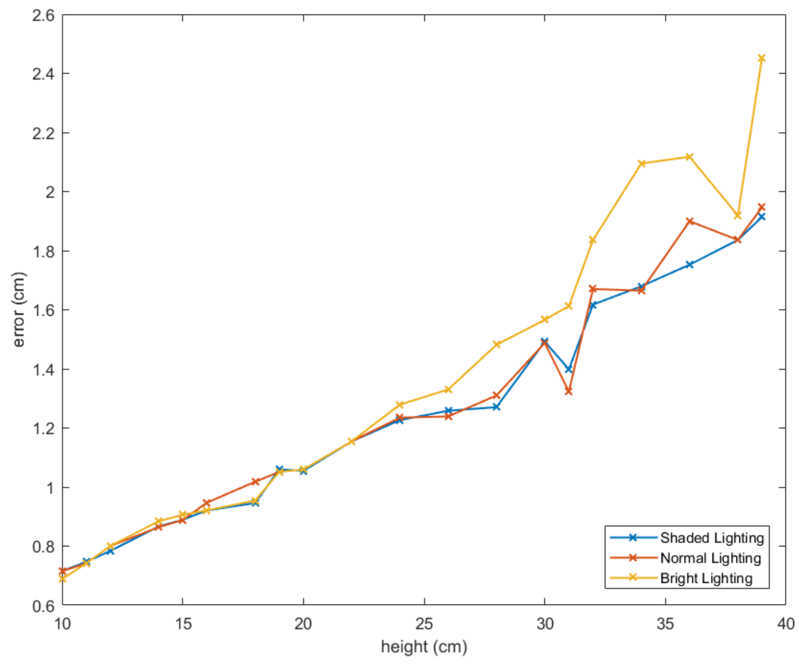


Figure 4.2

Plot of ArUco Detection Error in Different Lighting with Increasing Height

The graph in Figure 4.2 shows the calculated error for each brightness level plotted against the camera height. The heights range from 39 cm to 10 cm with an interval of 1 to 2 cm. Bright lighting appears to produce the most error overall while shaded lighting appears to produce the least error overall. Additionally, the error produced readily decreases as the camera approaches the marker, showing that ArUco detection increases in accuracy the closer to the marker it is. The highest error reported is a 2.453 cm difference from the expected value in bright lighting at a height of 39 cm. The lowest error reported is a 0.689 cm difference from the expected value in bright lighting at a height of 10 cm.

Regarding the precision of ArUco detection in varying lighting conditions, three distance data sets were calculated between each lighting condition (shade/normal, normal/bright, shade/bright) using the same distance equation used for error calculation. Then, the variance of the three distances for each height were calculated to show the precision of ArUco detection. The results of this data are shown in Figure 4.3 where the calculated variances show that the detected points are reasonably precise in varying lighting conditions and that precision appears to increase with decreasing height.

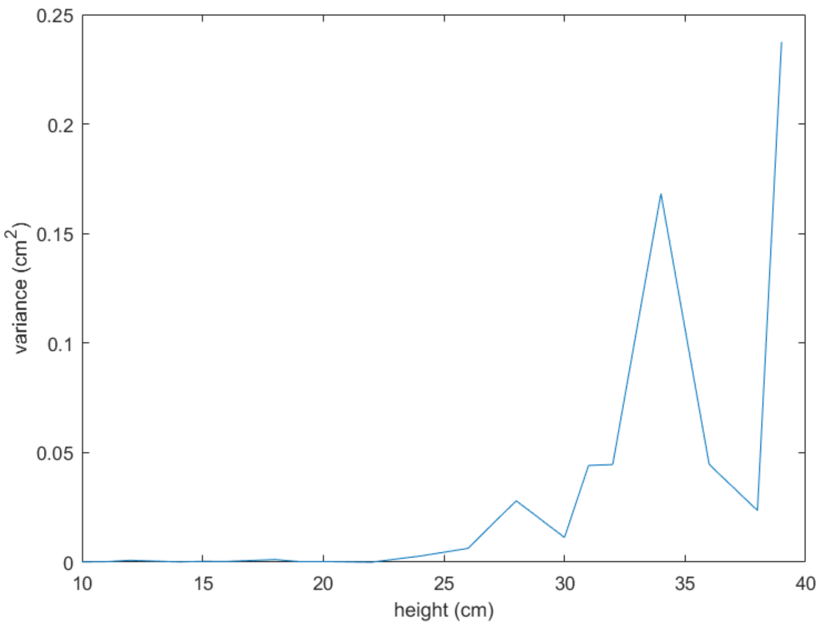


Figure 4.3

ArUco Detection Precision Data with Increasing Height

At a camera distance of 39 cm, the average error reported is approximately 2 cm. While this is higher than ideal, the average error reported decreases nearly linearly as the camera approaches the ArUco marker. For landing purposes, this level of accuracy will be sufficient for guidance to the landing pad. Post-landing position calculation typically operates in the 20 cm - 10 cm range, which shows to have approximate errors of 1 cm - 0.7 cm. This level of accuracy will be sufficient so long as the battery exchange procedure accounts for potential error. The level of accuracy shown will also be sufficient for use with battery exchange if measures are taken to handle greater errors at farther distances.

