

VIPLE Extensions in Robotic Simulation, Quadrotor Control Platform, and
Machine Learning for Multirotor Activity Recognition

by

Matthew De La Rosa

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2018 by the
Graduate Supervisory Committee:

Yinong Chen, Chair
James Collofello
Dijiang Huang

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

Machine learning tutorials often employ an application and runtime specific solution for a given problem in which users are expected to have a broad understanding of data analysis and software programming. This thesis focuses on designing and implementing a new, hands-on approach to teaching machine learning by streamlining the process of generating Inertial Movement Unit (IMU) data from multirotor flight sessions, training a linear classifier, and applying said classifier to solve Multi-rotor Activity Recognition (MAR) problems in an online lab setting. MAR labs leverage cloud computing and data storage technologies to host a versatile environment capable of logging, orchestrating, and visualizing the solution for an MAR problem through a user interface. MAR labs extends Arizona State University's Visual IoT/Robotics Programming Language Environment (VIPL) as a control platform for multi-rotors used in data collection. VIPL is a platform developed for teaching computational thinking, visual programming, Internet of Things (IoT) and robotics application development. As a part of this education platform, this work also develops a 3D simulator capable of simulating the programmable behaviors of a robot within a maze environment and builds a physical quadrotor for use in MAR lab experiments.

ACKNOWLEDGMENTS

I would like to express my thanks and appreciation to my supervisor, Professor Yinong Chen for his support, encouragement, and guidance in the fulfillment of the works presented in this thesis. I would also like to extend my gratitude to the other members of the committee, Dr. James Collofello and Professor Dijiang Huang. A special thanks goes out to friends, family, and colleagues whose continued support has helped me grow as an individual. A final thanks goes to Arizona State University faculty members who have made this possible.

TABLE OF CONTENTS

	Page
LIST OF TABLES	v
LIST OF FIGURES	vi
CHAPTER	
1 INTRODUCTION	1
2 BACKGROUND	5
VIPLE and VIPLE's Unity Simulator	5
Quadrotor Control and Applications.....	8
Activity Recognition	10
3 MULTIROTOR ACTIVITY RECOGNITION	12
4 IMPLEMENTATION	18
Quadrotor Build: PID Flight Controller and VIPLE Interface	18
VIPLE-Quadrotor Data Collection Platform	20
MAR Lab AWS Environment	21
5 EXPERIMENT AND METHODOLOGY.....	25
MAR Training Lab Module	26
MAR Classification Lab Module.....	29
6 RESULTS AND DISCUSSION.....	33
7 LIMITATIONS	37
8 CONCLUSION AND FUTURE WORKS	39
REFERENCES	41

TABLE OF CONTENTS

	Page
APPENDIX	
A QUADROTOR HARDWARE REVISIONS	43
B LOSS AND ACCURACY OVER EPOCHS	50
C MAR LAB ENVIRONMENT SETUP GUIDELINES.....	54

LIST OF TABLES

Table		Page
3.1.	Quadrotor-VIPLE Motion Config Options	15
6.1.	Accuracy and Loss over Epochs	35
B.1.	Accuracy and Loss over Epochs Dataset	51

LIST OF FIGURES

Figure	Page
2.1. VIPLE and its Compatible Devices	6
2.2. VIPLE Unity Simulator v1.x – Maze Use-Case	7
3.1. MAR Lab Architecture Overview	13
3.2. VIPLE Implementation of a Series-Based Flight Plan	14
3.2.1. SimpleWait Custom Activity	14
3.2.2. Takeoff Custom Activity	14
3.2.3. Land Custom Activity	15
3.3. Hello Jupyter Practice Module.....	17
4.1. Active EC2 Instance Running MAR Labs Jupyter Notebook Server.....	22
5.1. Raw IMU Data Plot.....	30
5.2. IMU Data Flight Plan Classification	32
6.1.1. Simple Flight Plan Classification on HMC Dataset.....	33
6.1.2. Simple Flight Plan Classification on HRC Dataset.....	34
A.1. Annotated Components of Flight Controller Shield.....	45
A.2. Hardware Diagram of Flight Controller Shield.....	45
A.3. Quadrotor PID Tuning Test Rig.....	46
A.4. Quadrotor Assembly Revision 1	47
A.5. Fractures in 3D Printed Frame	48
A.6. Quadrotor Assembly Revision 2	48
A.7. Quadrotor Revision 2 Hardware Parts Nulling	49
C.1. Architecture Diagram of AWS MAR Lab Environment	55

LIST OF FIGURES

Figure	Page
C.2. Quadrotor IAM User	56
C.3. MAR Lab S3 Bucket	56
C.4. Public Elastic IP Address	57
C.5. MAR Lab EC2 Instance	57

CHAPTER 1

INTRODUCTION

Training and applying machine learning models to solve classification problems often require a combination of skills in data collection devices, software tools, software-hardware communication, programming, and data analysis. The solution to classification problems often manifest in the general process of collecting data, calculating features from the data, and formatting the features in a manner compatible with a classifier. While the tools and libraries that exist allow data scientists to perform these tasks, there are few tools available that are capable of teaching high school and college freshman students how to train and apply these classifiers within the IoT and robotics fields. Writing the scripts necessary to orchestrate such data flow and manipulation, setting up a special runtime environment with the necessary software libraries used for training machine learning models, and meeting the specialized hardware specifications necessary for training classifiers on a large data set all add to the technical hurdle of teaching the core principles of understanding machine learning.

Existing machine learning tutorials also suffer from a narrow solution scope making curriculum difficult through example due to the vast range of methodologies available in data acquisition and manipulation. Such tutorials like TensorFlow's MNIST [1] and RapidMiner's titanic example [2] have a highly curated experience where users work with pre-generated and pre-formatted data with pre-defined interactions in solving specific hypothetical problems. This takes away from the first-hand experience of obtaining or generating the data, manipulating the data, and prepping a classifier for training and classification applications. Often, this process is hidden behind a black box

that lacks the transparency, flexibility, and depth necessary to show students the process of transforming raw data into a format that is compatible with the classifier.

Teaching machine learning through first-hand experience requires a versatile learning environment with the ability to both orchestrate the machine learning process and expose the underlying mechanisms in driving the orchestration. This includes generating the data for lab use, storing and managing this data, building an interactive environment capable of manipulating the data while showing users that process, and using that environment to train and apply a linear classifier.

To teach users how to train and apply a linear classifier, lab activities in this study are framed within the Multirotor Activity Recognition (MAR) problem. MAR is an extension of the activity recognition classification problem and aims to classify windows of raw Inertial Measurement Unit (IMU) data recorded by a quadrotor's on-board Micro-electromechanical Systems (MEMS) sensor to identify what action the quadrotor is taking. The MAR lab environment operates within an educational context and performs tasks on the user's behalf. These tasks implement the solution to solving the MAR problem but require users to run the scripts and inspect the console output. The work is done to extend the capacity of ASU VIPLE (Visual IoT/Robotics Programming Language Environment) to perform machine learning and data analysis.

Users are expected to generate data by implementing a flight plan in VIPLE to control a quadrotor. The corresponding IMU data for that flight session is then retrieved by the MAR lab environment. This training data can then be used to create a flight plan classifier, thus creating a feasible solution to the MAR problem. While the scope of this paper covers machine learning orchestration, in-depth exploratory data analysis will be

excluded from the discussion. Quadrotor flight optimizations are also not considered to be in scope, although these lab sessions could potentially provide beneficial reports to aid in that endeavor.

The works featured in this thesis describe a system capable of reducing the complexity involved in implementing a solution to the MAR problem. By doing so, MAR labs aims to increase the range of audience that these tools can be used by (specifically instructor-led labs). It is also the intention of these labs to increase the productivity in data analysis by expediting the machine learning process through orchestration. Finally, by building and compiling a collection of flight sessions within the MAR database, this paper provides a framework for collecting IMU data produced by multirotor-based platforms. By building a database of IMU data, future iterations of MAR classifiers can be trained with more data to create a more accurate classifier.

This thesis expands on the VIPLE platform by implementing a Unity simulator for supporting robotics programming in maze navigation and adding supported devices by building a physical drone capable of generating data necessary for machine learning experiments. This thesis also aims to explore the merits of implementing a fully orchestrated machine learning pipeline where users interact through activity labs to generate their own data, manipulate that data, and apply that data to train and use a linear classifier by allowing users to interact with data directly, in real-time, and allowing them to make changes to the machine learning pipeline to obtain a better understanding of what processes take place behind the black boxes obscured by existing machine learning libraries and tools.

The rest of the thesis is organized as follows. Chapter 2 discusses the background, the primary components involved in developing MAR labs, and the motivations for implementing the project. Chapter 3 describes Multirotor Activity Recognition and provides the context necessary to better understand the design decisions made when implementing the MAR labs with respect to quadrotor controllers, activity recognition, and existing machine learning applications in quadrotors. It also develops the solution for the MAR problem within an educational context and describes the architectural overview of the project in its entirety. Chapter 4 outlines the implementation, where components of MAR labs are described individually and in more detail. Chapter 5 presents the experiment and methodology. It describes the process of setting up MAR labs and collecting IMU data to use in MAR labs. The effectiveness of the VIPLE & Quadrotor platform, and the MAR labs in solving the MAR problem are discussed in this chapter. Chapter 7 summarizes the limitations of the research and experiments. Finally, this paper closes with a conclusion that covers contributions and future works. Details on the quadrotor build and revisions made to the quadrotor are further explained in the Appendix.

CHAPTER 2

BACKGROUND

ASU VIPLE is a visual based IoT and robotics programming tool that allows students to build programs and connect to hardware devices through a visual interface [3]. The purpose of the tool is to teach the programming process with focus on the development and deployment of programs onto a variety of robots and IoT devices [4]. The visual based aspect of VIPLE allows students to click and drag components of a program together to build robotic applications.

To inspire application development in robotics, VIPLE has been used to teach high school and college freshman students the programming skills necessary to create applications. As shown in figure 2.1, several simulated and physical devices have been developed and used for robotic applications such as NXT robots, humanoid sensors, and integrations with IoT devices such as Alexa's voice control [5][6]. However, recent years have seen the wide-spread application of multirotors, Unmanned Aerial Vehicles (UAVs), and machine learning as platforms for more recent robotic developments. Such control mechanism for VIPLE's quadrotor and machine learning platform are not present.

VIPLE and VIPLE's Unity Simulator

Contributions of this research made to VIPLE include the unity simulator and the physical quadrotor, shown in the top-left corner of figure 2.1. The VIPLE Unity Simulator was developed with the goal of introducing users to the concept and usage of VIPLE as a programming language and to simplify the technical hurdle of programming IoT and robotics devices by allowing users to run VIPLE code in a virtual environment

without having to physically build a robot and its testing environment. ASU VIPLE documents, tutorials, and software downloads are available at:

<http://neptune.fulton.ad.asu.edu/VIPLE/>

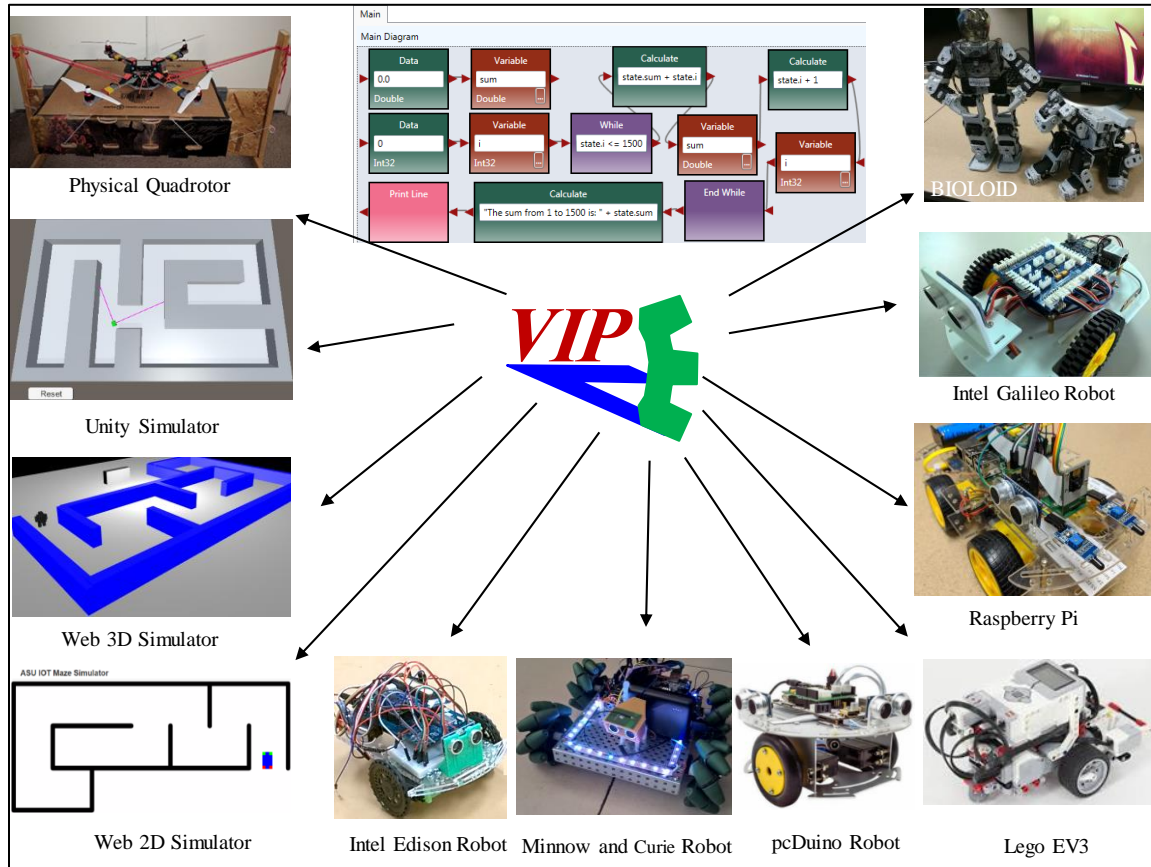


Figure 2.1: VIPLE and its Compatible Devices

The VIPLE Unity Simulator is currently being used in FSE 100 classes with great success. The first implementation and iterations of the VIPLE Unity Simulator were designed for the maze use-case. In this scenario, users were able to interact with the maze environment by adding and removing walls while interactions with the robot (the small green block) were strictly controlled by the VIPLE process. Like all other VIPLE clients, the VIPLE simulator was embedded with a TCP interface, allowing VIPLE to send

commands to the virtual robot over a localhost connection. The simulator is also responsible for communicating simulated sensory readout back to the VIPLE host over the TCP bridge.

