2019

An Approach to Semi-Autonomous Indoor Drone System: Software Architecture and Integration Testing

Shobhan Singh

University of Mississippi

An Approach to Semi-Autonomous Indoor Drone System: Software Architecture and Integration Testing

A Thesis
presented in partial fulfillment of requirements
for the degree of Master of Science
in the The Department of Computer and Information Science
The University of Mississippi

by

Shobhan Singh

May 2019

ABSTRACT

Autonomous drones are gaining traction among hobbyists, photographers and videographers, and in other commercial applications. With current trends, indoor drones have huge potential and there is an increase in commercial demand, as well as interest in developer community, but such systems face challenges on achieving high localization and stability as the sensor data from barometer and GPS becomes unreliable. These drones need a Radio Controller (RC) transmitter to drive them and to provide autonomous and customized control functions to the Drone System. Developers use software and libraries on a Companion Computer, but suffer from integration challenges, especially if there are changes within atomic operations of the drone while adding new or removing existing sensors.

To address these problems, we establish a semi-autonomous functionality by removing the RC transmitter, and remotely connecting the Drone System to track status and executing user-based input commands. In order to resolve the limitation in hardware connections on the Flight Controller, we integrated the sonar sensor into a companion computer, from where the data is continuously fed to an embedded system through MAVLink (Micro Aerial Vehicle Link) network communication protocol. In this study, we also implemented a modular architecture which enables scalable integration of sensor modules into the Drone System to streamline the process of development, deployment, testing and debugging.

DEDICATION

To the butterfly who fluttered its wing and changed Earth's weather patterns.

aaj ki chai mein chaipatti tair rahi hain.

(Today, there is a tea leaf floating in my cup of tea.)

shayad chini bhi kam hogi.

(There might be less sugar in the tea as well)

aaj kaam shayad jyada hoga.

(Today, he/she might have a little more work than usual)

koi nahi, ungli mein chipka kar nikal leta huin.

(No worries, I'll stick it on my finger and get it out)

janab, aapne bhi kya chai banai.

(O dear, what a tea have you made!)

koi nahi, patile mein daal kar phir ubal leta huin.

(No worries, I'll pour in the pan and boil it again.)

ACKNOWLEDGEMENTS

I am highly thankful to Dr. Byunghyun Jang for his insightful advising and constant encouragement, not just for this research but throughout my graduate course. For this project, I found Dr. Jang's Professor-Manager style very comforting; it helped me to analyze every aspect of the project like a free 'git forking' bird - git stash when fail and git merge when success. His reiteration on 'Do Not Quit' is always an echo that stays in minds of his students. His comments and analysis of the problem forces to think the basics and swat fundamental bugs of the system.

The enthusiasm and help at Heroes lab are highly appreciated. I especially thank David, Mason and Shirish for their kind advice.

I am thankful to my mom and dad for helping (and not helping! :)) out whenever I asked. They had always been an inspiration. I thank my brother - *Chote Bhai Sahab* - for his constant encouragement and his insight on electrical concepts.

I thank my tribal friends — Abhijeet, John, Harshul, Manjeet, Michael, Naveen, Trisha, Sai, Shirish, Saqlain — for staying classy <wink emoji, folded hand emoji>. I thank Gallus gallus domesticus, Ovis aries, Capra aegagrus hircus, and Hordeum vulgare for being in my life. I thank my mates at cricket club for teaching team spirit, calmness, agression and the fightback attitude. As they say at the other Oxford across the Atlantic, the battle of Waterloo was won on the playing fields of Eton. I now believe it. I am not thankful to Olemiss football team who did not play well this season :). #ForeeverRebels

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AFM Autonomous Flight Management

BEC/UBEC (Universal) Battery Eliminator Circuit

BSD Berkeley Software Distribution; an open-source license

COM Communication port

CLI Command-line Interface

CW/CCW Clockwise/Counter-Clockwise

ECHO Echo Pulse Output; in Circuit

ESC Electronic Speed Controller

GCS/GS Ground (Control) Station; this is a remote computer to probe and control the drone system.

GND Ground; in Circuit

GPIO General-Purpose Input/Output

GPL GNU General Public License

GPS Geo-Positional System

GUI Graphical User Interface

HEROES HEteROgEneous Systems research lab; at The University of Mississippi

I2C Inter-Integrated Circuit protocol

IDE Integrated Development Environment

IP/IPs Internet Protocol Address/Internet Protocol Addresses

LiDAR Light Detection and Ranging

LiPo Lithium Polymer battery

MAVLink Micro Air Vehicle Link; networking protocol for communicating with small unmanned vehicle

NASA National Aeronautical and Space Administration

PWM Pulse Width Modulation

PX4 3DR® Pixhawk® Mini Autopilot Module

OEM Original Equipment Manufacturer

OS Operating System

RC Radio Control

RPI3 Raspberry Pi 3 Module

SD Secure Digital; used in association with memory cards.

SITL Software In The Loop

SLAM Simultaneous Localization and Mapping

SMART Spaceship Mission Assessment and Re-planning Tool team; at NASA

SPI Serial Peripheral Interface

SSH Secure Shell

TELEM Telemetry Port in PX4; used for MAVLink communication and supports flow control.

TCP/IP Transmission Control Protocol/Internet Protocol

TRIG Trigger Pulse Input; in Circuit

UART Universal Asynchronous Receiver-Transmitter

UAV Unmanned Aerial Vehicle

USB Universal Serial Bus

VM Virtual Machine

Wi-Fi Wireless Fidelity; this is a local area wireless technology.

CHAPTER 1

INTRODUCTION

An Unmanned Aerial Vehicle (UAV), commonly known as a drone, is an aircraft without a human pilot aboard. UAVs are a component of an unmanned aircraft system (UAS); which also includes a ground-based controller and a system of communications between the two (Watts et al., 2012). The commercial successes of DJI, GoPro, 3D Robotics and other drone companies have made researchers push technological advances to achieve a higher degree of autonomy (Canis, 2015). Current commercial drone systems are automatically controlled but are manually driven, and their degree of autonomy lies below a score of 4 defined by Proud et al. (2003) from the NASA Spaceship Mission Assessment and Re-planning Tool (SMART) team. To push for a system with higher scale, the open software libraries available have limited support for integrating navigational and control algorithm. On a higher scale of autonomy, we should be able to integrate machine learning and artificial intelligence algorithms as well. To achieve a higher scaled UAS system we must work in the direction which enables not only automatic control but also automatic drive functionality.

There have been some commercial efforts in getting drones to fly indoors as well, using Pozyx, OptiTrack and other technologies (3D Robotics, 2019b). Outdoor drone systems face reliability problems related to localization when used indoors. Barometers are inconsistent indoors because of artificially modified condition, e.g., on temperature and pressure (Li et al., 2013). The GPS is also unreliable due to a signal loss indoors, even though it has centimeter level precision outdoors (Dedes and Dempster, 2005). Due to these reasons, the indoor drone system becomes highly unstable; it is best to remove the barometer and GPS sensors and look for alternatives. In our case, we trimmed down to sonar and LiDAR (Light Detection and Ranging) sensors as alternatives to barometer and GPS, respectively, as proposed in previous studies (Gossett, 2018), for enabling indoor localization.

As our study progressed, we faced problems related to sensor integration with the existing drone system. Therefore, we revisited our approach and switched to one that is loosely coupled and scalable (Parnas, 1972). We observed that the existing open source software libraries were not modular enough when interacting with each other. The concepts of companion computer open source softwares are available; these enable communication between the companion computer and the flight controller, e.g., Dronekit-Python. It enables navigational components (e.g. goto/waypoint commands), GUI implementations, mission management and direct control over vehicle movement and operations (3D Robotics, 2019a). On a larger note, Dronekit-Python wraps ArduPilot embedded C++ methods as Python methods. These Python methods are atomic and immutable in nature and act like a basic control unit for a drone. Each of these Python methods send pre-defined standard MAVLink messages to the Flight Controller, e.g., *set_roi(location)* python method creates and sends *MAV_CMD_DO_SET_ROI* MAVLink messages which point the camera gimbal at a specified *location*. If there are no supporting messages in MAVLink communication protocol, we cannot add new operations or fine tune existing operations with Dronekit-Python. Although, developers tend to do this convenient Python implementation, we opted for a more permanent solution by updating the embedded C++ based firmware code on the Flight Controller.