4.2 Drone Landing

Evaluating the effectiveness of the drone landing procedure is less quantitative than evaluating the ArUco marker detection accuracy. This is because the success of the drone landing requires only that the drone land on the landing pad. While it is preferred that the drone land within the designated "safe zone" of the pad, this condition is not enacted by the landing procedure and is instead evaluated during post-landing. Because of this, the drone landing procedure will be evaluated by recording the success rate of the drone landing on the landing pad over a designated number of trials.

While initially these trials were performed outdoors on a landing pad separate from the trailer's landing pad to some success, ultimately this setup proved to be too volatile for effective testing. Field testing is a necessary component to evaluating the full capability of this system, but it introduces too many variables for reliable prototyping.

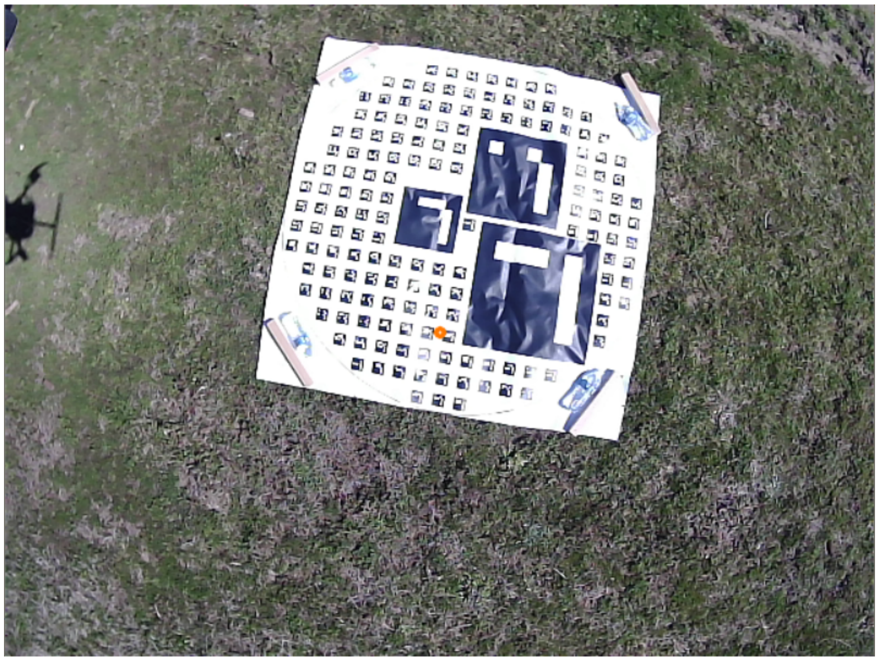


Figure 4.4

Example of Poor Visibility due to Uneven Terrain

During field tests, the landing pad used for testing is printed on the same anti-reflective matte tarp as the pads installed in the trailer, except this test pad is not mounted onto a wooden board. To stabilize the tarp and prevent warping during takeoff, the corners are weighed down or staked into the ground. However, if the terrain underneath the pad is too uneven or if the pad shifts and wrinkles due to gusts of wind or the drone's takeoff, the camera may have difficulty detecting the ArUco markers. Figure 4.4 shows an example of poor visibility on uneven terrain in bright lighting conditions. This issue is made less severe by placing a small board under the pad that fits under the 3 larger ArUco markers.

It should be noted that the majority of field tests performed have been done using an alternate drone communication setup where the DroneKit scripts are executed by a remote computer with a telemetry module tuned to the drone's flight controller. This differs from the current setup where the Raspberry Pi communicates with the drone's flight controller directly through serial. The initial setup was done using a Python 2 UDP server over Wi-Fi to receive the visual data from the Raspberry Pi and then broadcast corrections to the drone through DroneKit. This was done to avoid losing control of the drone in case of losing connection with the Raspberry Pi, but due to network and multitasking issues this system was not deemed feasible. The current system was verified in field to ensure that a hand controller could overtake the system if necessary before proceeding with direct Raspberry Pi control. Several sample flight scripts were run using the direct Raspberry Pi connection to the drone and each time a hand controller could override the DroneKit commands. This ensures the safety of the drone within this system while providing reliable drone control.

Controlling the drone using GPS based MAVLink prompts has also proven difficult. The drones' GPS module is only accurate to 2.5 m [1] and is unable to make small, minute corrections for landing purposes. MAVLink controls have the ability to control the motor speeds directly, but there has yet to be an opportune time to test this given recent weather developments and battery constraints. Field testing has proven difficult enough that work was put into installing an indoor GPS system to test drone flights within the lab, but this was ultimately deemed unfeasible due to time constraints and the indoor GPS system's early development stage.

Demonstrated in Figure 4.5, an intermediate solution was found in that the drone can be hung from the top of the lab's drone cage above the test landing pad. This way, the error calculations and generated vehicle instructions can be verified without needed to account for weather conditions or battery life.