Contributions made the VIPLE Unity Simulator included designing, architecting, and implementing the first iteration as a proof of concept. The VIPLE Unity Simulator was developed in C# on top of the Unity game engine with the use of the UnityEditor. Unity was chosen as the primary framework for creating a 3D virtual environment due to its simple framework library and cross platform compatibility between windows and mac. As part of the initial proof of concept, in-game assets, implementation of user interactivity with the virtual maze map (spawning walls, providing visual cues), and creating controllers that allowed the virtual robot to make measurements of the virtual environment in real-time. Figure 2.2 provides a more detailed visual of the maze use-case.

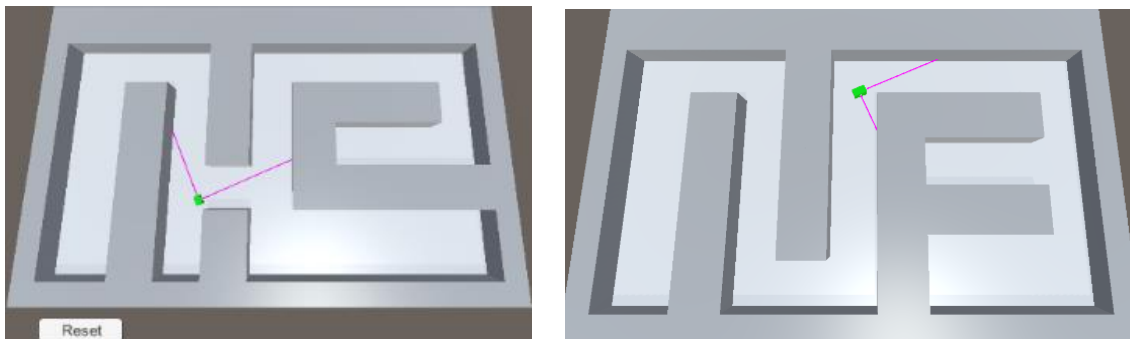


Figure 2.2: VIPLE Unity Simulator v1.x – Maze Use-Case

The objective presented to students is to implement the wall-following algorithm through VIPLE’s programming interface. Two virtual distance sensors, one in the front and one on the right, were implemented to record and send real-time sensory readouts to

VIPLE from within the virtual environment, allowing the user to implement a simple wall-following algorithm. The maze consists of square blocks that can be reconfigured by simply clicking the wall or open space to remove or add a block. Figure 2.2 shows two different configurations.

A fully operational VIPLE Unity Simulator has been developed and since been handed off to the undergraduate capstone projects at ASU for further development. In pursuit of more challenging goals, the focus of this thesis shifted from providing VIPLE with a simulator to expanding on VIPLE as a tool, to teach machine learning by building a quadrotor capable with VIPLE and using the quadrotor-VIPLE platform as a starting tool for teaching machine learning.

Quadrotor Control and Applications

Quadrotors are a robotic platform constructed from four rotors and placed in an arrangement that allows them to take flight. Actions such as roll, pitch, and altitude changes are motion controls enabled by the 6 degrees of freedom. This is achieved through a series of computational corrections calculated by an on-board flight controller equipped with an IMU sensor. Such computations are typically governed by a specialized PID Controller that is responsible for maintaining the setpoint of a quadrotor's state. These controllers were chosen for their robust nature [7]. Multi-rotors provide six degrees of freedom give them a high degree of controllability and versatility.

This makes the platform ideal for surveillance and sensory applications. These applications involve use cases where an elevated platform is required to provide hard-to-reach observations at a low cost, and possibly at scale. Such applications include the

agricultural industry where autonomous quadrotors can be used for performance measuring on plantations [8]. These quadrotors allow for automated video capture, making it much easier for researchers and users to collect real-time data and make real-time assessments. In cases where costs of obtaining aerial videos are a concern, quadrotors become a necessary tool[9]. Quadrotors can also be used to address use cases where a quick, responsive, and mobile robotic unit can be deployed in emergency situations [10].

As multirotor-based solutions have been proposed in a growing number of research opportunities, a fair number of studies have been dedicated to their robustness. While precise mechanics allow a quadrotor to maintain a stable flight, works such as [11] focus on sub-optimal conditions that quadrotors may experience. In this case, the correctional feedback loop mechanisms in a quadrotor is expanded to include fault detection and recovery. Such fault tolerant applications go as far as implementing a control solution for quadrotors that suffer from a complete failure in one, two, or three propellers [12]. Obstacle avoidance has also been a leading discussion in the multirotor platform. Given the automated nature of most applications and the complexity of navigating a 3D space with six degrees of freedom, there exists solutions to solving the quadrotor collision avoidance problem [13].

With VIPLE, students can implement control algorithms to solve real problems such as maze traversal and direct keyboard-based controllers. These qualities of VIPLE make it an ideal tool for deploying simple control schemes to control a quadrotor. Hence extensions and contributions made to VIPLE involve creating a quadrotor and implementing a VIPLE compatible interface on the quadrotor. The simplicity of

controlling robots through the VIPLE platform makes it an ideal tool for generating the data used in MAR labs. In this work, VIPLE's primary purpose is to serve as a controller for the quadrotor. IMU data is stored locally on-board the quadrotor during each flight session while Amazon Web Services (AWS) are used to create the appropriate linear classifier. With these two environments, the MAR lab's machine learning pipeline can encompass the entire process of data collection, data manipulation, training a linear classifier, and using said classifier to solve the MAR classification problem.

Activity Recognition

A common application of classifiers is their deployment in activity recognition. Typically, these studies involve the deployment of IMU sensors on a subject with the idea that specific actions produce a recognizable IMU signature. One such example is the study on human activity recognition (HAR) [14][15] where the goal is to classify human activities such as sitting, walking, or taking an elevator. Other solutions to HAR include the breakdown of tasks into subtasks [16]. Typically, these studies aim to provide the framework for context aware applications. The activity recognition problem also can be extended towards canines as [17] shows, IMU data can be used to aid in the recognition of canine activities such as standing or sitting through unsupervised learning. Linear classifiers have been chosen as the specific classifier due to its common application in supervised learning and availability within TensorFlow's estimator API. The supervised nature of the data stems from tagging windows of IMU data as belonging to a specific activity. The classifier is given the appropriate classification in parallel with the data being given, and the label for a given class is a discrete.

The goal of MAR labs is to deploy similar sliding window sampling techniques to recognize multirotor activity. The sliding window sample considers the frequency at which the data was recorded, the length of the window, and IMU data collection techniques. MAR extends existing activity recognition by translating IMU data signatures to quadrotor activities. MAR was chosen since IMU data has a strong correlation to a specific flight pattern as discussed in the PID Controller section. The strong correlation between the IMU-based PID Controller and the activity recognition of IMU data implies that the classification should be simple, where a given action tends to produce a unique IMU signature.

CHAPTER 3

MULTIROTOR ACTIVITY RECOGNITION

Multi-rotor Activity Recognition (MAR) is the primary focus of the thesis. The MAR problem aims to identify the actions performed by a quadrotor in flight by recognizing windows of raw IMU data as belonging to a specific class of multirotor activity. During flight, a quadrotor's roll and pitch serve as indicators to what the quadrotor is doing. Specific activities such as strafing left tend to produce windows of IMU data that indicate a negative roll. Likewise, strafing right tends to produce a positive roll value. These indicators will serve as the premise for training a feasible classifier that can make these distinctions in data and thus implement a solution to solve the MAR problem.

To investigate the MAR problem within an educational context, training and applying the model are built as two separate learning lab modules: the MAR Training Lab Module, and the MAR Classification Lab Module. A cloud-based solution was used to allow users with web-enabled devices to participate in the lab without the need for additional hardware or software. Users participate in a lab session by running prepopulated scripts within each lab module and inspecting the script's console outputs. In both lab modules, the quadrotor's flight controller properties file is configured to record labelled IMU data for supervised training, or unlabeled IMU data for classification exercises. To build an environment suitable for both the VIPL-quadrotor platform and AWS, figure 3.1 below provides a visual description of the system's architecture overview capable of machine learning orchestration.

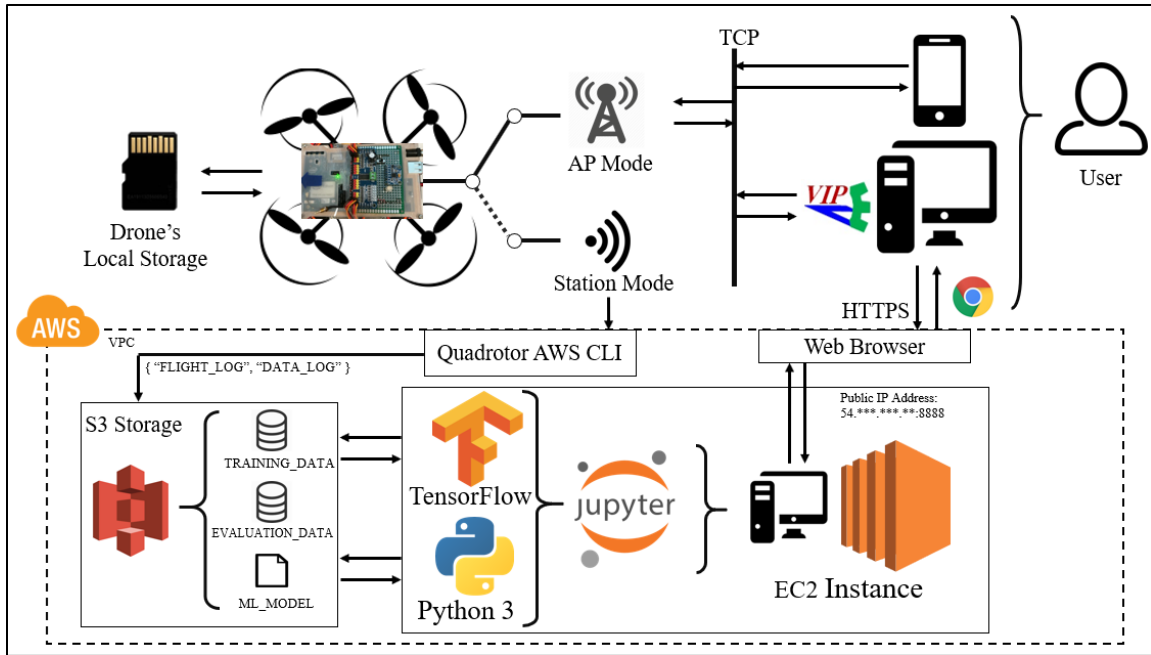


Figure 3.1: MAR Lab Architecture Overview

The start of the MAR pipeline is the VIPLE and quadrotor platform used to create a flight plan, execute the flight plan, and generate IMU data. As a control tool, VIPLE provides the user with the ability to create and run a given flight plan with instructions on which activities to execute (pitch, roll, etc.). Collectively, figures 3.2 and all sub-figures 3.2.x below show an example VIPLE implementation of a flight plan that the quadcopter platform can execute. In this sequential program, the quadrotor is commanded to move up, hover, move to the left, hover, move to the right, hover, land, then terminate the flight session.