On a similar note, we found the library is not scalable when new sensors are added. We are not able to do integration if there are no standard messages in MAVLink communication protocol for such newer sensors. To resolve this, we used a customized MAVLink message and used a Ground Control Software (GCS) to send the message to the firmware on the Flight Controller. Also, there could be a need for subtle changes within the atomic functionality of the drone, e.g., reading sonar sensor data and then disarming/checking for obstacles, i.e., forking within the wrapped embedded methods. In that case, we need to look back into the embedded flight controller source code. It is to note that the firmware itself have limitation on hardware connections and using this sensor integration technique provides us a workaround for a scalable model. Also, to note that we are looking at a more challenging implementation of a software patch for the embedded C++ firmware code that enables such new operations/functionality in the drone.

We also found that current sending and receiving of sensor data between companion computer and flight controller is not the most efficient way of communicating. Instead, collecting the data at the companion computer and then sending it as MAVLink commands saves us from this two-way communication. This unidirectional communication also halves the amount of processing we need to send and receive data.

The goal of this thesis is to expand on existing drone systems at the Heterogeneous Systems research (HEROES) lab and improve on the autonomy ability of drone. The major contributions of this thesis are:

- Removal of the RC Transmitter and arming the motors so that the *Autonomous Driven - Autonomous Control* functionality could be achieved in the future.
- Implementation of a modular architecture for development, deployment and testing of the drone system. This system follows a unidirectional flow of data which makes it more efficient.
- Instructions to set up the software stack and the proof of concept using integration testing over the two interfaces: ground station to companion computer interface and companion computer to flight controller interface.
- Integrating the sonar sensor attached to the companion computer, to the flight controller by using MAVLink network protocol messages.

CHAPTER 2

BACKGROUND

The initial drone research at the HEROES Lab converges on guiding the drone using a RC Controller. The software stack in the preceding studies (Gossett, 2018) was implemented using Python development on an on-board Raspberry Pi 3 (RPI3). The Pulse Width Modulation (PWM) value to the rotors was set automatically by an on-board autopilot flight control unit using the DroneKit-Python library on the RPI3. Dronekit-Python uses MAVLink messages to communicate to the flight control unit. Figure 2.1 describes the data flow for the preceding Drone System.

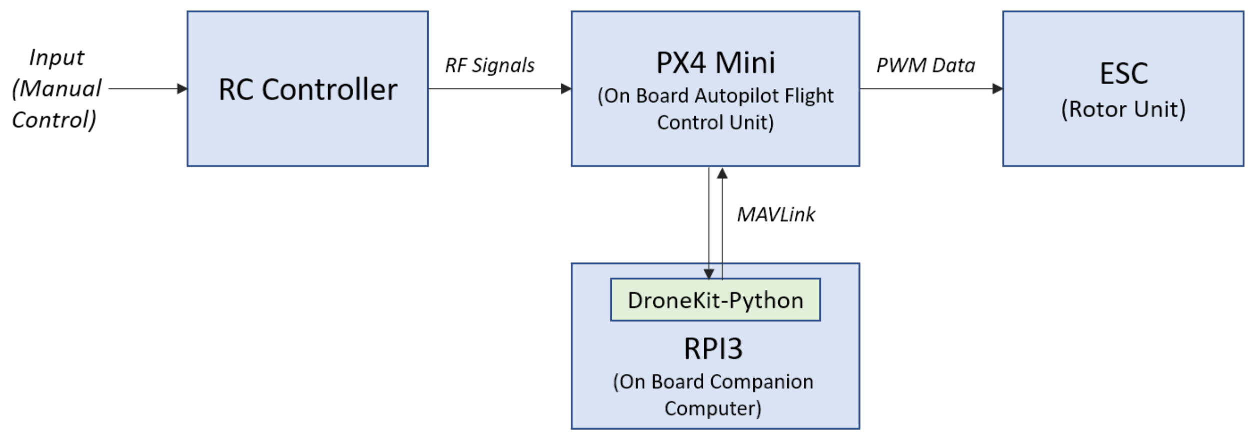

Figure 2.1. Data flow in initial design.

The software development includes using RC overrides and feedback loops to control the values of RC Channels. Autonomous operations such as taking off, holding altitude and landing used throttle channel (Channel 3), which would run a feedback loop for the throttle value until the target altitude is reached, or the drone would hold the target altitude, or the throttle channel value would purge to zero, respectively. A similar approach is taken to control attitude where corresponding channels, viz., channel 1 for roll, channel 2 for pitch and channel 4 for yaw, are used in a feedback loop. (Gossett, 2018)

Although the Python libraries are easy to pick up and are well documented, the preceding study faced a problem regarding the stability of the drone. The drone stays uncontrollable for an initial period and then may flip over due to the feedback delays. There are also concerns, which are discussed earlier in the Introduction section, for delay in to-and-from data communication, adding logic within the fundamental control methods/unit such as takeoff() and landing(), and disabling sensor libraries.

The ArduPilot firmware has updated its source code to include ESC re-calibrations. This solves the problem of needing to calibrate the ESC before missions. This code is by default executed every time when the flight control unit is powered up. We also changed the time it executes by changing the source code.

As stated earlier in the Introduction section, NASA through its SMART team (Proud et al., 2003) has defined a scale for degree of autonomy to answer, “How autonomous should an AFM (Autonomous Flight Management) System be?” For the autonomous drone study at HEROES Lab, the objective is to achieve a score of 4 or higher, where the computer takes prime responsibility in making decision and taking action.

Level	Observe	Orient	Decide	Act
8	The computer gathers, filters, and prioritizes data without displaying any information to the human.	The computer predicts, interprets, and integrates data into a result which is not displayed to the human.	The computer performs ranking tasks. The computer performs final ranking, but does not display results to the human.	Computer executes automatically and does not allow any human interaction.
7	The computer gathers, filters, and prioritizes data without displaying any information to the human. Though, a "program functioning" flag is displayed.	The computer anlayzes, predicts, interprets, and integrates data into a result which is only displayed to the human if result fits programmed context (context dependant summaries).	The computer performs ranking tasks. The computer performs final ranking and displays a reduced set of ranked options without displaying "why" decisions were made to the human.	Computer executes automatically and only informs the human if required by context. It allows for override ability after execution. Human is shadow for contingencies.
6	The computer gathers, filters, and prioritizes information displayed to the human.	The computer overlays predictions with analysis and interprets the data. The human is shown all results.	The computer performs ranking tasks and displays a reduced set of ranked options while displaying "why" decisions were made to the human.	Computer executes automatically, informs the human, and allows for override ability after execution. Human is shadow for contingencies.
5	The computer is responsible for gathering the information for the human, but it only displays non-prioritized, filtered information.	The computer overlays predictions with analysis and interprets the data. The human shadows the interpretation for contingencies.	The computer performs ranking tasks. All results, including "why" decisions were made, are displayed to the human.	Computer allows the human a context-dependant restricted time to veto before execution. Human shadows for contingencies.
4	The computer is responsible for gathering the information for the human and for displaying all information, but it highlights the non-prioritized, relevant information for the user.	The computer analyzes the data and makes predictions, though the human is responsible for interpretation of the data.	Both human and computer perform ranking tasks, the results from the computer are considered prime.	Computer allows the human a pre-programmed restricted time to veto before execution. Human shadows for contingencies.
3	The computer is responsible for gathering and displaying unfiltered, unprioritized information for the human. The human still is the prime monitor for all information.	Computer is the prime source of analysis and predictions, with human shadow for contingencies. The human is responsible for interpretation of the data.	Both human and computer perform ranking tasks, the results from the human are considered prime.	Computer executes decision after human approval. Human shadows for contingencies.
2	Human is the prime source for gathering and monitoring all data, with computer shadow for emergencies.	Human is the prime source of analysis and predictions, with computer shadow for contingencies. The human is responsible for interpretation of the data.	The human performs all ranking tasks, but the computer can be used as a tool for assistance.	Human is the prime source of execution, with computer shadow for contingencies.
1	Human is the only source for gathering and monitoring (defined as filtering, prioritizing and understanding) all data.	Human is responsible for analyzing all data, making predictions, and interpretation of the data.	The computer does not assist in or perform ranking tasks. Human must do it all.	Human alone can execute decision.