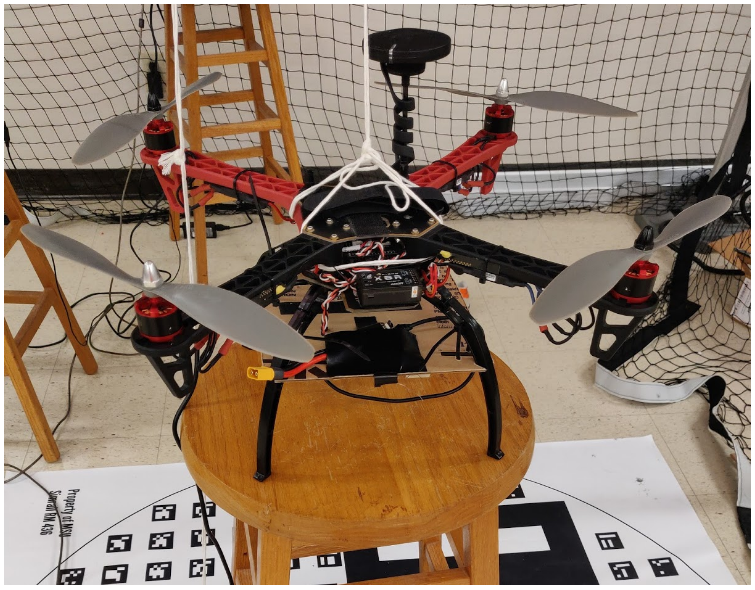
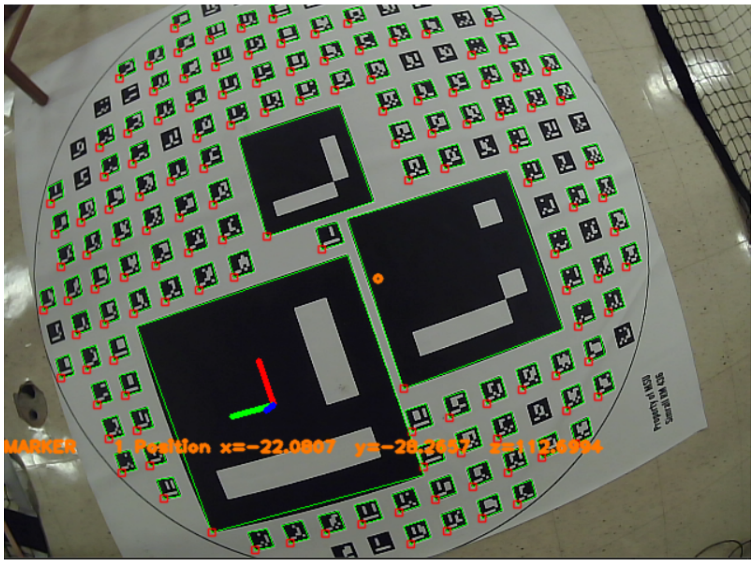


Figure 4.5

A Drone Shown Attached to a Rope and Hung from the top of a Drone Cage

Before error handling could be verified using this testing method, the corrections needed to be verified first. This was done by orienting the drone above a specific marker in the Northeast, Southeast, Southwest, and Northwest directions relative to the marker and rotating the drone on its Z axis to observe the generated cm corrections. A successful trial would output the opposite direction of the drone's orientation above the marker. For example, if the drone were in the marker's Northwest quadrant, the instructions should tell the drone to fly Southeast to reach the marker's center. This particular test verifies that the rotation of the drone does not alter the generation of instructions with respect to the drone's position above the marker. Each orientation above the marker generated correct instructions with a 100% success rate for a full turn of the drone's Z axis.

Once the directional corrections were verified, the error handling was tested. This was done by raising the drone up in increments and observing the edge where the descension error is deemed



```
MARKER    1 Position x=-22.0807  y=-28.2657  z=112.6994
```

Figure 4.6

Example of the Drone’s Visibility above the Lab Test Pad

low enough. As a starting point, the maximum angle of descent was set to 5°. This is because the drone’s position should be within the "radius" of a given marker. Given that the largest marker is 50.75 cm, the second largest 38 cm, and the third largest 24.5 cm, the maximum distance the drone should be from the center of the marker should be 25.375 cm at 3 m above ground, 19 cm at 2 m above ground, and 12.25 cm at 1 m above ground. This is referencing the heights at which the drone switches the marker ID to look for. With an angle of 5°, the maximum error allowed is 26.25 cm at 3 m, 17.5 cm at 2 m, and 8.75 at 1 m. Because the drone is a hanging object, taking physical measurements to determine accuracy is extremely difficult. However, general accuracy can be estimated by comparing the calculated maximum distance from a marker’s center to the reported distance by OpenCV. If the detected border distance is reasonably close to the calculated distance while remaining within the boundary of the active marker, the error handling for descension control can be seen as reasonably accurate.

Table 4.1

Landing Automation Descension Criteria Evaluation

Height (cm)	Calc. Dist. (cm)	Meas. Dist. (cm)	Error (cm)
168.860	13.659	14.773	1.115
155.833	12.947	13.634	0.686
136.158	11.849	11.912	0.063
123.941	10.150	10.843	0.693
111.785	9.663	9.780	0.117
103.748	8.784	9.077	0.292
93.114	7.935	8.146	0.211
89.450	7.815	7.826	0.011
80.298	6.760	7.025	0.265
59.002	4.912	5.162	0.250
42.902	3.196	3.753	0.558

Table 4.1 shows the error distances calculated directly using the descension criteria angle and the measured error distances reported by OpenCV. These measurements began at approximately 40 cm above the landing pad and increase to approximately 170 cm. It should be noted that the measured distance is defined by the position of the drone once the landing script deems the error low enough to descend, and as a result, is somewhat an approximation. However, the difference between the calculated and measured distances are shown to be under 0.5 cm for the majority of trials run. Additionally, during testing, the center point of the camera lens remained within the boundaries of the active marker for each trial. This suggests that the reported cm error distance from the center of the active marker is reasonably accurate.