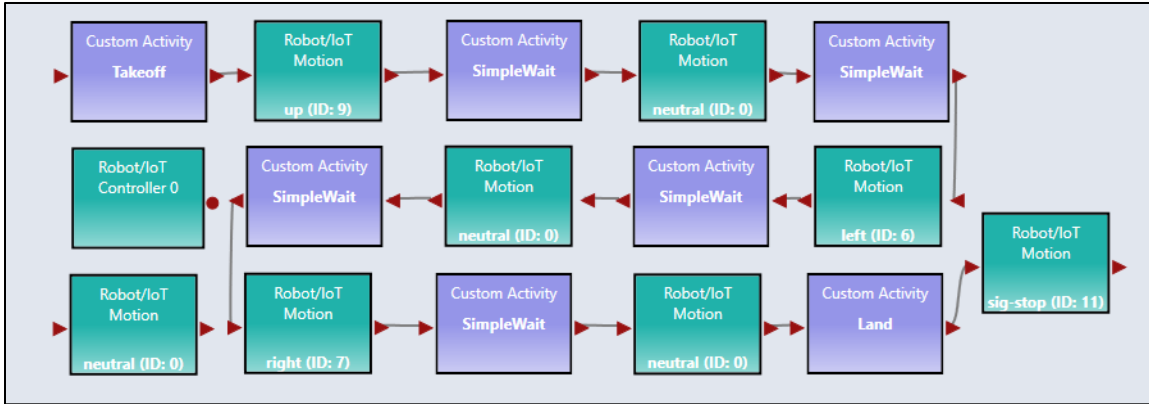


Figure 3.2: VIPLE Implementation of a Series-Based Flight Plan



Figure 3.2.1: SimpleWait Custom Activity

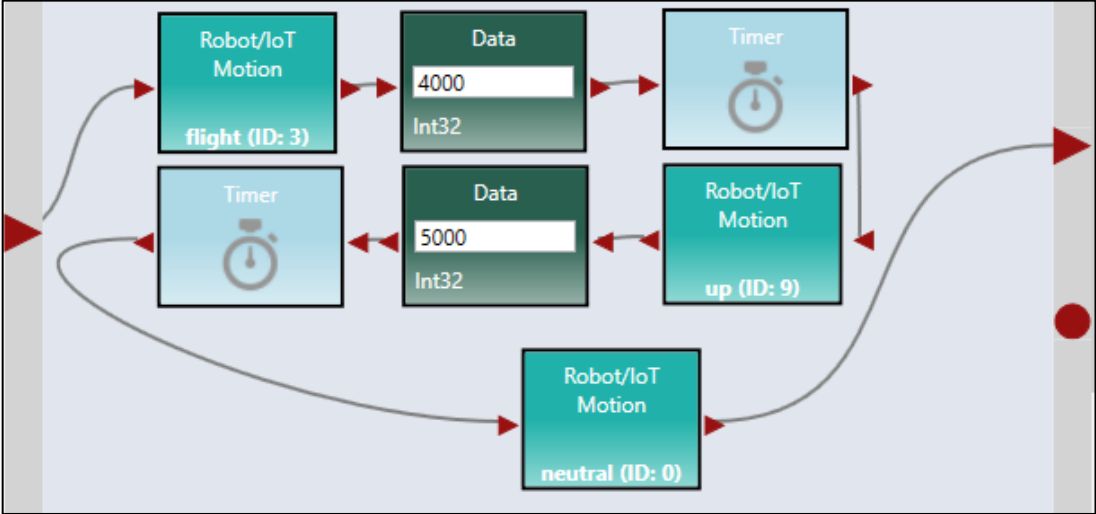


Figure 3.2.2: Takeoff Custom Activity

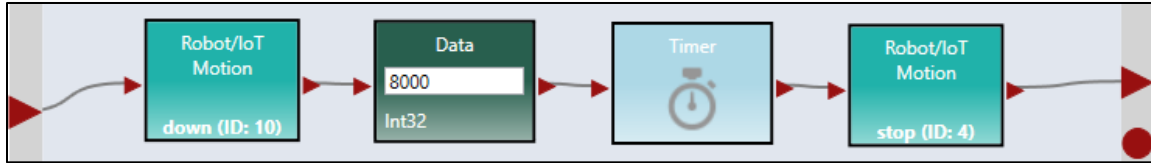


Figure 3.2.3: Land Custom Activity

Within each Robot/IoT Motion block, a custom command is defined to describe what activity the quadrotor should be performing. Implemented on the quadrotor’s VIPLE interface, an interpreter parses the JSON string received over TCP and writes to shared memory the current directive received from VIPLE. All available actions and descriptions are shown in table 3.1.


Table 3.1: Quadrotor-VIPLE Motion Config Options

Motion ID	Motion Name	Description
0	neutral	Default receive signal (do nothing) and hover during flight
1	false-flight	Enter a flight session, but with rotors disabled
2	test	Run the test script that activates each rotor in order of [1, 2, 3, 4]
3	flight	Signal flight controller to enter takeoff mode
4	stop	Signal a soft stop to the flight controller, enter landing mode
5	forward	Move the quadrotor forward, decrease pitch
6	left	Move the quadrotor left, decrease the roll
7	right	Move the quadrotor right, increase the roll
8	backward	Move the quadrotor backward, increase the pitch
9	up	Increase the quadrotor baseline throttle
10	down	Decrease the quadrotor baseline throttle
11	sig-stop	Force stop the quadrotor and end flight session

The main descriptors are timer delays that describe how long a quadrotor should be performing an action, the activity the quadrotor should be performing, and a pointer to the next timer-based activity execution. In the execution of each flight plan, two logs will be stored that describe the status of the quadrotor during a flight, and the IMU data that is directly generated from the quadrotor's on-board MEMS sensor. Both files are stored locally and uploaded at the end of each flight session. After executing a flight plan, the quadrotor begins uploading IMU data to the cloud through Amazon Web Services Command Line Interface (AWS-CLI). Further details are discussed in the next section, implementation, however it is important to note that this process expedites the time for access to usable data.

Now that VIPLE has been chosen as a suitable quadrotor control platform, an environment for the MAR lab modules must be established. Installing Python and TensorFlow library meets the requirements for supporting a framework capable of training and applying a linear classifier for MAR. On top of this runtime, Jupyter Notebooks was chosen to provide an interactive interface since scripts can be displayed and run by the user. Jupyter Notebook's markup cells provide lab sessions with the necessary textual explanation and instructions for what scripts are being executed and for what purpose, while coded cells display the code to be run along with console outputs from the given script. These code cells are prepopulated with the scripts necessary to implement MAR lab sessions, however these scripts can be edited by the user. Jupyter Notebooks can also generate reports, allowing instructors to critique the results of a given lab session. A display of every cell's output can be shown in the results for a given lab

session, even if the output contains a scripting error, or encounters an exception thrown by the script.



Hello World

The following cell is a simple python script that can print a message to the user. Try running the cell by selecting it, and running the command ctrl+enter.

```
In [1]: print('Hello World')
```

Hello World

Simple calculations:

Cells can be used to run simple computations. Try running the cell below to perform a simple 'add' calculation and print the results:

```
In [2]: print(5 + 5)
```

10

Interacting with the console:

Cells run on a Jupyter notebook can interact with the computer's runtime. Jupyter notebook uses a live python runtime allowing the user to interact with scripts running in realtime. Below, user input can be read and displayed back to the user

```
In [3]: print('Awaiting user input...')
echoMessage = input('Enter a message to echo back: ')
print('You typed [' + echoMessage + ']')
```

Awaiting user input...
Enter a message to echo back: Hello JUPYTER NOTEBOOK
You typed [Hello JUPYTER NOTEBOOK]

Figure 3.3: Hello Jupyter Practice Module

For example, figure 3.3 shows the welcome module that runs a simple “Hello World” program where each coded cell is accompanied by a markup cell to explain what the python script is aiming to accomplish. As demonstrated by this welcome module, Jupyter Notebooks can run python scripts, display the output for the given python script, perform calculations made possible through python, and even take user input directly from the user interface.

CHAPTER 4

IMPLEMENTATION

Now that the various components of the MAR lab have been outlined, this section describes in detail how the quadrotor was built, how various controllers were implemented in VIPLE for the quadrotor, how IMU data was recorded by the quadrotor and received by MAR labs, and how the MAR training and classification lab modules are tasked in solving the MAR problem.

Quadrotor Build: PID Flight Controller and VIPLE Interface

The quadrotor used to execute the flight plans was built with an Edison-Arduino board as the main controller. Four brushless rotors and Electronic Speed Controllers (ESCs) were placed on a 450mm quadrotor frame. Li-Po Batteries were used to power the quadrotor due to their space efficiency and high energy density. The Edison-Arduino board served as the primary flight controller for the quadrotor system. The on-board Wi-Fi capabilities fulfill the hardware requirement needed to establish a TCP/IP connection between the quadrotor's VIPLE client and the ground base's VIPLE host. However, a custom breakout board had to be designed to meet the additional hardware specs to include an on-board MEMS sensor and accommodations for four Pulse Width Modulation (PWM) signals needed to drive each ESC. The custom circuit board also includes an LED to serve as a visual indicator of the quadrotor's status.

To drive the quadrotor, a C++ based flight controller was implemented onto the Edison-Arduino board through Intel's System Studio IoT Edition to provide the necessary stabilization for obtaining stable flight. Configuration files were created to

grant users the ability to adjust the flight controller's parameters. The most important components to the Flight controller was the PID controller responsible for calculating the rotor corrections needed to be executed by the on-board rotors. An internal state iterator was responsible for switching between the various modes available on the quadrotor (false flight, flight, test flight, standby). The flight controller was designed to operate at 250Hz by implementing software-based timer that puts the process to sleep during each iteration of corrective calculations until the next cycle begins.

The flight controller process begins by initializing the PID controller, the internal state machine, and the shared memory space between the controller and the VIPLE interface. Two separate loggers open file streams within an S3 synced directory to record the flight controller's status and the IMU data received by the MEMS sensor. The flight controller's configurations were then read from disk or assigned a default value. These configurations included the gains for the PID controller, battery level, trim settings to prevent drifting, label configurations for files being logged, hovering throttle, and flags for uploading flight sessions automatically to S3 on exit.

The PID controller is a feedback loop that continuously reads from the MEMS sensor and the system's desired setpoint to produce a continuous output of corrective feedback. To provide the roll, pitch, and yaw controls necessary to control a quadrotor's flight, a setpoint describes the system's desired readout. In the case that a quadrotor hovers, the setpoint for the pitch, roll, and yaw remain zero. If the desired control of the quadrotor is to move forward, the setpoint of the pitch axis is set to a negative non-zero value. The PID Controller consists of three components: proportional, integral, and differential. The proportional control is responsible for calculating the difference between

the input of the MEMS sensor and the setpoint. The integral component acts as an error accumulator that allows the system to make corrections to errors that persist over a given time (this prevents controllers from only approaching a setpoint rather than achieving the desired setpoint). The derivative controller acts as a resistance to rate of change and helps dampen corrective overcompensation output by other controllers. This corrective feedback loop operates on a system-wide loop that calculates the quadrotor's current IMU readout, reads the quadrotor's setpoint, calculates the angle travelled, adjusts for drift compensation, calculates the correctional adjustments determined by the PID calculations, and applies the PID corrections to the throttle.

VIPLE-Quadrotor Data Collection Platform

Data collection begins when a quadrotor enters flight mode. During a flight session the Edison-Arduino board is constantly reading from the MEMS sensor and storing the raw value into local storage. The on-board MPU6050 can operate at a frequency of 400kHz which is well within the range of the quadrotor's correction frequency of 250Hz and provides a reading of the proper acceleration of x, y, z in G's, and the angular velocity recorded in degrees per second ($^{\circ}/s$) on the roll, pitch, and yaw axis. The MPU6050 gyroscopic sensor was configured to operate at a full-scale range of $\pm 250^{\circ}/s$ while the accelerometer was configured to run at ± 8 G's. These values are then formatted and printed through one of the flight controller's output stream into a *.csv file. Once the quadrotor enters landing mode and receives instructions to stop the flight session, or a termination signal is received, the quadrotor's flight controller

executes an exit script that invokes the AWS CLI to sync new files onto the S3 environment.

The quadrotor is also host to a VIPLE interface that interprets VIPLE's JSON-based messages sent and received over a TCP/IP connection (as described in the Approach section). This interpreter runs on a separate process and communicates directives to the flight controller over shared memory. Shell scripts orchestrate the execution of these processes on startup. If the quadrotor does not have access to a Wireless Local Area Network (WLAN), scripts were written to configure the Edison's on-board Wi-Fi antenna to enter Access Point (AP) mode, allowing the quadrotor to broadcast its own Wi-Fi network.

For VIPLE, two different types of control methods were implemented through VIPLE. The first manual-based control method is a key-mapped controller that accepted key inputs from users and translated them directly to a motion control. This allows users to have direct control over the quadrotor's actions in real time. Featured in figure 3.2, a second type of control scheme was implemented through VIPLE which describes a flight plan that the quadrotor can execute automatically. This VIPLE program implements a series of directives and timer delays and is ideal for executing multiple iterations of a given flight plan.

MAR Lab AWS Environment

To interact with the AWS cloud environment, the AWS-CLI was installed on-board the quadrotor's Edison-Arduino board. The AWS-CLI was configured with the appropriate credentials needed to perform write commands on, AWS' Secure Simple

Storage Service (S3). More shell scripts were written to proactively upload new flight session data. A pre-defined flight log file directory was chosen and monitored for file changes by storing its file directory state into local storage. A ‘diff’ between this file and the current state was performed after each flight session to search for any new flight logs. AWS-CLI upload scripts were run upon discovering new flight logs.