Figure 2.2. Level of autonomy assessment scale (Proud et al., 2003).

CHAPTER 3

OBJECTIVE

In order to communicate with the Drone System, we considered a new headless approach where input would be transmitted from a remote PC to the on-board Computer Unit (RPI3). The RPI3 will then communicate to the Flight Controller Unit (PX4) using MAVLink messages to drive the rotors. Figure 3.1 describes the data flow for the proposed Drone System.

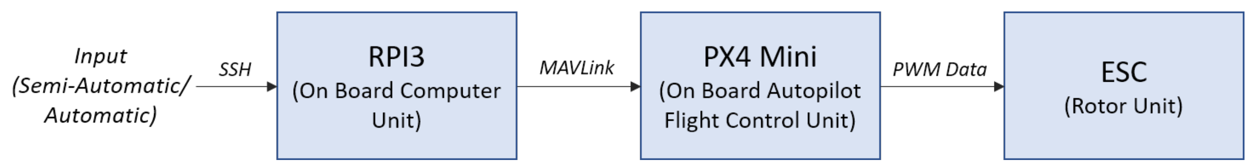

Figure 3.1. Data Flow in Proposed Design.

The design supports data flow that is unidirectional in nature. Although RPI3 communicates on MAVLink messages, this is a user customized message unlike the one provided/leveraged by the Companion computer development software. There are no changes to the flight controller generating PWM data but there are flags that we need to set in order to enable this communication. To summarize, this approach has considerable advantage:

1. Unidirectional data flow saves a considerable amount of time by using one-way communication.

2. Inputs could be communicated programmatically to the Drone System instead of being controlled using the RC Transmitter.

3. Existing Home Wi-Fi system could be leveraged to communicate between the remote computer and the Drone System.

4. Integration of testing and debugging mechanism to the system.

Overall, the drone research converges to two broad concepts — semi-autonomous and fully autonomous. Semi-autonomous describes the software development pertaining to responsiveness of drone. Fully autonomous integrates the artificial intelligence, navigational algorithms and recommendation engine. Following is the summary of the autonomous concepts:

Semi-Autonomous (Software part):

1. Set up of streamlined data flow. This creates a robust I/O stream on both interfaces, GS-RPI3 and RPI3-PX4.

2. Drone responsiveness.

3. Precision in control system to achieve attitude and altitude.

4. Fundamental understanding and interpretation of software system available. E.g. ROS, PyDrone, DroneKit-Py, ArduPilot, MavProxy and other libraries.

Fully Autonomous (Algorithm part):

1. Traveling from A to B knowing the obstacles before the start of mission.

2. Reading 2D/3D data and implementing pathway.

3. Real-time implementation moving obstacles such as person or ball.

For this study, our objective is to create a semi-autonomous Drone System which could be later developed into a fully autonomous system.

CHAPTER 4

DRONE TECHNOLOGY

Drone technology is developed on two independent fronts — the drone makes and materials, and the in-built control system. In this project we are mostly dealing with the control system which inherently controls the drone motor movements. To enable the 3D movement, the drone is equipped with four motor — two running in the clockwise (CW) direction and two running in the counter-clockwise (CCW) direction assembled in alternative fashion.

The physical movement of a drone can be understood using two concepts — altitude and attitude. Altitude is related to the height of the drone above ground. Altitude is controlled by powering up all motors to go up and powering down to go down. Attitude is related to yaw, roll, and pitch of the drone. Considering the right-hand Cartesian coordinates, the three rotational component yaw, roll and pitch will rotate on Y, Z, and X axis, respectively. Figure 4.1 describes the rotational component of the Drone System.

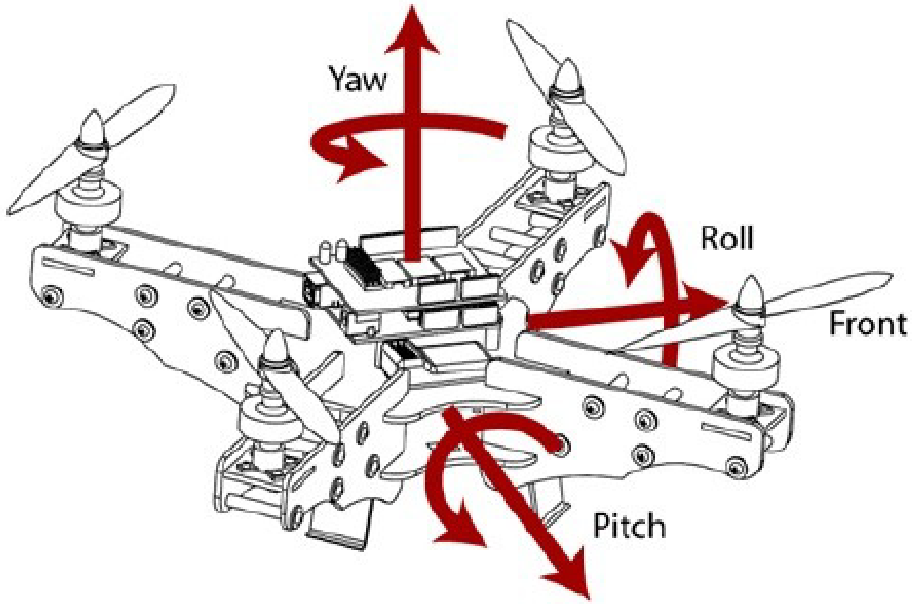

Figure 4.1. Attitude control in drone system (Sopasakis, 2017).

In order to achieve yaw, the two counter-clockwise motors power up to go left and the two clockwise motors power up to go right. Similarly, for roll, the right two motors power up to go left and left two motors power up to go right. Pitch controls going forward where the back two motors power up and going backwards where the front two motors power up. Figure 4.2 describes the possible movements in the Drone System.

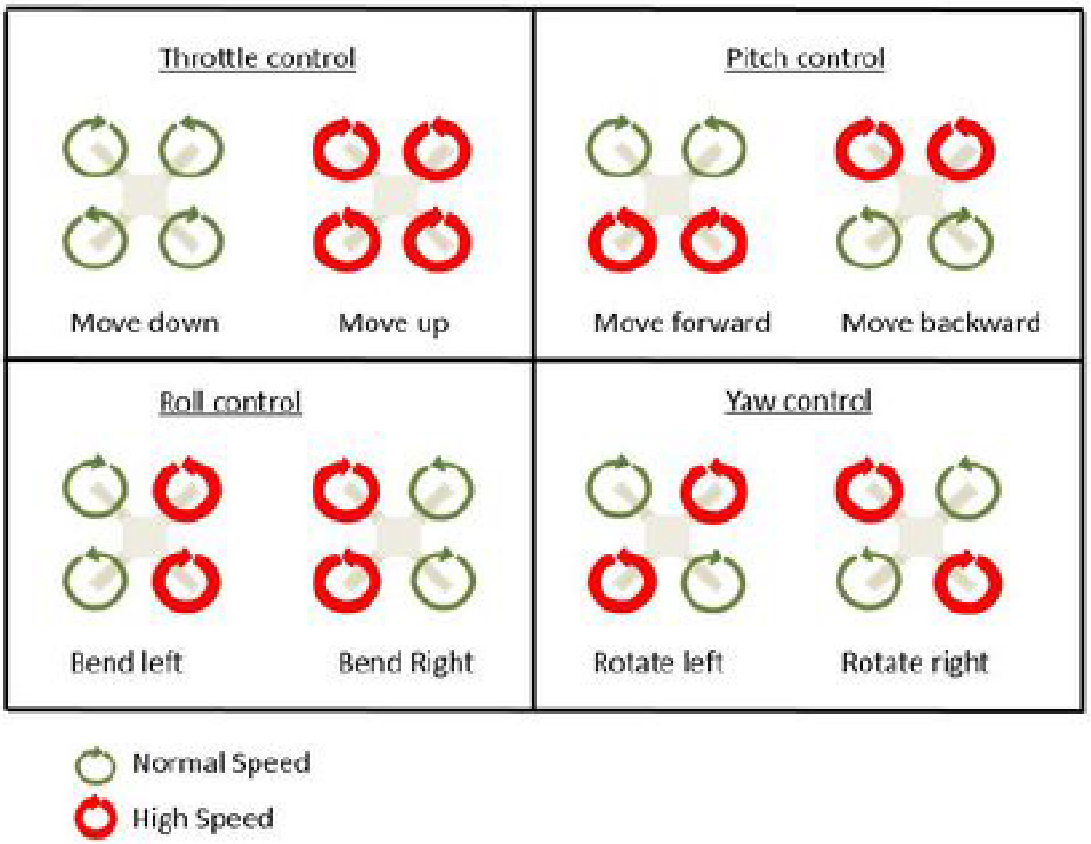

Figure 4.2. Possible movements in drone system. (Sopasakis, 2017)

Apart from motor control, the drone is equipped with various sensors to control and verify movement. Common sensors that are used on a drone system includes a gyroscope that controls attitude, a barometer that controls altitude, a GPS that verifies localization, and cameras that enable object avoidance.