4.3 Post-Landing

Initial development of the post-landing procedure began by recording the the positions of each ArUco marker on the landing pad into a Python dictionary. Because the markers are arranged in rows and columns, the central marker was used as an origin point with the rows above and the columns to the right represented as positive units. Likewise, the rows below the central marker and the columns to the left are represented as negative units. One unit is equal to approximately 8.5 cm, which is the distance between the centers of adjacent markers.

To evaluate the effectiveness of the post-landing procedure, both the ability to calculate the position of the drone on the pad and designate whether the drone is within the "safe-zone" or not needs to be evaluated. The "safe-zone" of the landing pad is defined by a given radius within the landing pad where the drone is contained safely enough to draw into the trailer. If safe, the radius of the drone is represented by a green circle in the post-landing user interface, however, if unsafe, the drone is represented by a red circle.

The testing setup for the post-landing system involves mounting the image processing system to a drone and placing it in various areas on the lab's test landing pad. The height of the camera on the drone is approximately 10 cm. The success rate of "safe-zone" designation is defined by the number of correct identifications out of a total of 20 trials. The accuracy of position estimation on the pad is evaluated by comparing the output position to the measured position of the system on the pad and calculating the distance between the two as the resulting error.

To measure the camera's position on the pad, a reference point is printed to the output camera feed such that it represents the center of the camera lens. After recording the OpenCV output of the camera position, a meter stick is lined up in the camera frame to measure from the center point

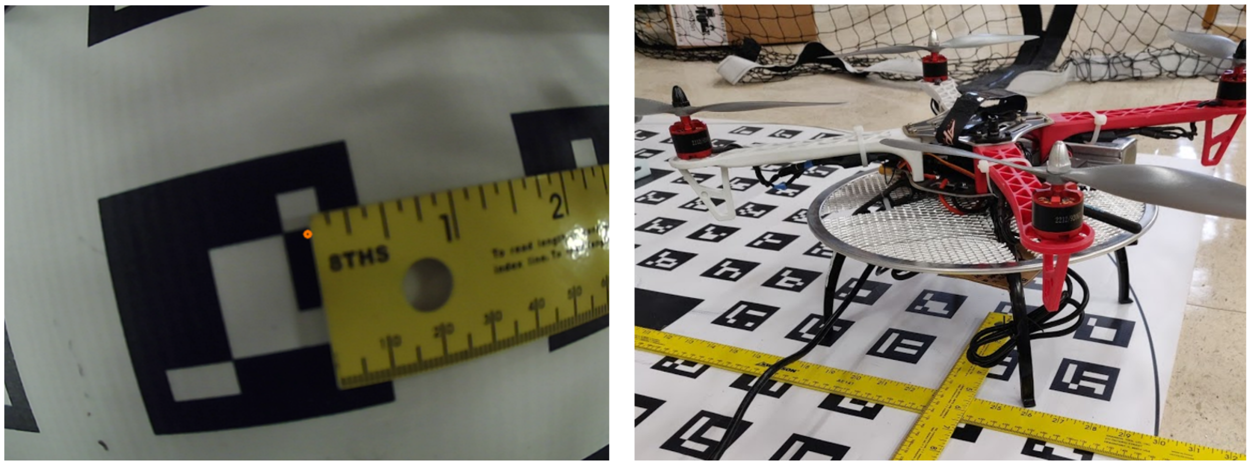


Figure 4.7

Method of Measuring Drone Camera Position

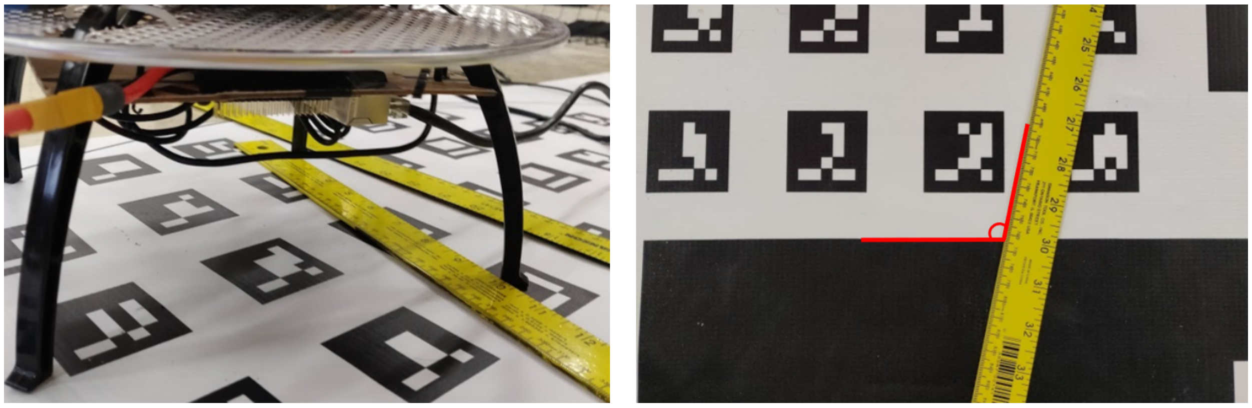


Figure 4.8

Method of Measuring Drone Heading Angle

to the X axis and record the y displacement. The X axis is represented on the landing pad by an additional meter stick that is then used to measure the x displacement. A demonstration of this method is shown in Figure 4.7 where the leftmost image shows the camera frame's reference point and the rightmost image shows the measurement setup on the pad.