To interact with IMU data generated from flight sessions in the VIPLE-Quadrotor platform, S3 serves as a shared space between the VIPLE-Quadrotor platform and the AWS cloud space. The quadrotor’s AWS CLI is configured with the appropriate account settings necessary to retrieve permissions necessary to upload files onto S3. S3 makes for an ideal storage solution since it shares a domain with the EC2 environment and offers scalability necessary for plenty of data storage. Write operations are limited on the public facing interface and require that uploaders have the appropriate credentials needed to write a file. The computationally intensive data manipulation and model training can be offloaded to a virtual machine hosted on AWS’ Elastic Compute Cloud (EC2).

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	IPv4 Public IP	Key Name
	i-00760522051e073f1	t2.medium	us-east-1b	stopped		-	general-shell
	i-0960f6248205a881a	t2.micro	us-east-1b	stopped		-	general-shell
	i-097f56e32513454d2	t2.medium	us-east-1b	running	2/2 check...	54.156.204.28	jupyter-quadcopter-lab-ec2

Figure 4.1: Active EC2 Instance Running MAR Labs Jupyter Notebook Server

The VMs operate on a Linux-based operating system with the appropriate software and runtimes installed to host the MAR lab sessions. Temporary directories are used by Jupyter Notebooks to save a lab session’s state which holds checkpoints for the

trained linear classifier and IMU data transferred over from S3. The VMs come installed with Python, TensorFlow, and Jupyter Notebooks all of which provide the necessary framework needed to host MAR lab sessions as described in the previous section. VMs are also configured with a public IP address via AWS' elastic IP service allowing VM instances to interact with the public domain. Additional security configurations were made to allow public access to port 8888 giving users the ability to connect to the Jupyter Notebook server over an HTTPS connection on a web-enabled device. One such VM is shown in figure 4.1, where a t2.medium EC2 instance is running with a Public IP address. As described by AWS documentation, T2 instances are a low-cost general-purpose instance with burstable performance making it ideal for server applications. To access the MAR lab modules, a user would need to enter "https://{public_ip}:8888" into the web browser. As a minimal security measure, the Jupyter Notebook server was configured with a self-signed SSL certificate to enable HTTPS connection. Lastly, the server was configured with a simple password to prevent unauthorized changes made to the lab.

Logs displayed within each lab session provide useful insight on the execution status for a given scripted task. However, to bring better understanding to how data is being manipulated and what the data looks like, the Jupyter notebook environment includes libraries necessary to provide a visual representation of data being processed during the session. It is made clear to the user what code and scripts are being run, what task is being executed, and how that execution takes place, and what the logs report as user-readable output for inspecting the machine learning process each step of the way. In this case, the user has access to both read and write the code being run to orchestrate the

machine learning process. In addition, TensorFlow provides a log of the linear classifier's runtime whether that is training progression, or classification.

The premise for the MAR Training Module is to have users train a linear classifier to perform MAR. This linear classifier is trained using IMU data collected and described by the Implementation section. The MAR training activity module is prepopulated with the steps necessary to download the IMU data, extract samples by label into subdirectories, calculate features from each sample through sliding window protocol, and finally train a classifier.

The classification module is where users learn how to take raw, unlabeled IMU data and convert it into a plausible flight plan. Running through the module in its entirety and using the classifier from the previous training module would be a sufficient solution to the MAR problem.

CHAPTER 5

EXPERIMENT AND METHODOLOGY

The quadrotor can be configured to record labelled or unlabeled IMU data. Labelled IMU data is used to train a model through supervised learning, making the model capable of distinguishing windows of data as belonging to a specific quadrotor activity. Unlabeled IMU data is used as an exercise dataset for the classification lab module. If labels are enabled, then during flight, the quadrotor is responsible for making an extra entry of what action is being performed in each IMU readout. This tags the appropriate IMU data and labels it as belonging to a specific class of activity. Successful flight sessions and uploads contribute to the MAR database of labelled flight data. It is during the training phase, that users are expected to fly the drone, ideally with an even distribution of time spent performing the different available actions.

For each lab session, the quadrotor is powered on and initialized with a boot script that starts the VIPLE interface and flight controller process. The quadrotor is then configured to record labelled or un-labelled flight data depending on the context of the current session. A name is chosen for the flight session and set through the VIPLE interface. The flight plan is then executed and IMU data is recorded for the given session. Each MAR Lab module begins with an initialization of the runtime environment responsible for importing the appropriate libraries, defining helper functions, and local variables.

To demonstrate the feasibility of the lab session's ability to classify multirotor activities, three separate experimental procedures were performed to test the training and accuracy of the MAR classifier. The first experiments were done by manually controlling

the quadrotor through VIPLE, disabling the rotors, and holding the quadrotor device to manually mimic the quadrotor's action. This data set will be referred to as held-with-manual-control (HMC). The second experiment, called held-with-rotor-control (HRC), was recorded by enabling the rotors but holding the device to prevent drifting. The device was again controlled manually through VIPLE; however, the quadrotor's enabled rotors meant the motion of the device did not have to be manually manipulated. The third and final experiment was done by controlling the quadrotor through VIPLE, enabling the rotors, and allowing the quadrotor to fly without any type of harness; this set will be called Untethered Flight (UF).

MAR Training Lab Module

```
===[STEP 2 - DOWNLOAD TRAINING DATA SET]===
Please enter the name of your lab group's training dataset:
Group Name:
Downloading prepped training data...
Downloading from:mar-lab-workspace/exercise-training/group-training-dataset/held-with-manual-control/imu-data-log-latest
Downloading to: ./data/exercise-training-session/held-with-manual-control/imu-db/imu-database-training-set.csv
Downloading prepped evaluation data...
Downloading from: mar-lab-workspace/exercise-training/group-training-dataset/held-with-manual-control/imu-data-log-eval-latest
Downloading to: ./data/exercise-training-session/held-with-manual-control/imu-db/imu-database-evaluation-set.csv
Download completed...
===[STEP 2 - END]===

===[STEP 3 - SEPARATING DATASETS BY LABEL]===
Extracting labels for training dataset...
Extracting data labels from: ./data/exercise-training-session/held-with-manual-control/imu-db/imu-database-training-set.csv
Extracting data labels to directories in: [./data/exercise-training-session/held-with-manual-control/imu-db/]...

Extracting labels for evaluation dataset...
Extracting data labels from: ./data/exercise-training-session/held-with-manual-control/imu-db/imu-database-evaluation-set.csv
Extracting data labels to directories in: [./data/exercise-training-session/held-with-manual-control/imu-db/]...

Samples have been extracted by label
===[STEP 3 - END]===
```

The session begins by downloading both the training and evaluation datasets of raw, labelled IMU data from S3 and parsing the entries in each file into multiple sub-files

based on their tagged label. During this process, the IMU file is read line by line and stored into a labelled sample file. Once a different label is read, a new labelled sample file is created and stored in the appropriate subdirectory.

```
===[STEP 4 - EXTRACT FEATURES FOR EACH LABEL]===

Calculating features for samples in the training data directory...

Extracting features for the label [backward]...
Extracting features into the filepath ./data/exercise-training-session/held-with-manual-control/imu
-db/backward/*.csv
Extracting features for the label [forward]...
Extracting features into the filepath ./data/exercise-training-session/held-with-manual-control/imu
-db/forward/*.csv
Extracting features for the label [left]...
Extracting features into the filepath ./data/exercise-training-session/held-with-manual-control/imu
-db/left/*.csv
Extracting features for the label [neutral]...
Extracting features into the filepath ./data/exercise-training-session/held-with-manual-control/imu
-db/neutral/*.csv
Extracting features for the label [right]...
Extracting features into the filepath ./data/exercise-training-session/held-with-manual-control/imu
-db/right/*.csv
Extracting features for the label [up]...
Extracting features into the filepath ./data/exercise-training-session/held-with-manual-control/imu
-db/up/*.csv
Extracting features for the label [down]...
Extracting features into the filepath ./data/exercise-training-session/held-with-manual-control/imu
-db/down/*.csv

The features have been extracted and stored into: ./data/exercise-training-session/held-with-manual
-control/training-feature-data-latest.csv

Calculating features for samples in the evaluation data directory...

The features have been extracted and stored into: ./data/exercise-training-session/held-with-manual
-control/evaluation-feature-data-latest.csv

===[STEP 4 - END]===
```

These sample files, which now hold IMU data for a specific label, are then traversed sequentially via the sliding window protocol with a length of 4 and a step of 1. The average and median features are calculated for every IMU readout from each window and stored into a file where it will be in a format ready for training the classifier. The data is in a format that now holds a cross product of the feature and IMU data tuples: {average-acceleration-x, average-acceleration-y, average-acceleration-z, average-gyro-roll, average-gyro-pitch, average-gyro-yaw, median-acceleration-x, median-acceleration-y, median-acceleration-z, median-gyro-roll, median-gyro-pitch, median-gyro-yaw}

```

Training at epoch 0 ...
Training at epoch 1 ...
Training at epoch 2 ...
[...]
=====
Detail execution of epoch 99
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from ../mar-classification-exercise/tmp/model/held-with-manual-control/model.ckpt-35145
INFO:tensorflow:Saving checkpoints for 35146 into ../mar-classification-exercise/tmp/model/held-wit-h-manual-control/model.ckpt.
INFO:tensorflow:loss = 80.08334, step = 35146
INFO:tensorflow:global_step/sec: 89.1394
INFO:tensorflow:loss = 20.71344, step = 35246 (1.123 sec)
INFO:tensorflow:global_step/sec: 95.0501
INFO:tensorflow:loss = 90.13267, step = 35346 (1.052 sec)
INFO:tensorflow:global_step/sec: 93.2219
INFO:tensorflow:loss = 187.21191, step = 35446 (1.073 sec)
INFO:tensorflow:Saving checkpoints for 35500 into ../mar-classification-exercise/tmp/model/held-wit-h-manual-control/model.ckpt.
INFO:tensorflow:Loss for final step: 293.45258.
Evaluating model...
INFO:tensorflow:Starting evaluation at 2018-10-23-20:40:03
INFO:tensorflow:Restoring parameters from ../mar-classification-exercise/tmp/model/held-with-manual-control/model.ckpt-35500
INFO:tensorflow:Finished evaluation at 2018-10-23-20:40:07
INFO:tensorflow:Saving dict for global step 35500: accuracy = 0.8218593, average_loss = 0.5593836, global_step = 35500, loss = 111.832596
{'accuracy': 0.8218593, 'average_loss': 0.5593836, 'loss': 111.832596, 'global_step': 35500}
=====
Final Model Evaluation
  accuracy: 0.8218593001365662
  average_loss: 0.5593835711479187
  loss: 111.83259582519531
  global_step: 35500
=====
[DONE]

===[STEP 5 - END]===

```

The final step of the lab preps the linear classifier by configuring the appropriate labels, feature columns, model's checkpoint directory, and input training and evaluation file datasets. The TensorFlow API is called to train the linear classifier for 100 epochs, where an evaluation of the model is performed every 20 epochs.

The successful execution of all cells in the training lab module means that a linear classifier was successfully created for the given training data. Checkpoints for the trained model are stored by TensorFlow into a temporary directory meaning that this instance of a trained model can be restored by TensorFlow (and the subsequent classification lab module) for later use.

MAR Classification Lab Module

```
===[STEP 2 - DOWNLOADING FLIGHT LOG - START]===

Please enter the name of your lab group...
Group Name: held-with-manual-control
Please enter the specific flight session, or skip to choose the latest log...
Flight Session:
Pulling flight data from S3 for group [ held-with-manual-control ] on session [ latest ] ...
Downloading from: /mar-lab-workspace/exercise-classifying/group-classification-dataset/held-with-manual-control/flight-log-latest
Downloading to: ./data/group-workspace/held-with-manual-control/flight-log-latest.txt
Finished download flight log to flight-log-latest.txt
===== held-with-manual-control's FLIGHT LOG =====
[ENABLED] CLASSIFICATION MODE for UNLABELLED raw IMU CSV data log...
Setting Battery Gain 1 [0.985]
Setting Roll Acc_Trim [1.73728]
Setting Roll Acc_Trim [1.73728]
Setting Roll Trim [0.81]
Setting Roll Trim [-1]
Setting Roll Trim [-35.5]
Setting Take-off Trim Rotor 1 [110]
Setting Take-off Trim Rotor 2 [0]
Setting Take-off Trim Rotor 3 [70]
Setting Take-off Trim Rotor 4 [0]
Setting Flight Trim Rotor 1 [0]
Setting Flight Trim Rotor 2 [0]
Setting Flight Trim Rotor 3 [0]
Setting Flight Trim Rotor 4 [0]
Initializing PID Flight Controller...
Initializing Analog and GPIO pins...
Initializing I2C Devices...
[DONE] Initialized I2C in Fast mode....
Initializing I2C Slave Device....
Constructing Gyro...
Initializing Gyro I2C Device....
Initializing Distance Sensor....
[DONE] Initialized Gyro I2C Device....
Constructing ESC Controller...
Initializing ESC Controller I2C Device....
[DONE] Initialized ESC Controller I2C Device....
Initializing runtime variables...
[Done] Sample Time - uSeconds = 4000
[Done] Loop Timeout - uSeconds = 4000
ENTER SETUP
Calibrating Gyro Device...
Finished Gyro Callibration...
EXIT SETUP
[Warning] Running slow by -3.00562e+06
[Warning] Running slow by -4292
[Warning] Running slow by -6634
Entering Takeoff...
Entering Flight Mode...
[Warning] Running slow by -2815
[Warning] Running slow by -6591
[Warning] Running slow by -11893
[Warning] Running slow by -17036
[Warning] Running slow by -6391
[Warning] Running slow by -2163
Entering Landing Mode...
Entering Warm Standby Mode...
[Warning] Running slow by -2.99704e+06
[Warning] Running slow by -5892

===[STEP 2 - END]===
```