CHAPTER 5

HARDWARE STACK

The foundation of the drone is based on the DJI Flame Wheel F450 Dronekit which comes with its own frame, motor unit, propellers and power distribution board. Most of the connections are wired and soldered as instructed in the kit manual. These connections, as well as the connections with newer module such as sonar, Companion Computer, Ground Station, etc. are shown in Figure 5.1 below. Most of the research on parts and assembly of the Drone System was done in earlier studies at HEROES lab (Gossett, 2018). The description of each part is summarized in further sections.

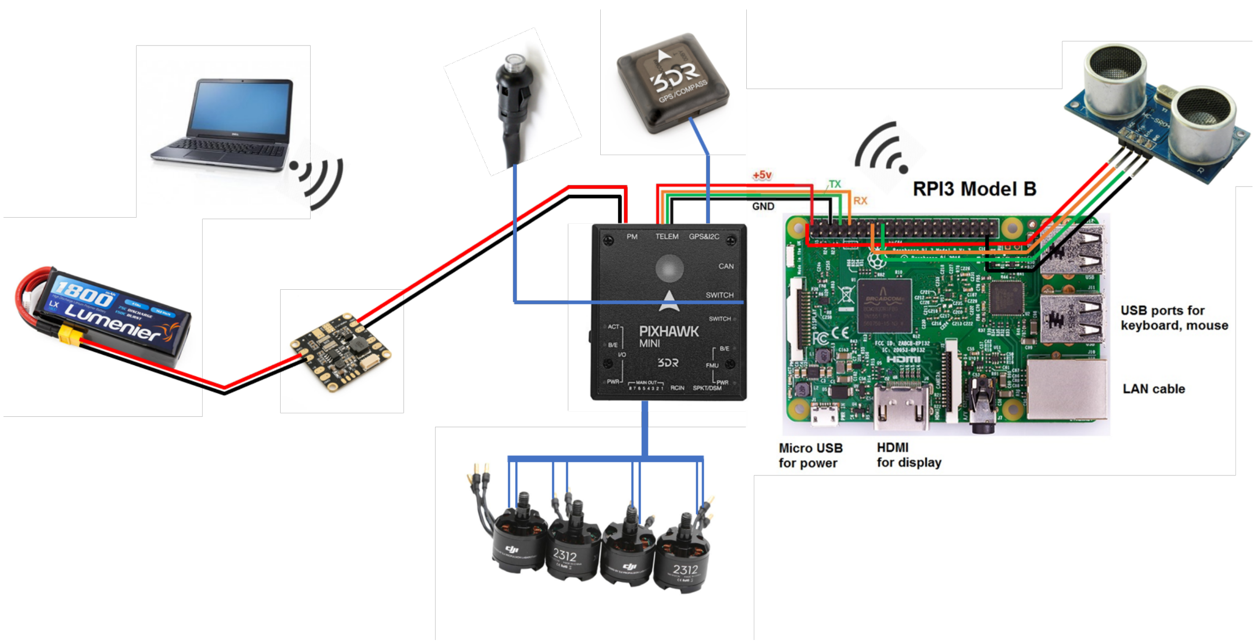

Figure 5.1. Connection on Hardware Interfaces.

5.1 Frame

As suggested earlier, the frame was part of “Almost Ready to Fly” Flame Wheel Dronekit F450 and, hence, was easy to assemble. The frame is made of hard ”ultra-strength” plastic component. The diagonal wheelbase (motor to motor distance) of the frame is 450mm and the frame naked weight is 282g. The propellers are 250mm long which adds roughly another 100 mm to diagonal wheelbase per propeller, making the drone safe diameter to 650 mm.

Figure 5.2. Frame for DJI Flame Wheel F450 Dronekit (QuadcopterGarage.com, 2018).

5.2 Flight Control Unit

As suggested earlier, the Flight Control unit controls the motors in order to keep the quadcopter in the air. Overall, the Flight Controller is a circuit board with sensors that detects the orientation changes of the drone, and receives user commands. In our study, we used Pixhawk Mini (PX4) Flight Controller from 3DR, which is equipped with the ARM Cortex M4 processor. All the sensors and command flow are connected directly (wired) or indirectly (via the companion computer) to the PX4. The detailed specifications of the Flight Controller Unit could be checked at PX4 Dev Team (2019).

Figure 5.3. PX4 Autopilot Unit with an External Compass Unit (PX4 Dev Team, 2019).

5.3 External Compass Unit

Although there is an inbuilt compass unit in the PX4, it has been depreciated due to a hardware manufacturing defect (davids5, 2017). The hardware defect makes the internal MPU9250 IMU unreliable. The MPU9250 is disabled by default on the PX4 firmware. An external GPS/compass unit is provided which solves this problem, as a secondary sensor ICM20608 provides both the accelerometer and the gyroscope, while the external GPS provides the magnetometer (PX4 Dev Team, 2019). Figure 5.3 shows the wired connection made between the PX4 and the External Compass Unit.

5.4 Companion Computer

For our study, we have used the Raspberry Pi 3 (RPI3) as a companion computer, which connects to the Flight Control Unit using a wired connection between the only telemetry port available on PX4 and the UART connection built into RPI3 GPIO. The RPI3 is equipped with an

ARM Cortex A53 processor which reads and interprets the sensor data and sends commands to PX4. In further chapters we would go in a more detailed RPI3 setup procedures.

5.5 Motor Unit and Electronic Speed Controller (ESC)

The Dronekit comes with four DJI 2312 motor units as shown in Figure 5.4. Each motor weights 60g, provides 960 rpm/V, and has a maximum recommended load of 400g for a 4S LiPo Battery setting. The Drone System is also equipped with four DJI 420 LITE ESC as shown in Figure 5.4. The ESC regulates power to the motor and is paired with each motor. It is compatible with CW and CCW motors from E310 Tuned Propulsion System (2312). It has maximum recommended allowable voltage as 17.4V and maximum allowable current as 30A. The ESC sends PWM voltage values as the output to motors. The detailed specification for the propulsion system could be checked at DJI (2014).

Figure 5.4. E310 Tuned Propulsion System.

5.6 Sonar Sensor

For the study, we have used HC-SR04 Ultrasonic Range Finder that points down to give distance measurements from the ground. The sensor has four pins: Ground (GND), Echo Pulse Output (ECHO), Trigger Pulse Input (TRIG), and 5V Supply (Vcc). Its connections to the RPI3 are made as described in Figure 5.5. The output ECHO is reduced to 3.3V using a voltage divider circuit in order to match the recommended GPIO port rating (Gossett, 2018).