To measure the drone's heading angle, a meter stick is placed against the drone's support legs pointing towards the front of the drone. This supplies the correct angle of heading. This angle can then be measured by evaluating it with respect to any straight edge present on the landing pad. The implementation of this method is shown in Figure 4.8 where the leftmost image shows the meter stick placement and the rightmost image shows the method of angle evaluation.

While the heading angle error is calculated by simply taking the difference between the measured angle and the angle reported by OpenCv, the position error is calculated by evaluating the distance between the measured point $[x_m, y_m]$ and the point reported by OpenCV $[x_{cv}, y_{cv}]$. This can be done using a simple 2D distance equation as such:

$$error = \sqrt{(x_m - x_{cv})^2 + (y_m - y_{cv})^2} \tag{4.2}$$

Table 4.2

Safe-Zone Identification Success Rate and Position Accuracy Data

Success	Failure	Success Rate	Position Error	Heading Error
20	0	100%	0.719 cm	3.159°

Figure 4.9 illustrates the reported positions by OpenCV compared to the measured positions while Table 4.2 shows the success rate of "safe-zone" identifications and the average accuracy of

position and heading estimation. The average calculated position error of 0.719 cm at a height of 10 cm is slightly higher, but reasonably close to the error calculated by the ArUco accuracy tests. Post-landing position calculation is predominately for determining the safety of the drone and for battery exchange. Both processes employ methods to allow for a small amount of error, so positioning errors up to approximately 1 cm are considered reasonable. Additionally, the average heading error is reported as 3.159° which is negligibly small enough to consider reasonable.

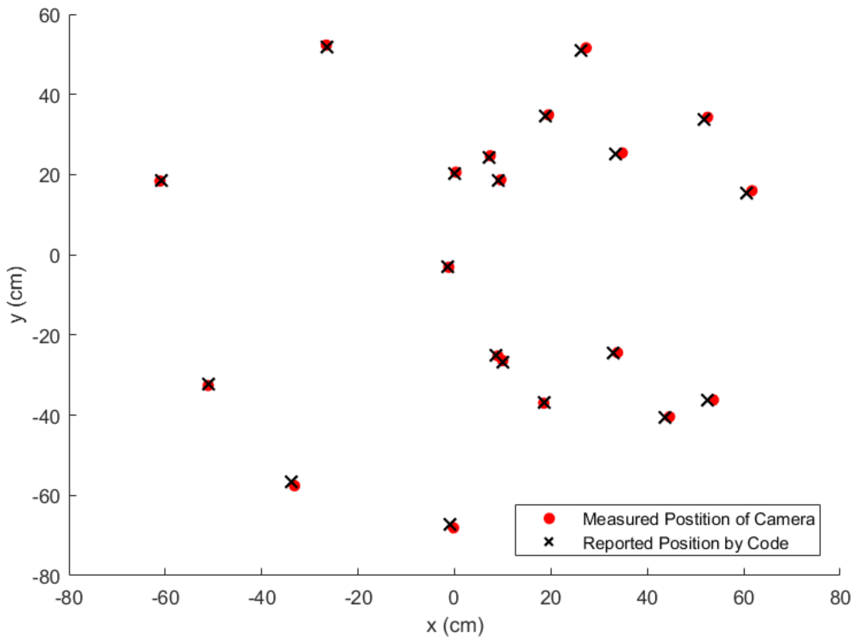


Figure 4.9

Measured Position of Camera on Landing Pad as Compared to the Calculated Position

4.4 Battery Exchange

The autonomous battery exchange system also operates under qualitative terms. Because the success of removal and replacement of a battery is defined by the robot arm's ability to completely remove and securely place a battery, the evaluation of this system should also be done through success rate.

The ideal testing setup for battery exchange evaluation requires that a drone with the necessary battery port mounted be placed in front of the robotic arm in a given orientation and position on the landing pad. The position of the drone will be reported to the arm and the arm will then attempt to remove the battery and replace it. However, during development of this system, one of the OpenManipulator's motors malfunctioned and had to be shipped out for repairs. The lack of a physical, working arm restricts the possibility of full system testing and required additional planning.

The ROS controller package for the robotic arm includes everything necessary for operating in the 3D robotics simulator Gazebo. This way, the full battery replacement setup can be constructed in a simulation environment to verify the system's implementation.