This module begins by asking the user for the group name and the flight session to download the appropriate flight log. Displaying the full flight logs also allows users to

confirm that the quadrotor flight was a success and that no anomalies occurred during startup.

```
===[STEP 3 - DOWNLOADING FLIGHT DATA - START]===  
  
Pulling latest flight log data from S3 for group [held-with-manual-control]  
Downloading from: /mar-lab-workspace/exercise-classifying/group-classification-dataset/held-with-manual-control/imu-data-log-04-13-21-27-13.csv  
Downloading to: ./data/group-dataset/held-with-manual-control/imu-latest-dataset.csv  
Finished downloading flight data to imu-latest-dataset.csv  
[DONE]  
  
===[STEP 3 - END]===
```

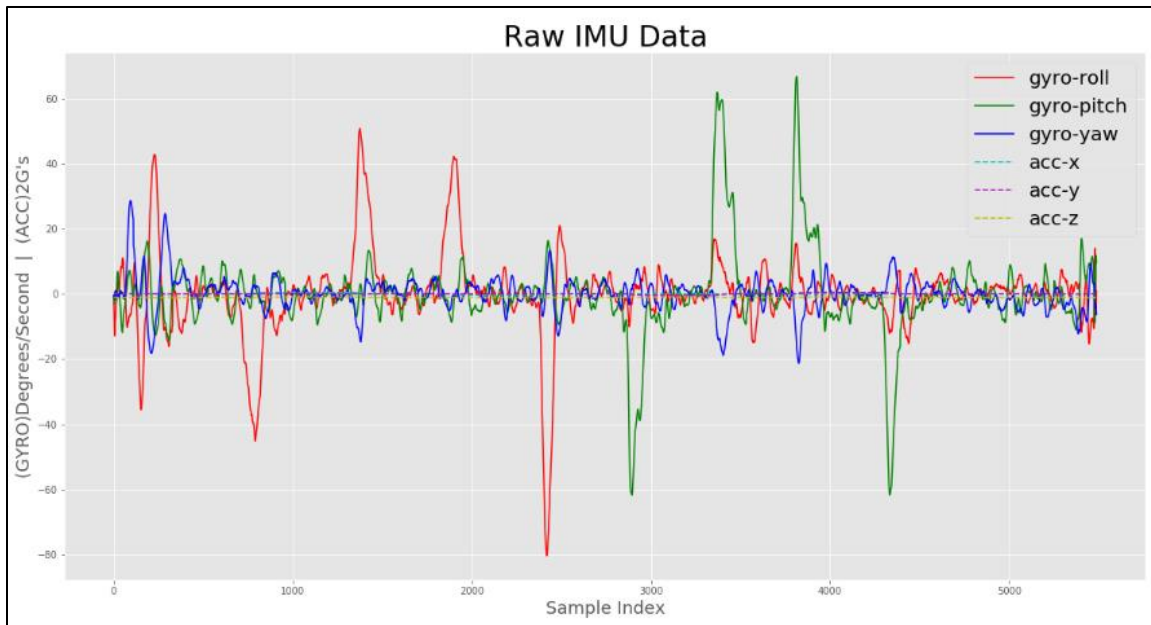


Figure 5.1: Raw IMU Data Plot

The appropriate raw, unlabeled IMU data for the given session is downloaded and a file preview is printed to show the data in its raw form. As shown in figure 5.1, a visual representation of the IMU data is then presented to the user in step 4.

```
===[STEP 5] - DEFINING FEATURE FUNCTIONS - START ===
```

```
Defining AVERAGE feature function  
Defining MEDIAN feature function  
Finished feature definitions
```

```
===[STEP 5 - DONE]===
```

A set of feature functions are defined for use in the next step. These functions are defined to take in a list of numerical decimal values as parameters and return the feature defined by the function. In the sample above, the average and median features are simply a direct implementation of calculating the average and median of the input parameters.

```
===[STEP 6] - EXTRACTING FEATURES FROM SAMPLES OF SLIDING WINDOWS - START ===
```

```
Clearing file...  
Extracting features from file [./data/group-dataset/held-with-manual-control/imu-latest-dataset.csv] into the directory [./data/group-workspace/held-with-manual-control/imu-and-features-dataset-latest.csv]  
Extracting sliding window data samples from [./data/group-dataset/held-with-manual-control/imu-latest-dataset.csv]...  
Finished extracting features to imu-and-features-dataset-latest.csv  
DONE
```

```
===[STEP 6 - DONE] ===
```

Like the training module, the raw IMU data for the entire flight session is traversed via sliding window protocol and the average and mean features are calculated from these windows of data. The user can also choose to run the optional cells that describe the process of extracting the feature from a random sample of IMU data. Now that the data is ready to be processed by the linear classifier, the next step of the lab produces a textual description of the predicted flight plan.

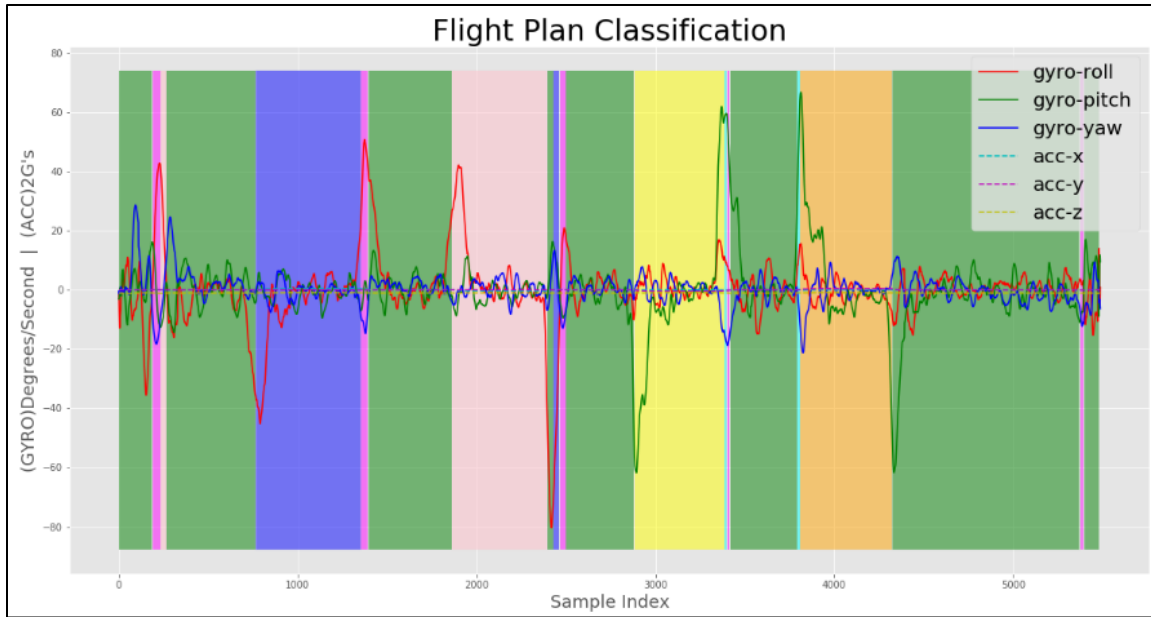


Figure 5.2: IMU Data Flight Plan Classification

This description is visualized in the final step of the lab, where shaders are applied on top of the raw IMU data graph to give a visual representation of the quadrotor's predicted activities during flight. It is expected that users record the flight plan for a given flight session before starting the classification module to assess that the flight plan produced by the classifier is correct. The successful execution of all cells means that a linear classifier has been used to generate a hypothetical flight plan for a given set of unlabeled IMU data. However, the accuracy and assessments made by the classifier depend on how well-trained the classifier was.

CHAPTER 6

RESULTS AND DISCUSSION

Both the training and classification lab modules were able to successfully process the HMC, HRC, and UN dataset as described by the previous section, experimentation and methodology. However, insufficient data was recorded for the UF dataset due to technical issues described in the limitations section. Despite these setbacks, all reporting from the lab sessions provided the real-time logs necessary to verify that the file manipulations were performed correctly.

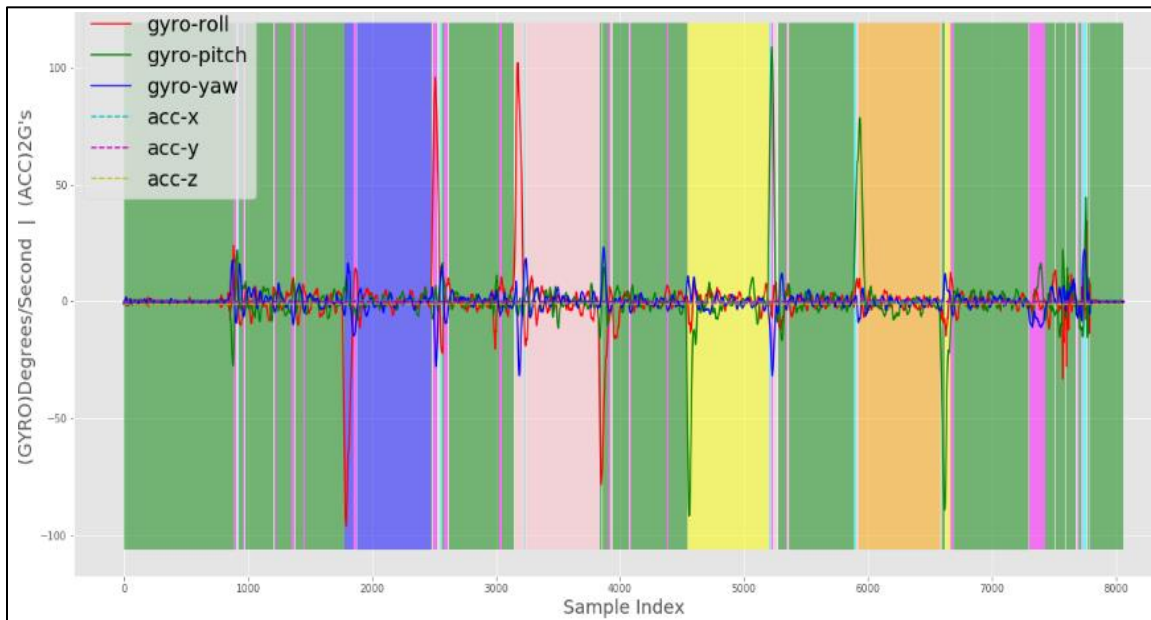


Figure 6.1.1: Simple Flight Plan Classification on HMC Dataset

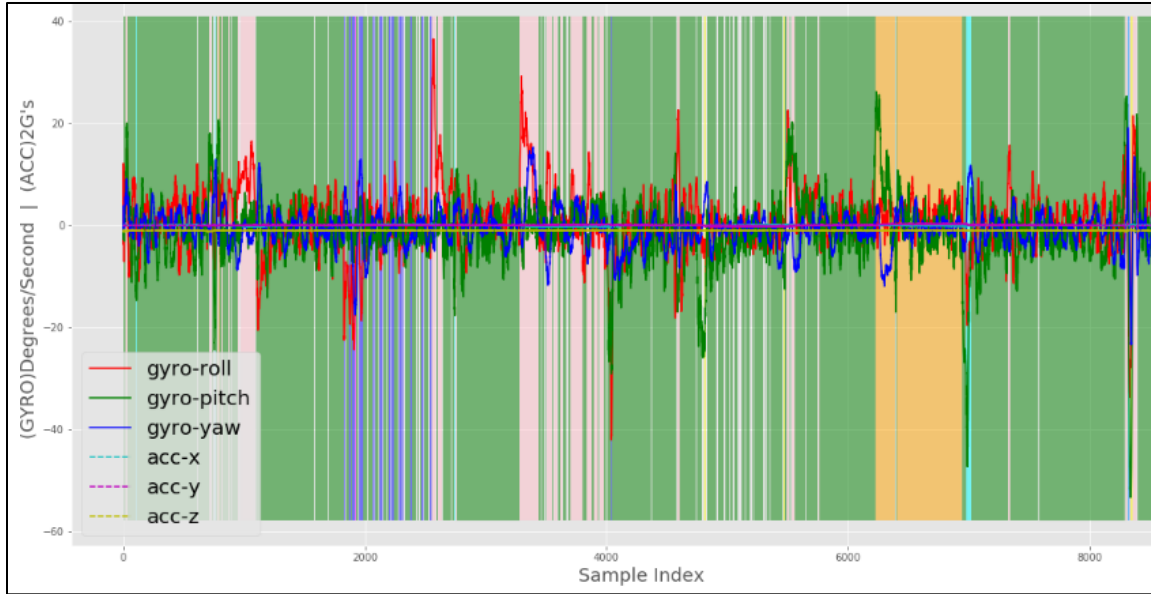


Figure 6.1.2: Simple Flight Plan Classification on HRC Dataset

Figures 6.1.1 and 6.1.2 provide a visual comparison of one specific flight plan described by the following sequence: {takeoff, neutral, left, neutral, right, neutral, forward, neutral, backward, neutral, land}. Of the two data sets, the HMC flight shows a more distinguished pattern, however some misclassifications from the HMC dataset could be attributed to human error of a slow reaction time. There are clear distinctions made between the 4 second transitions between state. However, the HRC flight plan is sparse with small windows of ‘neutral’ activity. The ‘forward’ classification for the HRC data and HRC-trained classifier is particularly weak in this instance.