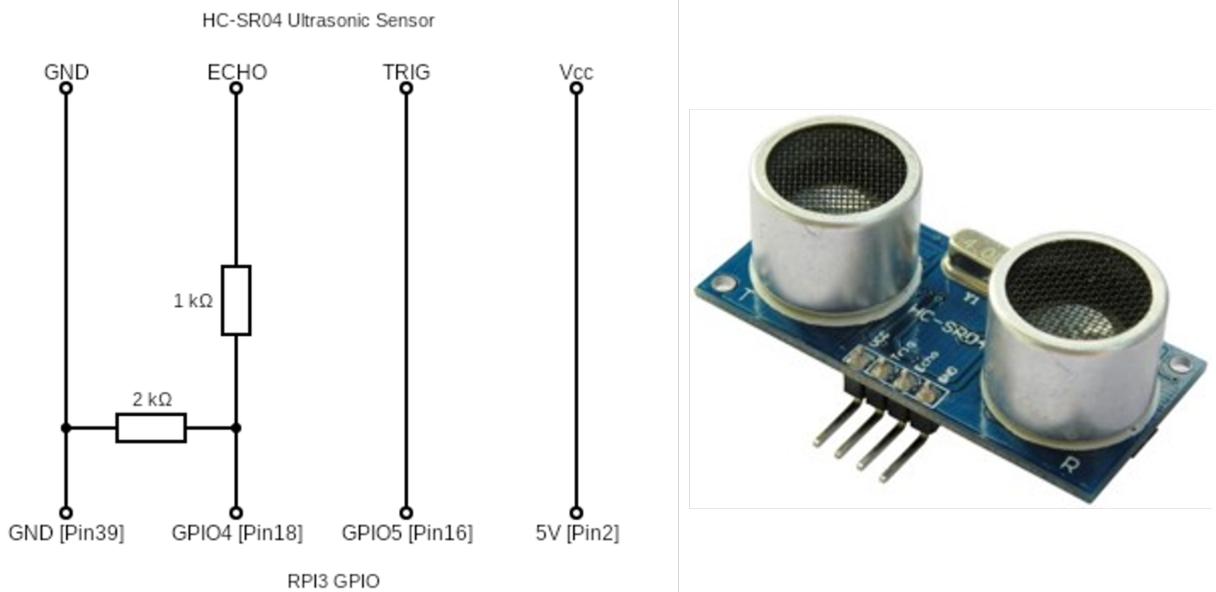

Figure 5.5. HC-SR04 Ultrasonic Range Finder Sensor and Connection with Raspberry Pi.

5.7 Safety Switch

The Flight Controller has an integrated safety switch that can be used to enable/disable the outputs to motors and servos, once the autopilot is ready to take off. If this switch is hard to access, an external safety button is also attached to the Flight Controller.

5.8 Battery and Power Module

The Drone System is internally powered using 1800 mAh 4S Lipo battery rated at 14.8 V. Normally, the battery would get discharged within 15 to 20 minutes of flying use. It should be noted that if the battery gets discharged to less than 13.5 V (approx 90% of rated voltage), it could cause permanently damage. The battery is charged using EV-Peak Synchronous Balance Charger/Discharger.

Although, the ESC can run on 14.8V delivered directly from LiPo battery, the radio, telemetry and flight controller run on 5v. In order to deliver an additional 5V, we can either use an another

5v battery or use a Battery Eliminator Circuit (BEC). There are some ESCs that have built-in BEC and puts 5v on the signal cable, but the DJI 420 LITE ESC, which we are using, does not have an in-build BEC. Therefore, the Drone System is equipped with a Micro Power Module, a PM06-v1.0 UBEC (Universal BEC) which converts the main LiPo battery pack voltage, 14.8V, to a lower voltage, 5V. The power module can deliver a maximum current of 3A on DC input of 7V-42V. The power output of the module is 5.1V-5.3V.

5.9 Ground Station

The ground station is a general-purpose computer which is used as development and testing machine. In our case it was a 4th generation Intel i3 with 12 GB RAM. The ground station is equipped with Windows 10, Cygwin VM, Eclipse IDE, Mission Planner, Python development framework, Wi-fi and USB capabilities.

CHAPTER 6

ARCHITECTURE

With technological advances, the Drone Software Development is now incentivized for more powerful systems. This has enabled more processing power and support for a graphical user interface. There are many such software packages available commercially and we have divided them into four broad groups based on functionality - Flight Control Firmware, GUI based GCS, Terminal based GCS, and Companion Computer Software.

The Flight Control Firmware, viz. ArduPilot based on the GPL license, PX4 Flight Stack based on the BSD license, iNav, Librepilot, etc. mostly interprets sensor data to control the output to motor units. GUI based GCS, viz. Mission Planner, QGroundControl, etc. are ground station-based software which sends user commands to firmware and displays drone current status. It is computation intensive in nature. Terminal-based GCS, viz. MAVProxy, also sends user and status commands to the firmware, but it is less resource consuming in that it does use a GUI and does not perform additional functionality such as features related to map, testing and simulations. The companion Computer Software includes libraries and framework such as DroneKit-Python, APSync, Maverick, etc. which create apps that run on an on-board companion computer and communicate with the Flight Controller.

The current Drone System is divided into three major development modules:

1. Desktop module, which contains the dev and test box
2. Flight Controller module
3. RPI3 module

During this study, there have been efforts taken to streamline the process of development, deployment and testing of new features. As a result, we selected an architectural pattern which tends to be easier to debug when the system errors out.

In the current setup as shown in the Figure 6.1, the Flight Controller module runs the Ardupilot firmware on the PX4 hardware, the RPI3 module runs the MAVProxy GCS software and the Desktop module runs the Mission Planner for deployment to the firmware, Secure Shell for testing (test box) and C++ development framework (dev box).

The RPI3 acts as a companion computer sharing processed data with the firmware over MAVLink Messages. Additional sensors are attached to the RPI3 to create a scalable integration of sensor technology into the Drone System. In the future, the scope of such integration is not limited to sensors but may include other technologies as well.

The MAVProxy based RPI3 acts like a true middleware with two-way communication to firmware, enabling integration of sensor technology, enabling testing/user command interface, and converging to a modular approach for software development.

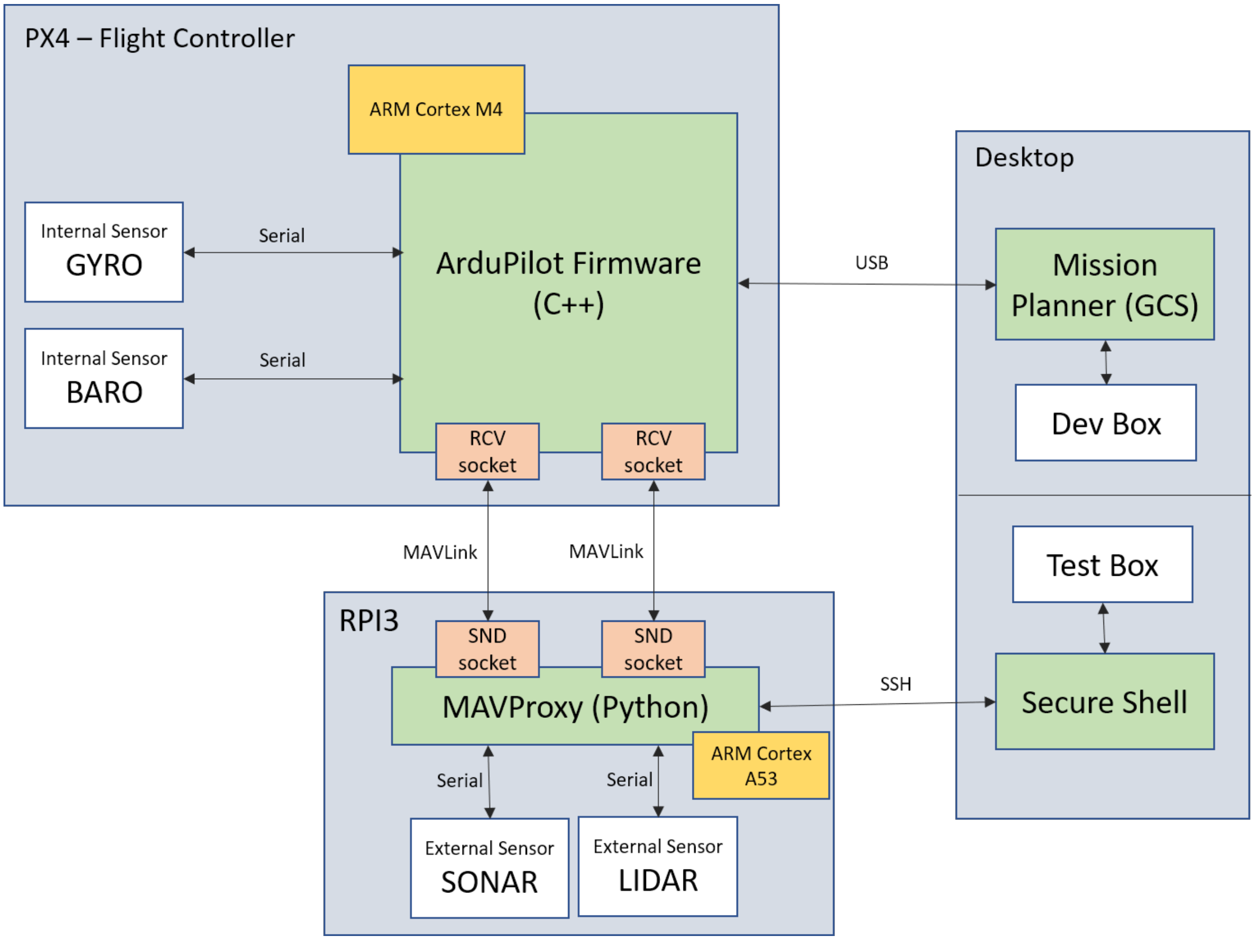

Figure 6.1. Current Environment Setup.