In addition to the OpenManipulator-P ROS package, a test model for the landing pad had to be made for testing. This was done by making a to-scale mesh of the pad in Blender and applying the landing pad ArUco markers as a texture. This mesh could then be used to generate a Gazebo object with a motor controller compatible with ROS. A test "drone" was also designed and exported from Blender for use in testing. A simulated camera was also attached to the bottom of the drone to determine the position of the drone on the landing pad. Using the landing pad model, the drone model, and the OpenManipulator-P model, the simulation results in Figure 4.10 were achieved using the geometrical transforms detailed in Chapter III.

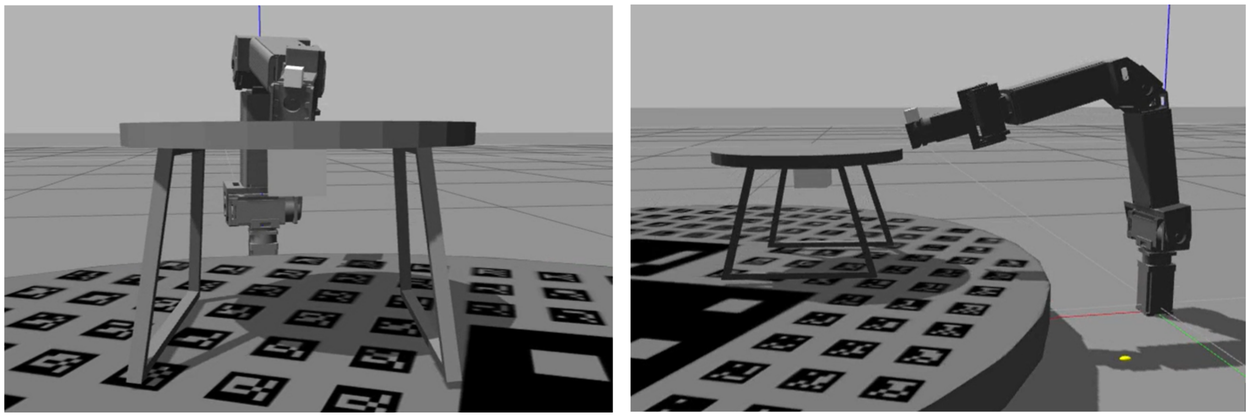


Figure 4.10

Simulation Results of OpenManipulatorP Seeking Drone

Figure 4.10 shows the previously discussed test setup where the robot arm has successfully located and aligned to the the landed drone. This shows that the calculations used to translate the drone's coordinates to that of the arm's coordinate system are ultimately successful for general guidance. However, using this particular setup makes reaching the drone from more difficult positions nearly impossible due to the arm's short reach. Without the implementation of the gantry system, the arm is severely limited in range. Additionally, the position of the drone in Figure 4.10 shows the last link of the drone at an angle upwards rather than parallel with the drone. To successfully remove a battery, the arm's end effector must be in-line with the drone's battery port. Implementing the gantry system could also potentially fix this problem by allowing the arm to have more range of motion within the space. The image on the left also shows the OpenManipulator-P's last link at a Z axis angle slightly off from the drone's Z-axis angle. This is most likely due to ArUco detection and estimation error and should be corrected using the drone's on-board ArUco marker.

Before being disassembled for repairs, the physical OpenManipulator-P was able to successfully remove a battery from a drone's battery port using an ArUco marker as guidance. This occurred successfully a total of 5 times before disassembly. These in-lab battery removals combined with the progress achieved with the robot arm simulation, the use of the OpenManipulator-P robotic arm for variable drone battery exchange shows great promise.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

5.1 Summary of Study

The aim of this project was to further develop an autonomous drone landing and battery exchange system that is applicable to a wide variety of drone sizes and shapes. This was first done by identifying the major problems and solutions present in current research in this area. Drone autonomy has been a widely sought-after research topic and there are several branches to take in accomplishing it. The use of MAVLink protocol libraries to autonomously control drones and the use of OpenCV for image processing and pose estimation are among the many common research endeavors. Additionally, design elements from relevant projects that operate adjacent to this project within its lab of operation must also be considered to ensure compatibility.

Using these elements, the autonomous drone landing and battery exchange system was designed and constructed. Each subsystem required detailed procedural planning for best design and implementation. While this project is heavily based on hardware, it also relies greatly on software to ensure the hardware works as intended. These systems were then tested and evaluated to verify that the success rate of each system was reasonably high for safe and effective operation.

5.2 Discussion of Results

Because ArUco marker detection is used so heavily within the context of this project, first the accuracy of detection given varying camera distances and lighting conditions must be evaluated. By placing an ArUco marker directly under the center of the camera lens, the camera can be moved upwards and the OpenCV output values can be recorded. The results show that the closer the camera is to the marker the more accurate the estimation. Additionally, brighter lighting conditions appear to produce somewhat more error, however the estimated points for any given height were reasonably precise despite the lighting conditions. While the highest reported error is approximately 2 cm, this is still reasonably small enough to justify use with the landing, post-landing, and battery exchange procedures so long as proper measures are taken to account for the error.