Table 6.1: Accuracy and Loss over Epochs

Dataset	Accuracy over Epochs	Loss over Epochs
HMC		
HRC		
UF		

Detailed logs compiled into line graphs in table 5.1 indicate that with each epoch, the accuracy of the classifier increased. Full reference to the values is available in Appendix C. Within the results for the HMC dataset, a default configured linear classifier can predict class with an accuracy of approximately 82.2% percent. Logging reports

indicate that for every 10 epochs, it is evident that as the linear classifier adjusts its weights to minimize the loss function, the classifier's accuracy increases while the loss and average loss of the model decreases. The loss is calculated via SoftMax cross entropy by TensorFlow. The following figure shows the loss function and accuracy of the classifier against a given validation dataset over the number of epochs on which the classifier is trained on.

The classification lab module was able to use the classifier to deduce the flight plan of a given flight session. Figures 6.1.1 and 6.1.2 are examples where an unlabeled IMU dataset was shaded according to a predicted activity. A detailed description of the classification logs show there were significant windows of misclassifications made between the transition between states meaning that the linear classifier had a hard time classifying the activity transition point. As evident in the classification module, transitions between states describe a classification usually between neutral and a given activity.

One contributing factor to the inaccuracy of the classifier was the lack of distinguished signals between the increase altitude, neutral, and decreased altitude action. These actions share a similar signature due to the quadrotor's inability to determine its altitude. Implementing a sensor onboard the quadrotor to detect the altitude would have made for a better altitude holding algorithm since controlling the baseline throttle for all rotors proved to be too unresponsive. The result was that the quadrotor had a delayed and unresponsive reaction when changing altitude, making it difficult to tag windows of IMU data where the quadrotor was actively decreasing in altitude.

CHAPTER 7

LIMITATIONS

The quadrotor prototype presented in this thesis had several limitations that would have provided the MAR database with more useful data. One such limitation was the lack of an altitude sensor. Inclusion of an altitude device such as an ultrasonic range detector would have helped in maintaining a more stable altitude. While adjusting the baseline throttle for all four rotors was enough to change the altitude of the quadrotor, the altitude adjustment was responsively slow and could result in a fast drop in altitude enough to do a crash landing.

The lack of a Real Time Operating System (RTOS) also contributed towards the drifting problem. It was common that during a flight, the quadrotor would jitter due to system interrupts that would briefly halt the flight controller, causing the quadrotor to lose balance for a short period of time.

The inability to measure the level of battery. As the quadrotor's battery voltage dropped, the rotational speed of the rotors also decreased despite receiving the same PWM signal. To compensate for this issue, a battery gain was added to the flight controller's configurations requiring that users change these configurations as the quadrotor battery becomes more drained. Implementing a voltage divider and monitoring the battery via on-board analog pins may be enough to provide the necessary battery gain configuration automatically.

The quadrotor also suffered from a series of drifting problems causing the quadrotor to lean towards a preferred pitch and roll. A significant time was put into the

optimization and configuration of the PID controller gains, and during a heavy crash, the quadrotor had to be recalibrated before being flown again.

CHAPTER 8

CONCLUSION AND FUTURE WORKS

This work has provided VIPLE with extensions in 3D simulation for maze navigation and implementing a VIPLE-compatible quadrotor platform. A framework was provided for teaching students how to train and apply linear classifiers for MAR. Without additional hardware or software, a user is able to successfully train and apply a linear classifier through a web-enabled device to perform MAR on flight sessions recorded by the VIPLE and quadrotor platform.

The primary contributions of this thesis were extending VIPLE to the quadrotor platform, implementing an interactive machine learning pipeline for the MAR classification problem, and creating a repository of IMU data within MAR's database made available for future use. With an approximate even distribution of quadrotor activity, and by collecting the data through a manual control and manipulation of the quadrotor, the linear classifier was able to correctly predict a multi-rotor's activity with a window of IMU data with approximately 82% accuracy.

Future iterations of the MAR lab sessions can be applied to other robotic activity recognition. While quadrotors remain the focus, the pipeline described by this paper can be applied to other IMU sensory applications for use in activity recognition. Further iterations can also investigate automation and simplification of the current MAR labs for use in exploratory data analysis by generating automated reports for any given flight session.

Future works should consider using pre-built quadrotors rather than building one from scratch. At the time of writing, manufacturers such as DJI have recently provided

open source API libraries for consumer-ready quadrotors. Though the quadrotor in this thesis was able to sustain brief periods of flight time and perform basic activities, there were many issues encountered with implementing a quadrotor on the Edison platform.

While multiple instances of the VM described by the implementation section can be instantiated to host additional lab sessions, future iterations of MAR labs may want to consider encapsulating the Jupyter Notebook server into a Docker container to allow a single VM Instance to host multiple MAR lab instances, allowing the creation and host of additional MAR labs at a lower cost.

REFERENCES

- [1] L Deng. “The MNIST Database of Handwritten Digit Images for Machine Learning Research”. *IEEE Signal Processing Magazine*, vol 29, issue 6, Nov 2012.
- [2] Amber Dryer, et al. “A Middle-School Module for Introducing Data-Mining, Big-Data Ethics and Privacy Using RapidMiner and a Hollywood Theme”. *SIGCSE '18 Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, February 2018.
- [3] Gennaro De Luca and Yinong Chen. "Visual IoT/Robotics Programming Language in p-Calculus". *The 13th International Symposium on Autonomous Decentralized Systems*, Thailand, March 2017.
- [4] Yinong Chen and Hualiang Hu, "Internet of Intelligent Things and Robot as a Service", *Simulation Modelling Practice and Theory*, Volume 34, May 2013, Pages 159–171.
- [5] Yinong Chen. “Service-Oriented Computing and System Integration: Software, IoT, Big Data, and AI as Services”, 6th edition, Kendall Hunt Publishing, 2018.
- [6] Yinong Chen. Analyzing and visual programming internet of things and autonomous decentralized systems", *Simulation Modelling Practice and Theory* Volume 65, June 2016, pp. 1-10.
- [7] Teppo Luukkonen. “Modeling and Control of Quadcopter.” *Aalto University School of Science*, August 2011, http://sal.aalto.fi/publications/pdf-files/eluu11_public.pdf.
- [8] Vishakh Duggal, et al. “Plantation Monitoring and Yield Estimation using autonomous Quadcopter for Precision Agriculture.” *IEE International Conference on Robotics and Automation*, May 2016, <http://ieeexplore.ieee.org/document/7487716/>.
- [9] Michael Streber, et al. “Video-Based Estimation of Surface Currents Using a LowCost Quadcopter.” *3rd International Conference on Biosignals, Images, and Instrumentation*, March 2017, Chennai. <https://ieeexplore.ieee.org/document/document/8049342/>.
- [10] Josephin Dhivya and J Premkumar. “Quadcopter based technology for an Emergency Healthcare.” *3rd International Conference on Biosignals, Images,*

and Instrumentation, March 2017, Chennai.
<http://ieeexplore.ieee.org/document/8082284/>.

- [11] M Ranjbaran and K Khorasani. “Fault recovery of an under-actuated quadrotor Aerial Vehicle”. *49th IEEE Conference on Decision and Control (CDC)*, February 2011, <https://ieeexplore.ieee.org/document/5718140>.
- [12] Mark Mueller and Raffaello D’Andrea. “Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers. *2014 IEE International Conference on Robotics & Automation (ICRA)*, June 2014, <https://ieeexplore.ieee.org/document/6906588>.
- [13] Babajide Salau and Rajab Chaloo. “Multi-obstacle avoidance for UAVs in indoor applications”. *2015 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, December 2015, <https://ieeexplore.ieee.org/document/7475386>.
- [14] Ling Bao and Stephen S. Intille. “Activity Recognition from User-Annotated Acceleration Data.” *Massachusetts Institute of Technology*, 2005, <https://ieeexplore.ieee.org/document/5204354>
- [15] Murat Sorkun, et al. “Human activity recognition with mobile phone sensors: Impact of sensors and window size”. *2918 26th Signal Processing and Communications Applications Conference (SIU)*. July 2018. <https://ieeexplore.ieee.org/document/8404569>.
- [16] Spriggs, Ekaterina H.; de la Torre, Fernando; and Herbert, Martial, “Temporal Segmentation and Activity Classification from First-person Sensing.” *209 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. June 2009. <https://ieeexplore.ieee.org/document/5204354>.
- [17] Winters, Michael, et al., “Knowledge Engineering for Unsupervised Canine Posture Detection from IMU Data.” Robotics Institute. Paper 324. <http://repository.cmu.edu/robotics/324>.

APPENDIX A
QUADROTOR HARDWARE REVISIONS

In terms of hardware specifications, the VIPLE Quadrotor was built for recording and sending sensory data in real-time over TCP/IP via wireless IEEE 802.11a/b/g/n. The platform is equipped with a six-axis Inertial Movement Unit (IMU) sensor, an Intel Edison-Arduino breakout board, a 4s LI-PO Battery, a power distribution board equipped with a 9V voltage regulator, 4 brushless motors, and 4 electronic speed controllers (ESC). All components make the platform capable of stable flight for a quadrotor system. All on-board sensor must fit well within the parameters of a 250Hz correction rate calculated by the PID Controller. This hardware requirement is met by both on-board IC's (PCA9685 and MPU6050) which are both capable at operating at speeds faster than the quadrotor's correction rate.

To accommodate these two IC's a custom board was built to utilize the GPIO pins available on the Edison-Arduino Board. Figure A.1 is a closeup of the custom board with each component annotated: (A) MPU6050, (B) PCA9685, (C) Intel Edison Chip, (D) Status LED, (E) u.fl antenna connector. Figure A.2 provides a hardware schematic that better describes the electric connections made between all components on the custom shield.

The flight controller software is implemented on a C++ runtime and developed and compiled through Intel System Studio IoT Edition. The codebase for the entire flight controller is available at: <https://github.com/mldelaro/edison-pid-flight-controller>. The quadrotor's VIPLE interface runs on a separate process also implemented in C++ and made available in a separate repository at: <https://github.com/mldelaro/quadcopter-controller-tcp-runtime>. The VIPLE program used in controlling the quadrotor are also available at: <https://github.com/mldelaro/multirotor-activity-recognition-viple>.