6.1 ArduPilot

ArduPilot is an open source autopilot software and is capable of controlling vehicle systems, including conventional airplanes, multirotors, and helicopters, boats and submarines. It is one of the distinguished autopilot suites and is used in many OEM UAV companies, such as 3DR, jDrones, PrecisionHawk, AgEagle and Kespry; several large institutions and corporations such as NASA, Intel and Insitu/Boeing; and by countless colleges and universities around the world. (ArduPilot, 2016a)

ArduPilot (including Copter, Plane, Rover, Sub and AntennaTracker) and the ground control software (including Mission Planner, APM Planner2 and MAVProxy) are free software under the terms of the GNU General Public License version 3 as published by the Free Software Foundation. We may copy, distribute and modify the software as long as we track changes/dates of in source files and keep modifications under GPL. We can distribute our application using a GPL library commercially, but we must also provide the source code (ArduPilot Dev Team, 2019f).

The ArduPilot code base is quite large (about 700k lines for the core ardupilot git tree) and can be quite intimidating to a new user. The Dev team has documented the procedures and libraries well. The documentation is available online on the ArduPilot Dev Team (2019b) website. To understand the code at an abstract level, the code base runs two type of loops: a single open loop that runs all the 'core' libraries and methods and closed loops for every sensor to execute 'sensor' libraries and methods. For the single open loop, when an input is sent to ArduPilot, setters methods are called to set flags so that action methods could be executed during the next loop cycles. The multiple closed loop calls sensor methods regularly to fetch/read data from sensors and sets corresponding float variables. As an example, the AL_BARO, AL_CAMERA libraries will contain the 'sensor' methods for barometer and camera sensors. It should be noted that print methods are also available in AL_HAL and GCS libraries.

Figure 6.2 explains the overall ArduPilot architecture using data flow patterns in the system starting at main loop() in ArduCopter.cpp library and ending at outputting PWM to motors.

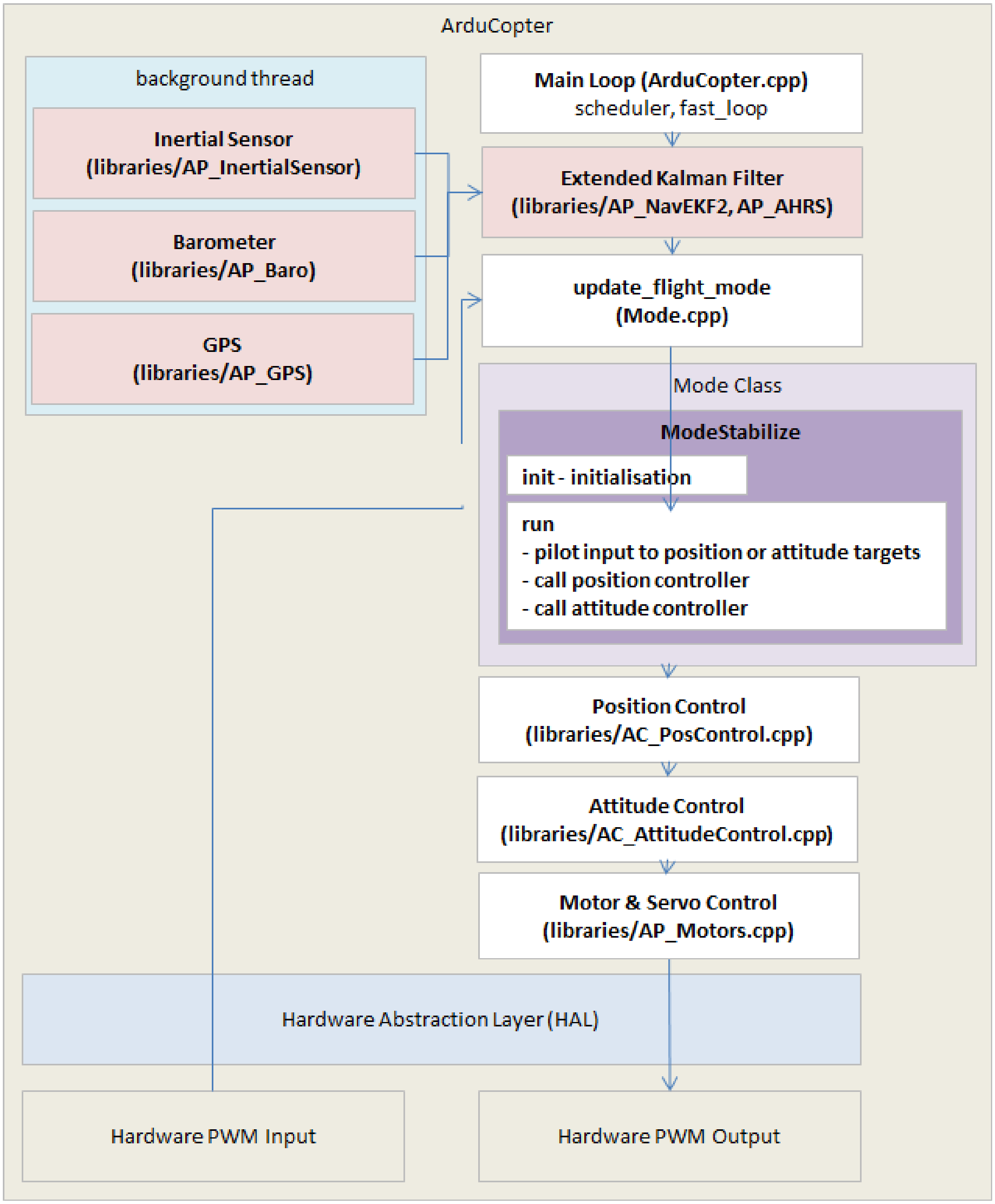

Figure 6.2. ArduPilot Architecture.

The UAV flies at various pre-defined flight modes such as *Alt Hold* which holds altitude and self-levels the roll & pitch, *Flip* which completes an automated flip, *Guided* which navigates drone to single points commanded by GCS, etc. Currently, there are more than 20 flight modes (ArduPilot Dev Team, 2019e). Flight Mode provides an abstraction, as well as coding space where we can write custom code for flight stabilization, autopilot, setting of variables, setting of sensor flags, etc. Figure 6.3 shows the architecture of manual modes, i.e., modes that requires a manual control, viz., *Stabilize*, *Acro*, *Drift*, etc.