While there has been minor success in autonomously landing a drone, there have been major issues regarding both field testing and GPS control of the drone. While field testing is necessary to ensure full capability of the drone landing system, the volatile nature of testing outdoors makes prototyping significantly harder. Additionally, while controlling the drone autonomously through GPS commands works well for general flight, the GPS module on the drones is not accurate enough to make the necessary adjustments for correcting position. Testing the drone's flight within the lab has shown to not be a feasible option, so verification of the drone landing script must be done otherwise. By hanging the drone from the top of the lab's drone cage, the program's output in response to the detected ArUco markers can be evaluated and verified before attempting to fly again. The flight instruction generation was verified by observing the program's response after positioning the drone above a marker and rotation the drone about its Z axis. This resulted in correct directional instructions in every tested position and orientation. The error handling of the landing process was verified by recording the reported distance away from the marker at the reported height on the boundary of acceptable error. This distance was compared to the calculated maximum acceptable distance and resulted in reasonably small error.

The post-landing procedure is verified by comparing the OpenCV calculated position to the measured position on the pad and evaluating the generated "safe-zone" status. The "safe-zone" status reports whether or not the drone has landed in a position that is considered "safe" to pull the landing pad back into the trailer. During testing, the "safe-zone" determination was correct 20 out of 20 times, resulting in a 100% success rate. Additionally, the average position error calculated was equal to 0.719 cm, which is within the range of acceptably small errors. The average heading angle error was reported to be 3.159° which is also small enough to consider reasonable.

While considerable success was achieved with the OpenManipulator-P while it was functional, after the motor malfunction the testing setup had to be reconsidered. Because the OpenManipulator ROS package includes the necessary files for Gazebo simulation, the remainder of the battery exchange system was constructed within a 3D simulation test-space to further verify functionality. This test-space included a ROS controlled landing pad model and test "drone" model. Both the "drone" model and OpenManipulator-P model were equipped with virtual cameras for detecting the landing pad's ArUco markers. If placed in a convenient position, the robotic arm's ROS control node can successfully detect and align to the drone. However, the lack of a functional gantry within the test-space greatly limits the movement of the arm. Additionally, the ArUco detection and subsequent calculations results in slight error in angle calculation when the drone's arm aligns to the drone. This can be further corrected using the drone's on-board ArUco marker as reference.

5.3 Future Work

While the post-landing sequence functions well and as intended, both the landing and battery exchange sequences require refining in future work. The landing sequence as it stands has decent framework in that the directional corrections and error handling work well, but the execution of this sequence has yet to be fully tested. When testing in the field, malfunctions can be difficult to diagnose and debug, so future testing should seek out an alternate, controlled environment. There have been recent plans to attempt test flights at a local indoor flight facility with a larger drone cage, but this a very new development in the current project and has not yet been explored.

Additionally, while the GPS module installed on the majority of drones in the lab does not have an accuracy high enough to make refined corrections, it should be noted that the field tests done with this module have had additional software issues regarding flight instruction generation. Taking this into account, the GPS module used may be able to provide sufficient corrections for successful landing dependent on the maximum angle of error. This possibility should be evaluated in future testing. However, if the GPS module provided does not allow for successful landing, MAVLink commands to directly control the directional velocities should be further researched and evaluated.

Regarding battery exchange, the main priority is in extending the arm's reach with a 3D gantry system to better locate the drone's battery. With the inclusion of the gantry, the OpenManipulator's movement will have to be modified to account for the gantry's additional reach. This could theoretically be implemented by determining the maximum distance the robot arm's base can be from the drone, moving the arm using the gantry as needed, executing initial movement of the arm, and then using the drone's ArUco marker to "level" the arm's end effector. The "levelling" of the end effector will likely include re-positioning the arm using the gantry, but it is difficult to determine the extent without direct testing.

Furthermore, while the battery port used for battery exchange serves its purpose when stationary, it would not perform well in flight. The construction of the current battery port has no mechanism to prevent the battery from displacing itself or falling out of the drone while in motion. Future work with this project could be dedicated to improving the battery port and casing design to provide a more secure battery housing for drone flight.

Additionally, the post-landing procedure's user interface only shows the drone's position on the pad after landing. If desired, this could be further expanded to track the drone's movement above the landing pad in real-time. This would provide a more dynamic visual and show the drone's landing process as it descends.

REFERENCES

[1] "Here2 GNSS GPS Module,", https://www.getfpv.com/here2-gnss-gps-module.html, Accessed: 2021-03-06.

[2] "Mission Commands,", https://ardupilot.org/planner/docs/common-mavlink-mission-command-messages-mav_cmd.htmlnavigation-commands, Accessed: 2021-02-06.

[3] "PM06 v2 Power Module,", http://www.holybro.com/manual/PM06V2_PowerModule.pdf, Accessed: 2021-01-30.

[4] "Raspberry Pi 4 Model B,", https://www.raspberrypi.org/products/raspberry-pi-4-model-b/, Accessed: 2021-01-30.

[5] 3DRobotics, "DroneKit-Python Documentation,", https://dronekit-python.readthedocs.io/en/stable/, Accessed: 2021-01-28.

[6] T. Addabbo, S. De Muro, G. Falaschi, A. Fort, E. Landi, R. Moretti, M. Mugnaini, F. Nicolelli, L. Parri, M. Tani, M. Tesei, and V. Vignoli, "An Automatic Battery Recharge and Condition Monitoring System for Autonomous Drones," *2020 IEEE International Workshop on Metrology for Industry 4.0 IoT*, 2020, pp. 1–5.