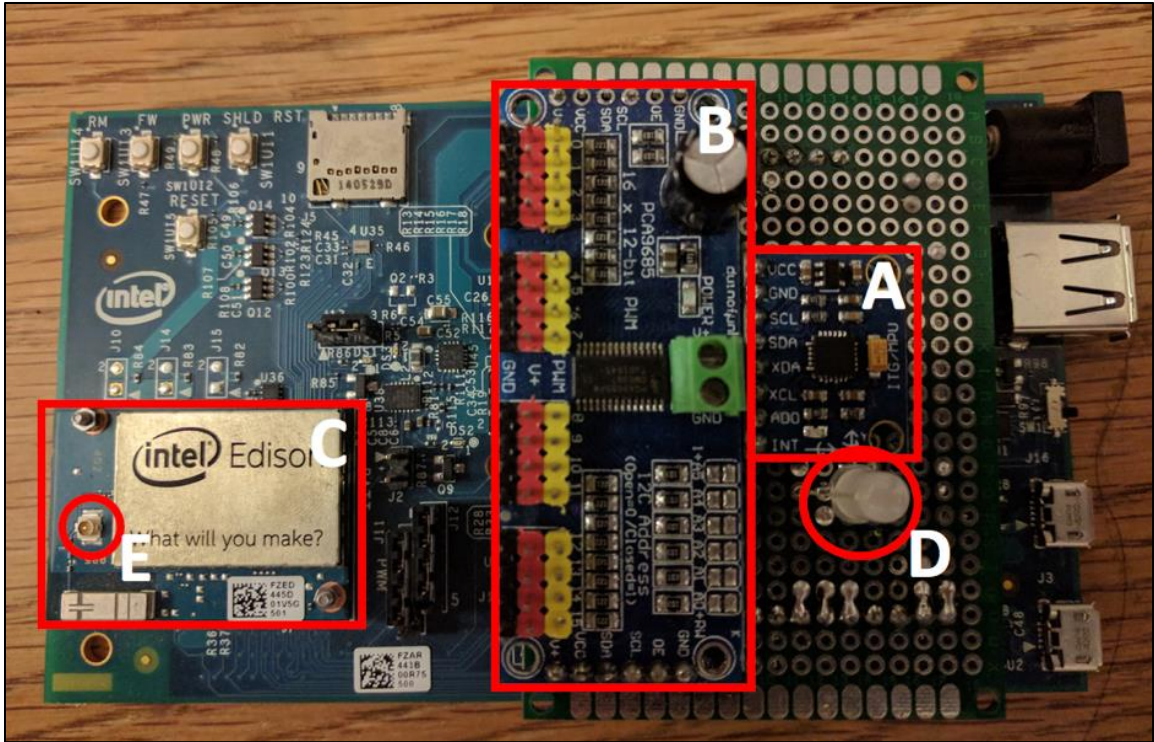


Figure A.1: Annotated Components of Flight Controller Shield

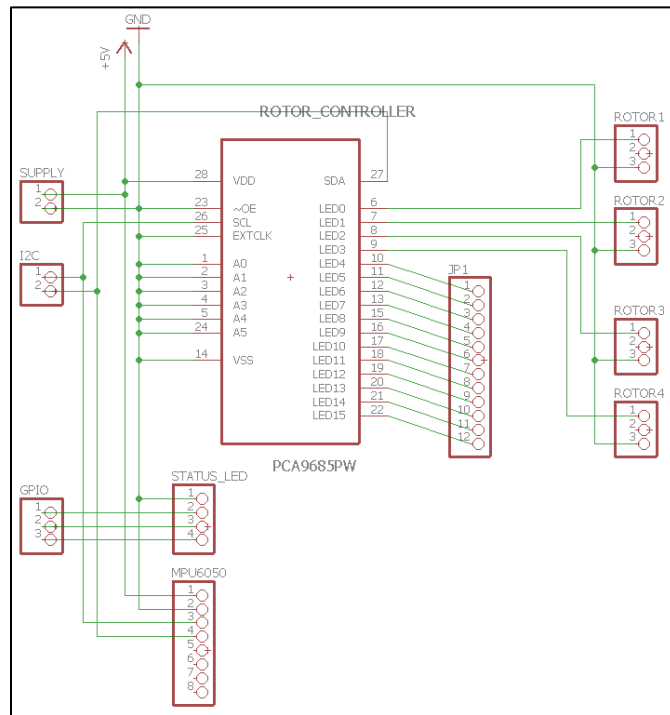


Figure A.2: Hardware Diagram of Flight Controller Shield

Before beginning any lab session, the quadrotor is powered on and mounted on a testing rack seen in figure A.3. The gain for the PID Controller is then adjusted to achieve a normal flight. The process begins by holding the quadrotor and tuning the P-controller to ensure that correctional forces are being applied properly when the quadrotor is tilted off-balance. This ensures that the difference between the error and setpoint of the system are configured correctly according to their respective roll, pitch, and yaw axis. Once the P-controller begins to over-compensate, and the quadrotor begins to oscillate, the gain is reduced, and the gain for the D-controller is increased. This process is then repeated for the I-Controller until the quadrotor can achieve a stable hover.



Figure A.3: Quadrotor PID Tuning Test Rig

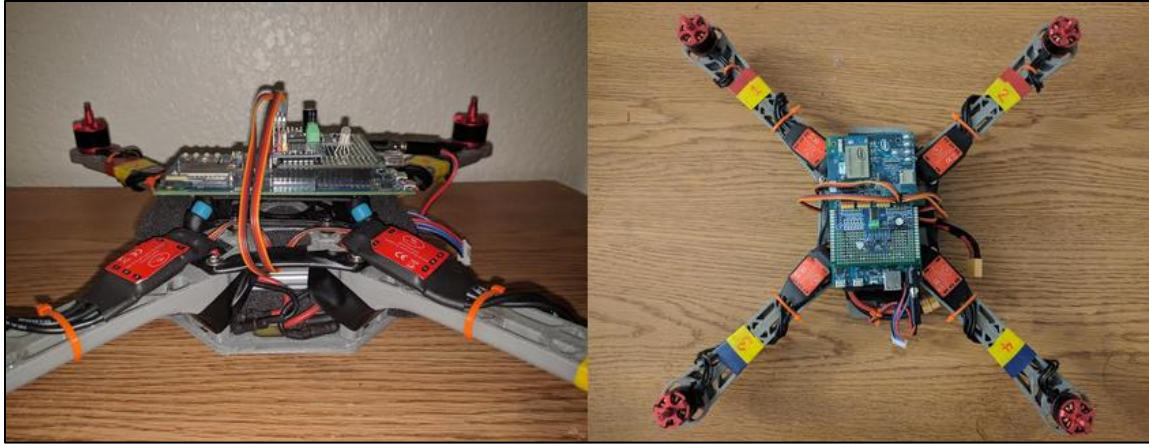


Figure A.4: Quadrotor Assembly Revision 1

The first, flyable, quadrotor prototype shown in figure A.4 was built with a quadrotor frame made from a 3D-printed standard Q450 frame (450mm x 450mm). The frame is capable of housing the necessary components and provides a decent platform for maintaining flight stability. However, after several initial test-flights, 3D printed ABS frames proved to be too vulnerable to hard landings and collisions. The extent of damage left from collisions is shown in the figure A.5 below.



Figure A.5: Fractures in 3D Printed Frames

Revision 2

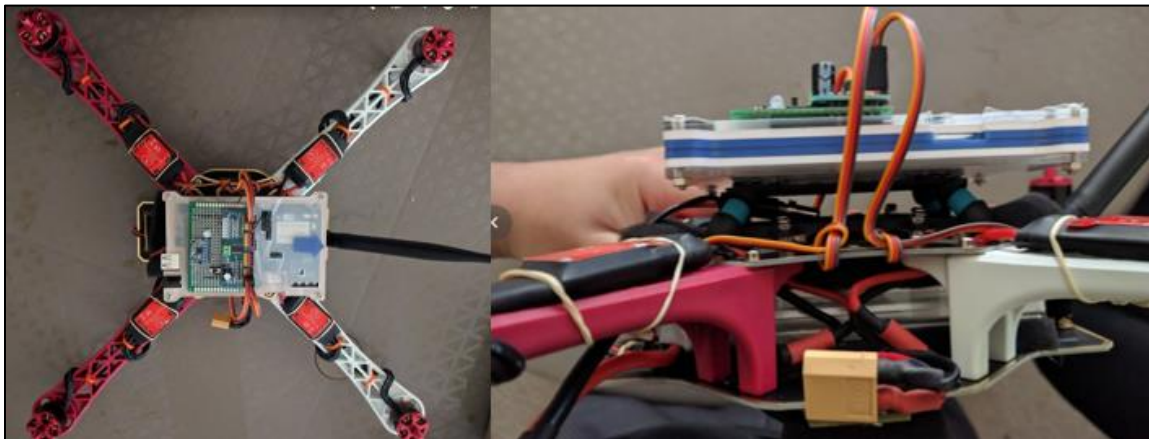


Figure A.6: Quadrotor Assembly Revision 2

The 3D printed quadrotor frame was replaced with a stiff plastic frame capable of withstanding the impact of failed flights. In addition, thread-lock was included to prevent screws from falling out of place. The quadrotor was then fitted with plastic landing gears.

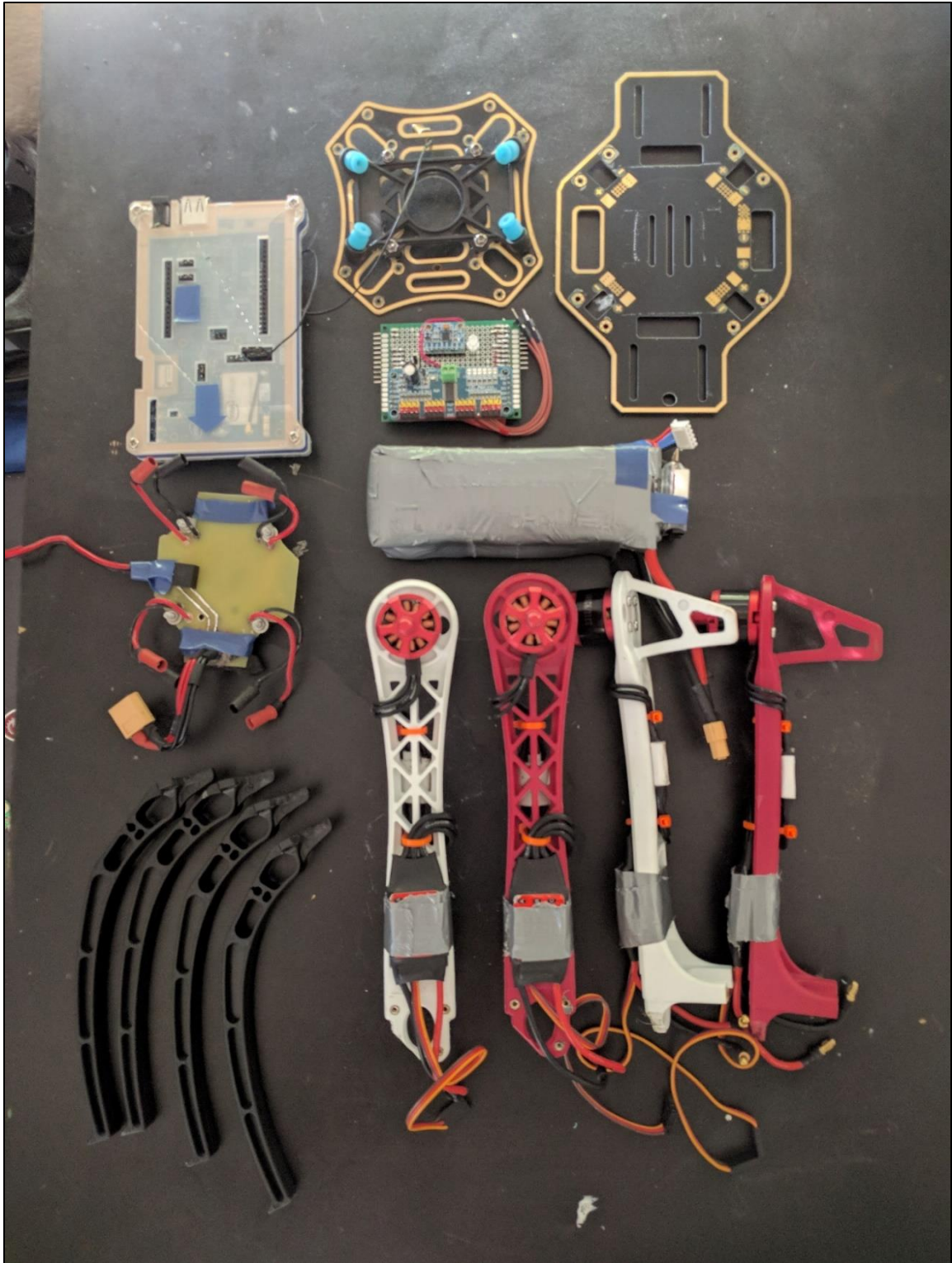


Figure A.7: Quadrotor Revision 2 Hardware Parts Nulling

APPENDIX B

LOSS AND ACCURACY OVER EPOCHS

Table B.1: Loss and Accuracy over Epochs Dataset

<i>Dataset</i>	HMC		HRC		UF	
<i>epoch</i>	loss	accuracy	loss	accuracy	loss	accuracy
0	364.9385	0.276532	1192.737	0.086027	517.7102	0.291168
1	287.2623	0.47678	774.1066	0.151633	439.7259	0.336742
2	244.9039	0.539255	636.3419	0.210591	347.1531	0.493524
3	218.7955	0.609522	545.2913	0.254849	368.2013	0.421657
4	203.4566	0.648242	485.6321	0.290017	362.5511	0.426429
5	191.1816	0.676408	439.577	0.324731	349.4738	0.479501
6	183.1928	0.69178	404.5905	0.358174	340.7291	0.472782
7	176.6142	0.701784	376.4477	0.391771	330.3822	0.484468
8	170.1751	0.711097	352.4684	0.422799	343.706	0.480475
9	165.9411	0.718114	333.8918	0.451938	342.7357	0.48291
10	161.9469	0.7243	318.3945	0.477299	331.6588	0.507644
11	158.6017	0.730964	304.1555	0.501925	322.7	0.516506
12	155.9568	0.737305	292.9115	0.522655	316.3803	0.529944
13	153.1771	0.744265	281.8796	0.541461	320.9269	0.528971
14	150.5465	0.749648	272.0388	0.558904	305.8332	0.536469
15	148.829	0.753945	264.0367	0.573469	304.0445	0.536372
16	146.62	0.758539	256.198	0.587616	309.7166	0.532866
17	145.5918	0.7613	249.5483	0.599466	303.4714	0.539877
18	143.424	0.765612	243.3117	0.610971	307.745	0.537248
19	142.0269	0.768613	237.475	0.620478	300.9675	0.540364
20	140.9478	0.770881	232.3645	0.629113	297.1738	0.544065
21	139.4354	0.773756	226.8478	0.637876	299.099	0.542799
22	138.5525	0.775559	222.3137	0.645703	297.6997	0.543286
23	137.4252	0.778053	218.0389	0.653004	293.4988	0.547668
24	136.2249	0.780167	214.0128	0.659959	293.7294	0.546986
25	135.1186	0.782083	210.3694	0.666661	292.4838	0.547083
26	134.6359	0.783098	206.5446	0.672408	290.1852	0.549908
27	133.8695	0.784436	203.379	0.678038	284.6116	0.551952
28	132.7255	0.786395	200.1933	0.683032	282.6082	0.554776
29	131.953	0.787973	197.2342	0.687291	286.0105	0.550492
30	130.9581	0.789875	194.3637	0.692594	284.2039	0.551173
31	130.4243	0.79072	191.7921	0.695936	278.8916	0.556042
32	129.6099	0.792369	189.5125	0.699822	277.9456	0.555361
33	128.931	0.79344	186.8407	0.704172	276.6161	0.556042
34	128.4373	0.794623	184.3958	0.70774	276.7295	0.55799
35	127.5732	0.796399	182.1754	0.711272	274.8663	0.557893