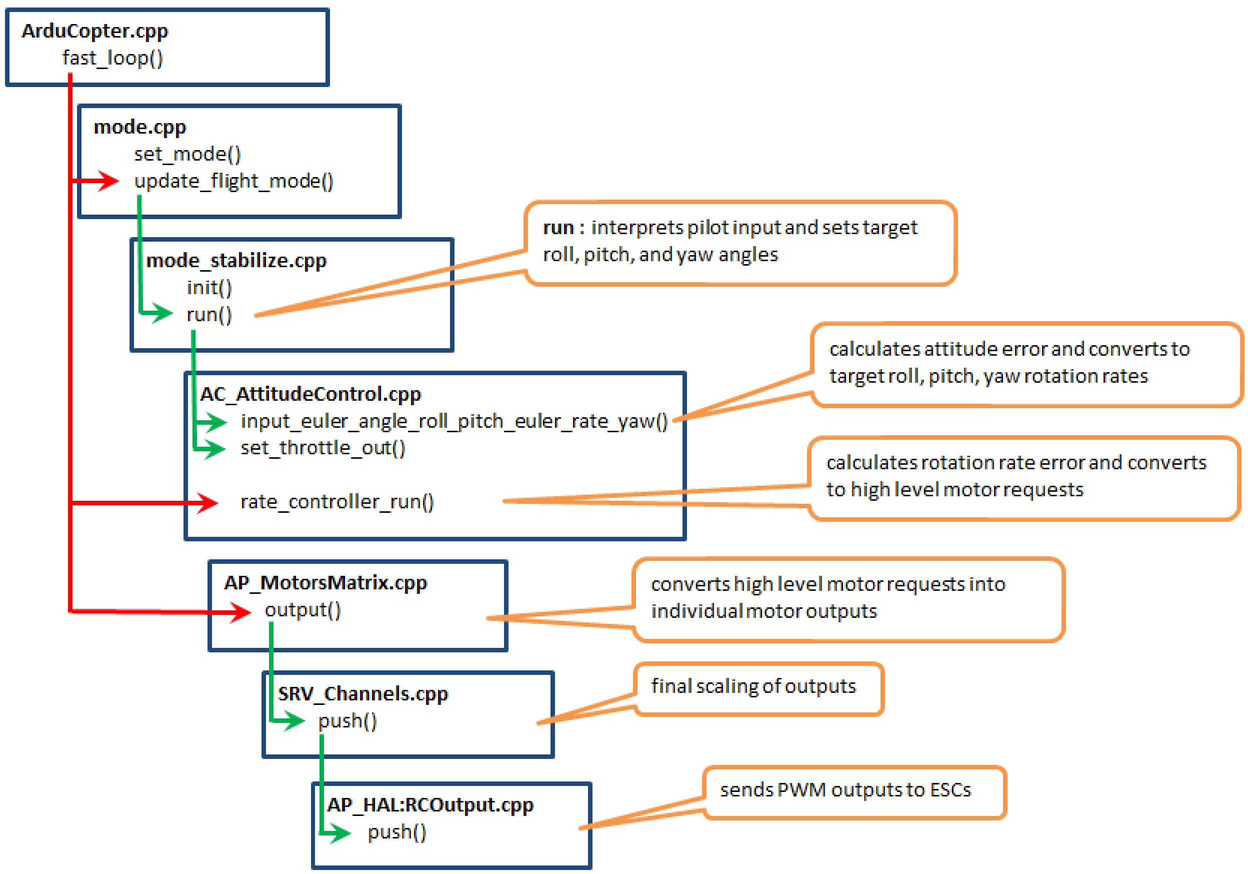

Figure 6.3. Sample ArduPilot Code Overview for Flight Mode.

6.2 MAVProxy

MAVProxy is a GCS terminal for MAVLink based UAV systems. MAVProxy was first developed by CanberraUAV to enable the use of companion computing and multiple datalinks with ArduPilot. It supports loadable modules, and has modules to support console/s, moving maps, joysticks, antenna trackers, etc. MAVProxy is too released under the GNU General Public License v3 or later (ArduPilot, 2016b).

The source code for MAVProxy is written in python. The MAVProxy python-based modules run in parallel threads to the main MAVProxy program in order to maintain overall stability. Modules need to be loaded before they can be used using management commands.

As suggested earlier, the software has modules to manage similar functionality and each functionality uses MAVLink messages to communicate to ArduPilot firmware. Definitions of these MAVLink messages are kept in *common.xml* and *ardupilot.xml* files. In order to be reusable, the definition of these messages is converted to both python and C++ header code using python pymavlink library, so that it could be referenced in both MavProxy and ArduPilot source code. It could be noted, therefore, the python MAVProxy library have a dependency on pymavlink library.

6.3 MAVLink

Micro Air Vehicle Link (MAVLink) is a lightweight messaging protocol for communicating with small unmanned vehicles (and between on-board drone components). MAVLink follows a modern hybrid publish-subscribe and point-to-point design pattern: Data streams are sent / published as topics while configuration sub-protocols such as the mission protocol or parameter protocol are point-to-point with retransmission.

It is designed as a header-only message marshaling library. MAVLink 2 has just 14 bytes of overhead and is a much more secure and extensible protocol than compared to its predecessor, MAVLink 1. As a result, it is highly efficient and well suited for applications with very limited communication bandwidth. It also provides methods for detecting packet drops and corruption, and for packet authentication.

As suggested earlier, messages are defined within XML files. Also, internally the Python pymavlink library uses the mavgen and mavgenerate tools provided by the MAVLink project to convert these XML files to Python and C++ header codes. The mavgen and its derivative tools also support other languages such as C, C#, Objective C, Java , Javascript, Lua, Swift and Clojure. (Dronecode Project, 2018)

Figure 6.4 shows a sample MAVLink message with predefined message id, message name, variable type and variable name sent/received during communication in xml format.

```xml
<message id="24" name="GPS_RAW_INT">
        <description>The global position, as returned by the Global Positioning System (GPS)
system, but rather a RAW sensor value. See message GLOBAL_POSITION for the global position e
up (GPS frame).</description>
        <field type="uint64_t" name="time_usec">Timestamp (microseconds since UNIX epoch or
        <field type="uint8_t" name="fix_type">0-1: no fix, 2: 2D fix, 3: 3D fix. Some applic
it is at least two, so always correctly fill in the fix.</field>
        <field type="int32_t" name="lat">Latitude (WGS84), in degrees * 1E7</field>
        <field type="int32_t" name="lon">Longitude (WGS84), in degrees * 1E7</field>
        <field type="int32_t" name="alt">Altitude (WGS84), in meters * 1000 (positive for up
        <field type="uint16_t" name="eph">GPS HDOP horizontal dilution of position in cm (m*
        <field type="uint16_t" name="epv">GPS VDOP horizontal dilution of position in cm (m*
        <field type="uint16_t" name="vel">GPS ground speed (m/s * 100). If unknown, set to:
        <field type="uint16_t" name="cog">Course over ground (NOT heading, but direction of
If unknown, set to: UINT16_MAX</field>
        <field type="uint8_t" name="satellites_visible">Number of satellites visible. If unk
</message>
```

Figure 6.4. Sample MAVLink message.

CHAPTER 7

SOFTWARE INTEGRATION

The software integration involves setting up the two interfaces between the three software modules: the RPI3-PX4 Interface and the GS-RPI3 Interface. Also, we need to set up the Desktop Test/Dev Box on GS and the Sonar sensors into the Drone System.

7.1 Setting up GS-RPI Interface

Assuming that the previously stated hardware setup and connections have been made, enabling the on-board RPI3 integration with the Desktop Ground Station (GS) module requires the following step. (ArduPilot Dev Team, 2019c)

1. Download Raspbian from provided by the Raspberry Pi Foundation (2018). The current version installed on the RPI3 is Raspbian Stretch updated 2018-11-26.

2. If the Operating System (OS) image has a dot iso extension file, use Rufus to copy the OS image to an SD Card. This is provided by Batard (2019). If the OS image has a dot img extension, use Etcher to copy the OS image to an SD Card. This is provided by Balena (2019). Using Win32 Disk Imager, provided by Davis (2019), might require formatting the SD Card using SDCardFormatterv5 provided by SD Association (2019). Select the time-consuming overwrite format option and not the quick format option.

Note: By default, we cannot read Linux file system/drive on Windows. Reading an SD Card will give error on Windows Explorer. Also, if we are able to see a partition, it is the `boot` partition, which contains the system settings, and not the user data on the OS. The user data is kept on Linux based partitions.

3. **Plug-in the RPI3** This might take some ten minutes when booted up for the first time. Connect it to Wi-Fi.

   Note: Currently the connection is set up as dynamic. Ole Miss Network does not allow to make static IPs. We would need our own router to make static IPs or get Network Administrator onboard.

4. **Disable the OS control of the serial port:** Enter `sudo raspi-config` on terminal, `Interfacing Options / Serial` then specify if you want a Serial Login Shell (Select `No`) then if you want the Serial Port hardware enabled (Select `Yes`).