[7] R. V. Aragón, C. Rocha Castaño, and A. C. Correa, "Impact and Technological Innovation of UAS/Drones in the World Economy : VI International Conference on Innovation and Trends in Engineering (CONIITI 2020)," *2020 Congreso Internacional de Innovación y Tendencias en Ingeniería (CONIITI)*, 2020, pp. 1–4.

[8] V. Becerra, "Autonomous Control of Unmanned Aerial Vehicles," *Electronics*, vol. 8, 04 2019, p. 452.

[9] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[10] S. Y. Choi and D. Cha, "Unmanned aerial vehicles using machine learning for autonomous flight; state-of-the-art.," *Advanced Robotics*, vol. 33, no. 6, 2019, pp. 265 – 277.

[11] K. L. B. Cook, "The Silent Force Multiplier: The History and Role of UAVs in Warfare," *2007 IEEE Aerospace Conference*, 2007, pp. 1–7.

[12] M. Day, "Extracting Euler Angles from a Rotation Matrix," 2014.

[13] X. Dong, Y. Ren, J. Meng, S. Lu, T. Wu, and Q. Sun, "Design and Implementation of Multi-rotor UAV Power Relay Platform," *2018 2nd IEEE Advanced Information Management,Communicates,Electronic and Automation Control Conference (IMCEC)*, 2018, pp. 1142–1146.

[14] ELP, "ELP High Speed 120 FPS PCB USB 2.0 Webcam Board,", http://www.webcamerausb.com/elp-high-speed-120fps-pcb-usb20-webcam-board-2-mega-pixels-1080p-ov2710-cmos-camera-module-with-21mm-lens-elpusbfhd01ml21-p-78.html, Accessed: 2021-01-09.

[15] T. Fiorenzani, "How Do Drones Work,", https://github.com/tizianofiorenzani/how_do_drones_work, Accessed: 2021-01-26.

[16] K. Fujii, K. Higuchi, and J. Rekimoto, "Endless Flyer: A Continuous Flying Drone with Automatic Battery Replacement," *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*, 2013, pp. 216–223.

[17] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, 2014, pp. 2280 – 2292.

[18] K. M. Hasan, W. S. Suhaili, S. H. Shah Newaz, and M. S. Ahsan, "Development of an Aircraft Type Portable Autonomous Drone for Agricultural Applications," *2020 International Conference on Computer Science and Its Application in Agriculture (ICOSICA)*, 2020, pp. 1–5.

[19] Y. Ju, H.-J. Mun, and K.-H. Han, "Image Processing Based Drone Landing Technique Considering GPS error and wind Direction," 2019.

[20] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 6235–6240.

[21] A. Moffatt, E. Platt, B. Mondragon, A. Kwok, D. Uryeu, and S. Bhandari, "Obstacle Detection and Avoidance System for Small UAVs using a LiDAR," *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2020, pp. 633–640.

[22] D. Oh and J. Han, "Fisheye-Based Smart Control System for Autonomous UAV Operation.," *Sensors (14248220)*, vol. 20, no. 24, 2020, p. 7321.

[23] OpenCV, "Camera Calibration,", https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html, Accessed: 2021-01-21.

[24] OpenCV, "Camera Calibration and 3D Reconstruction,", https://docs.opencv.org/master/d9/d0c/group__calib3d.html, Accessed: 2021-01-21.

[25] F. C. Park and K. M. Lynch, *Modern Robotics: Mechanics, Planning, and Control*, chapter 2: Configuration Space, Cambridge University Press, Cambridge, UK, 2017, p. 29.

[26] F. C. Park and K. M. Lynch, *Modern Robotics: Mechanics, Planning, and Control*, chapter 3: Rigid-Body Motions, Cambridge University Press, Cambridge, UK, 2017, pp. 76–82.

[27] PX4, “Hex Cube Black Flight Controller,”, https://docs.px4.io/master/en/flight_controller/pixhawk-2.html, Dec 2020, Accessed: 2021-01-09.

[28] Rafael Muñoz Salinas, “ArUco Library Documentation,”, https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc, Accessed: 2021-01-21.

[29] B. Rao, A. G. Gopi, and R. Maione, “The societal impact of commercial drones,” *Technology in Society*, vol. 45, 2016, pp. 83 – 90.

[30] D. Raveendhra, M. Mahdi, R. Hakim, R. Dhaouadi, S. Mukhopadhyay, and N. Qaddoumi, “Wireless Charging of an Autonomous Drone,” *2020 6th International Conference on Electric Power and Energy Conversion Systems (EPECS)*, 2020, pp. 7–12.

[31] ROBOTIS, “OpenManipulator-P Specifications,”, https://emanual.robotis.com/docs/en/platform/openmanipulator_p/specification/, Oct 2020, Accessed: 2021-01-09.

[32] H. Tanaka and Y. Matsumoto, “Autonomous Drone Guidance and Landing System Using AR/high-accuracy Hybrid Markers,” *2019 IEEE 8th Global Conference on Consumer Electronics (GCCE)*, 2019, pp. 598–599.

[33] J. Vaughan, “ROS Fiducials Package Summary,”, http://wiki.ros.org/fiducials, Accessed: 2021-01-30.