36	127.2415	0.796906	180.2734	0.714723	270.4576	0.559061
37	126.5637	0.797963	178.4303	0.71801	272.7347	0.559646
38	126.093	0.798836	176.3282	0.721697	272.6694	0.561204
39	125.5008	0.799738	174.7827	0.724321	271.745	0.561398
40	125.1229	0.800457	173.0065	0.727653	269.1653	0.561788
41	124.3918	0.801725	171.3013	0.730359	266.808	0.563833
42	124.2795	0.80195	169.8696	0.732711	265.7103	0.565294
43	123.695	0.80288	168.1864	0.736044	266.97	0.56393
44	123.3904	0.803697	166.8889	0.737969	267.684	0.565196
45	122.7073	0.804543	165.4178	0.740674	264.959	0.563833
46	122.5646	0.804951	164.1958	0.74309	263.5882	0.565488
47	122.158	0.805444	163.0384	0.745696	264.0213	0.565586
48	121.7841	0.80574	161.8298	0.748375	261.9625	0.568118
49	121.2397	0.806403	160.53	0.750835	259.5758	0.568605
50	120.8862	0.807037	159.5785	0.753205	261.1698	0.567923
51	120.8056	0.807318	158.4444	0.755267	260.5055	0.567533
52	120.1867	0.808009	157.2881	0.757691	262.7921	0.565488
53	119.9791	0.808347	156.3318	0.75958	259.6645	0.569481
54	119.7269	0.808967	155.2782	0.76185	259.1859	0.569189
55	119.2935	0.809474	154.4776	0.763157	258.585	0.569189
56	119.2257	0.809869	153.4354	0.765237	257.1381	0.57026
57	118.8954	0.810644	152.5312	0.766889	257.1774	0.569676
58	118.6285	0.81063	151.8774	0.768179	256.0153	0.569578
59	118.3192	0.81139	150.9541	0.769986	254.1434	0.57065
60	118.1169	0.811714	150.2242	0.771629	258.7646	0.568702
61	117.9384	0.812095	149.4252	0.772683	254.626	0.571039
62	117.6632	0.812518	148.626	0.774045	254.4203	0.570844
63	117.4054	0.812884	147.8685	0.775534	252.3472	0.571429
64	117.3071	0.813236	147.2647	0.776333	252.8183	0.57211
65	116.8628	0.813842	146.5217	0.777677	252.7447	0.572402
66	116.69	0.814208	145.9685	0.778757	252.6659	0.57211
67	116.6644	0.814406	145.2615	0.780129	248.5106	0.575226
68	116.3428	0.814744	144.6574	0.781391	252.1533	0.572597
69	116.173	0.814983	144.068	0.782544	249.3992	0.573376
70	115.8761	0.815293	143.5349	0.784042	250.2021	0.57474
71	115.753	0.815646	142.8875	0.785114	247.4861	0.574545
72	115.5656	0.815815	142.3905	0.786476	249.8033	0.572987
73	115.3469	0.816096	141.762	0.788028	248.6755	0.574155
74	115.2896	0.816139	141.3265	0.788927	248.0538	0.575324
75	115.0369	0.816731	140.8499	0.789781	246.5068	0.57659

76	114.9013	0.816984		140.3249	0.790871		247.8456	0.575032
77	114.6746	0.817351		139.8895	0.791788		245.709	0.576492
78	114.453	0.817646		139.2858	0.793104		245.5974	0.575324
79	114.4835	0.817505		138.867	0.793686		246.2751	0.575519
80	114.2058	0.817815		138.4198	0.794857		244.9167	0.577369
81	114.0799	0.817942		137.946	0.795865		245.9108	0.574837
82	113.8991	0.818295		137.5583	0.796773		243.7553	0.576882
83	113.7537	0.818576		137.0833	0.797545		244.7948	0.576882
84	113.6025	0.818872		136.7641	0.798198		244.6373	0.576687
85	113.4908	0.819351		136.3073	0.798961		244.4905	0.577369
86	113.4383	0.819295		135.9383	0.799733		242.9488	0.577953
87	113.2587	0.819506		135.513	0.800886		243.0432	0.577953
88	113.0732	0.819859		135.2173	0.801368		242.8307	0.577466
89	112.9594	0.82007		134.7473	0.802375		243.2606	0.576687
90	112.9082	0.820253		134.3974	0.802938		242.623	0.578635
91	112.7112	0.820549		134.0045	0.803719		242.224	0.578537
92	112.6092	0.820831		133.7314	0.804128		240.9254	0.579414
93	112.4704	0.820943		133.3883	0.804809		240.5668	0.579609
94	112.3257	0.821169		133.1007	0.805617		239.4564	0.580388
95	112.2842	0.821239		132.8191	0.805989		240.5338	0.579901
96	112.1205	0.821521		132.4778	0.806798		239.2618	0.58068
97	112.0285	0.821718		132.1282	0.807778		239.7636	0.580875
98	111.9803	0.821733		131.8068	0.808486		239.6605	0.581264
99	111.8611	0.822014		131.5014	0.808931		238.3819	0.581069

APPENDIX C

MAR LAB ENVIRONMENT SETUP GUIDELINES

MAR Lab environments follow the same usage guidelines outlined by Jupyter Notebook (version 4.4.0) documentation. In general, a user begins interacting with the MAR Lab session by visiting the public IP address of the Jupyter Notebook Server instance. This is created and configured through AWS.

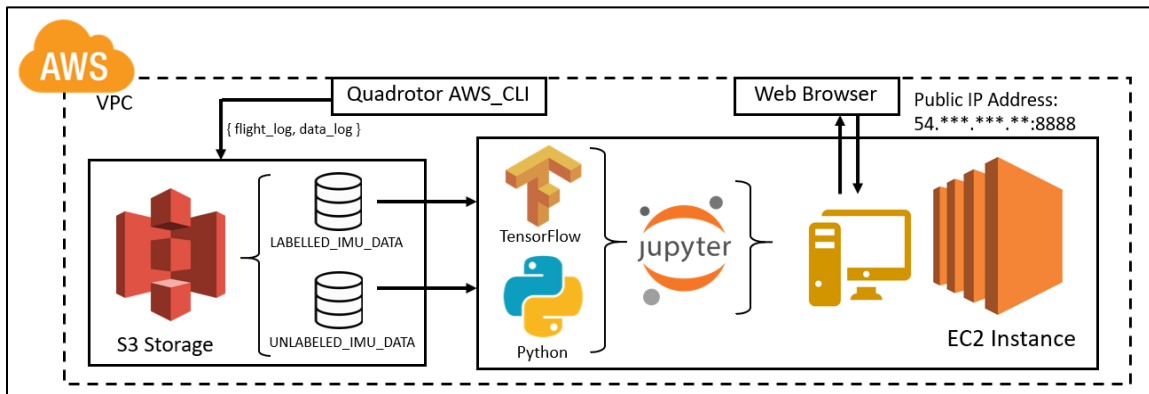


Figure C.1: Architecture Diagram of AWS MAR Lab Environment

AWS resources are created and configured through the online AWS Management Console as generally described by figure C.1. The purpose and usage of each component are further described in the Implementation section under the “MAR Lab AWS Environment” subsection.

Interaction with the lab environment begins with the installation of the AWS CLI on the quadrotor. Remoting into the intel Edison board and following the installation guidelines for the AWS CLI will provide the Edison board with the ability to upload files from the board. An AWS Identity Access Manager (IAM) user needs to be created on the quadrotor’s behalf. As shown in figure C.2, the quadrotor user is then granted read and write access to the S3 bucket that hosts the flight data from the quadrotor sessions.

Users > intel-edison-quadcopter

Summary Delete user ⓘ

User ARN am:aws:iam::{{AWS_ACCOUNT_ID}}:user/intel-edison-quadcopter [🔗](#)

Path /

Creation time 2018-03-04 02:16 MST

Permissions **Groups (1)** **Security credentials** **Access Advisor**

▾ Permissions policies (1 policy applied)

[Add permissions](#) [+ Add inline policy](#)

Policy name ▾	Policy type ▾	
Attached directly		
▶ s3-full-access	Inline policy	✕
▶ Permissions boundary (not set)		

Figure C.2: Quadrotor IAM User

As shown in figure C.3, the MAR Lab environment uses a single S3 Bucket for hosting all data collected from the Quadrotor-VIPLE platform. Versioning is enabled, and the bucket is configured to be publicly available with read-only access.

Amazon S3 Discover the new console Quick tips

Search for buckets

[+ Create bucket](#) [Delete bucket](#) [Empty bucket](#) 1 Buckets 1 **Public** 1 Regions [↻](#)

Bucket name ↑	Access ⓘ ↑	Region ↑	Date created ↑
midelarosa-thesis	Public	US East (N. Virginia)	Jan 29, 2018 12:27:52 AM GMT-0700

Figure C.3: MAR Lab S3 Bucket

An EC2 instance is host to the remaining software components encompassed in the EC2 block described by figure C.1. To provide a public-facing IP address, an Elastic IP resource is created and later used in the EC2 instance setup.

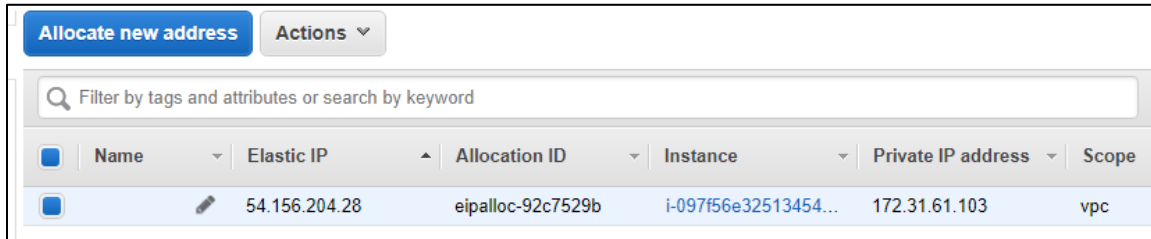


Figure C.4: Public Elastic IP Address

The VM is setup as a t2.medium instance and configured with the elastic IP address shown in figure C.5. The setup process begins by shelling into the Linux VM instance and following the installation documentation outlined for Python (version 3.6.4) and TensorFlow (version 1.5.0).

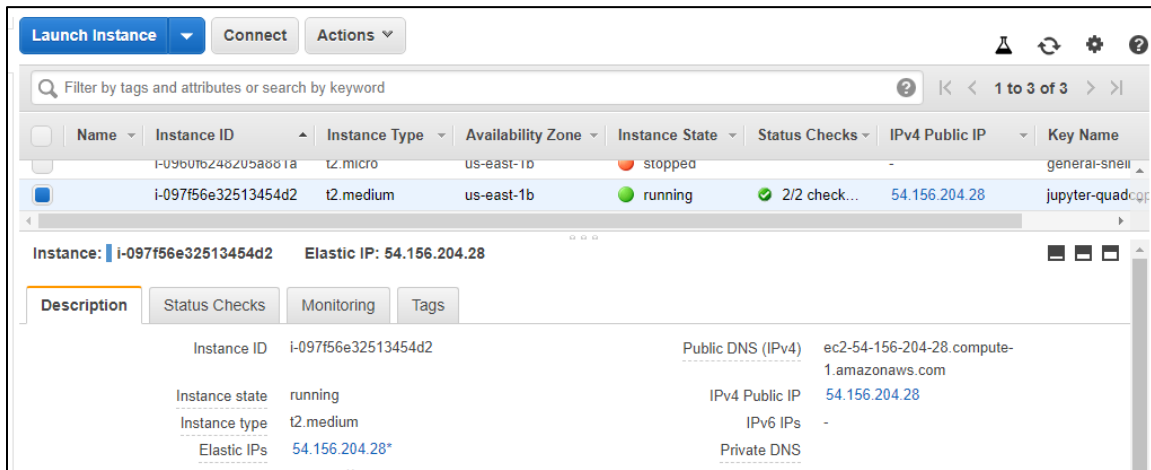


Figure C.5: MAR Lab EC2 Instance Details

Python and TensorFlow provide the necessary runtime for installing the Jupyter Notebook environment. Following the documentation and guidelines for installing both the Jupyter Notebook and the Public Jupyter Notebook Server will install the software necessary for hosting the MAR Lab sessions. Finally, the MAR Lab sessions can be imported into the Jupyter instance's home directory. The content of the MAR Lab sessions for both the training module and the classification module are available at <https://github.com/mldelaro/multirotor-activity-recognition>.