   Note: As a thumb rule, use `/dev/serial0` in configuration for any code that accesses the Serial Port. This is because, in the latest RPI3 updates, defining `/dev/serial0` automatically selects the appropriate device; so you can replace `/dev/ttyAMA0` with `/dev/serial0` and use the same software configurations on Pi3 and earlier models. To understand this in more detail - the `/dev/ttyAMA0` was previously used to access the UART now connects to Bluetooth. Hence, we connect PX4 (Flight Control Unit) with miniUART now available on `/dev/ttyS0`.

5. **Enable UART:** For some reason the default for RPI3 using the latest 4.4.9 kernel is to `DISABLE UART`. To enable it we need to change `enable_uart=1` in `/boot/config.txt`.

   Note: This step solves two problems, One, enables UART, and second, for RPI3 ”Baudrate is derived from system clock” which makes the miniUART useless as this clock can change dynamically when the system goes into reduced power or in low power mode. To fix this problem we modify the `/boot/config.txt` and remove this dependency by adding the following line `core_freq=250`. This change is not needed anymore. The SPI clock frequency and ARM Timer are also dependent on the system clock. (Raspberry Pi Foundation, 2019a)

6. **Enable SSH:** Enter `sudo raspi-config` on terminal, `Interfacing Options` and `Enable SSH`, and then execute the following commands.

```
$ sudo apt-get update
$ sudo apt-get install openssh-server openssh-client
```

   We can verify the active status of the ssh service by executing:

```
$ sudo systemctl status ssh
```

7. **Check IP Address:** Use `ifconfig` CLI command to find the IP Address which would be used for secure login (SSH) from remote ground station (GS).

8. **Install Mission Planner:** Follow the instructions provided by ArduPilot Dev Team (2019g) in order to install Mission Planner GCS.

7.2 Setting up RPI-PX4 Interface

Currently we are going to leverage the ssh connection which we have enabled in the last section to communicate with the RPI3 module. In future the connection over TCP capability should be researched. It would be important to know how the drone behaves on the TCP/IP given the protocol is highly reliable but memory consuming. Following are the softwares we would need:

1. **Install Libraries:** From GS CLI, enter `ssh pi@<IP Address Here>` and execute these commands on GS successively: (change ardupilot.xml in message def folder and copy the folder in pymavlink clone)

```
$ sudo apt-get install python-wxgtk2.8 python-matplotlib
  python-opencv python-pip python-numpy python-dev
  libxml2-dev libxslt-dev python-lxml
$ sudo pip install future
$ git clone https://github.com/ArduPilot/MAVProxy.git
$ git clone https://github.com/ArduPilot/pymavlink.git
```

Copy the *message definition* folder from newly created *mavproxy* folder to *pymavlink* folder. This is done to remove the dependency of pymavlink library on MAVProxy library. We are not using pip to install the libraries would be outdated and have dependency constraint. Instead, we are installing the most recent version from git. Execute these further commands:

```
$ cd ./pymavlink
$ sudo python setup.py build
$ sudo python setup.py install
$ cd ./MAVProxy
$ sudo python setup.py build
$ sudo python setup.py install
```

We might get errors related to modules not found, but need not worry as they are warnings and could be skipped. To remove the warnings, comment out the modules in *setup.py* file.

2. **Verifying RPI-PX4 Connection:** Execute these command on GS CLI successively.

```
$ sudo mavproxy.py --master=/dev/serial0
--baudrate 57600 --aircraft MyCopter
```

Whenever the RPI3 connects to the PX4, three files are created in the in the folder where the command was executed. If the logs were set up, the files would be created at `/home/pi/MyCopter/logs/YYYY-MM-DD` directory.

- *mav.parm* : a text file with all the parameter values from the Pixhawk.
- *flight.tlog* : a telemetry log including the vehicles altitude, attitude, etc which can be opened using the mission planner (and other tools).
- *flight.tlog.raw* : all the data in the *flight.tlog* mentioned above plus any other serial data received from the Pixhawk which might include non-MAVLink formatted messages like startup strings or debug output

Note: If MavProxy is set at higher baud rates, we could get error on the receiving end. In my case I got `MAV > Link 1 Down` error for communicating with PX4 at a high baud rate (Denotes PX4 TELEM port Pin 1 - RCV error / Handshake error). Here, we are setting baud rate at TELEM default rate of 57600. This is half of the highest baud rate for most current industry microcontrollers.

7.3 Setting up Testing Box

Testing for the drone system is set up in Windows machine (i3-4th gen, 12 GB RAM) and the development/builds are done with Eclipse IDE (Photon 2018-12). The steps followed are based on documentation available on ArduPilot Development Site (ArduPilot Dev Team, 2019a). Also, it is important to note that Python 2.7 will reach the end of its life on January 1st, 2020 and will be deprecated and there could be changes to the libraries we need to reconsider. Following are the updated steps used for this project:

1. Download Eclipse IDE for C/C++.

2. **Install Cygwin:** Follow the documentation provided by Cygwin Authors (2019) and then install the following in-software packages - autoconf, automake, ccache, gcc-g++, git, libtool, make, gawk, libexpat-devel, libxml2-devel, libxslt-devel, python2-devel, python2-future, python2-libxml2, procps-ng, gdb, ddd, zip, libcrypt-devel

3. **Start Cygwin Terminal:** Open and close Cygwin Terminal. This creates initialization files for the user in Cygwin home directory.

4. **Install pip:** We will now configure Cygwin based Linux VM. Execute these commands in Cygwin terminal:

```
$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
$ python get-pip.py
```

5. **Install the GCC compiler:** firmware.ardupilot.org/Tools/STM32-tools. Remember to check option Add path to environment variable while installing.

6. **Install packages:** Install the following packages in Cygwin Terminal:

   `$ pip install empy`

   `$ pip install pyserial`

   `$ pip install pymavlink`

7. **Clone ArduPilot project from Git:** Excecute the following command:

   `$ git clone https://github.com/ArduPilot/ardupilot.git`

   Its possible that some files might be missing after clone (In my case, ardupilot/Module folder was empty). In that case, run the following command to update recursively.

   `$ git submodule update --init --recursive`

8. **Setup Eclipse project:** Follow the manual provided by ArduPilot Dev Team (2019d). Change the target name to `configure --board Pixhawk1 --no-submodule-update` when creating the `New Build Target`.

9. **Building ArduPilot project:** Px4 currently compiles with `waf` instead of `make` commands. Building with `waf` library is described in *ardupilot/BUILD.md*. For compiling in Eclipse, open it with Administrative Privileges, right click the project and click Build. For the first build, it might take a lot of time (In my case, it was more than an hour). For this reason, it is not recommended to do a clean before every build. For errors on Eclipse related to 'Unresolved Inclusion Error' follow instructions provided by Hock-Chuan (2013).

   Alternatively, we can build using `waf` commands on Cygwin Terminal. This was the standard process in our study. In the git/ardupilot folder execute the following commands:

   `$ ./waf configure --board PixHawk1`

   `$ ./waf build --all` **or** `./waf build copter`

10. **Installing Mission Planner:** Follow instruction to install provided by ArduPilot Dev Team (2019g). Issues related to no port found on COM# (USB) are mostly because USB serial device is not set up or disconnected. To set up the device, open Device Manager > View > Show hidden devices. Go to Ports > Right click USB Serial device (or COM6, in my case) > Update Driver. Other reason could be a case of having a bad USB cable. See section on challenges for further details.

11. **Loading Firmware:** Connect the PX4 with your Windows System using USB port. Open Mission Planner. Click `Initial Setup` tab and select `Install Firmware`. Click `Load custom firmware` on the bottom right of the window and select the newest arducopter build (usually in `build/<yourConfiguredBoard>/bin` folder). Follow the instructions during installation.

7.4 Setting up Sonar Sensor

Raspberry Pi is equipped with row of 40-pin GPIO (general-purpose input/output) along the top edge of the board as shown in Figure 7.1. As well as simple input and output devices, the GPIO pins can be used with a variety of alternative functions, viz. PWM, SPI, I2C, Serial. (Raspberry Pi Foundation, 2019b)

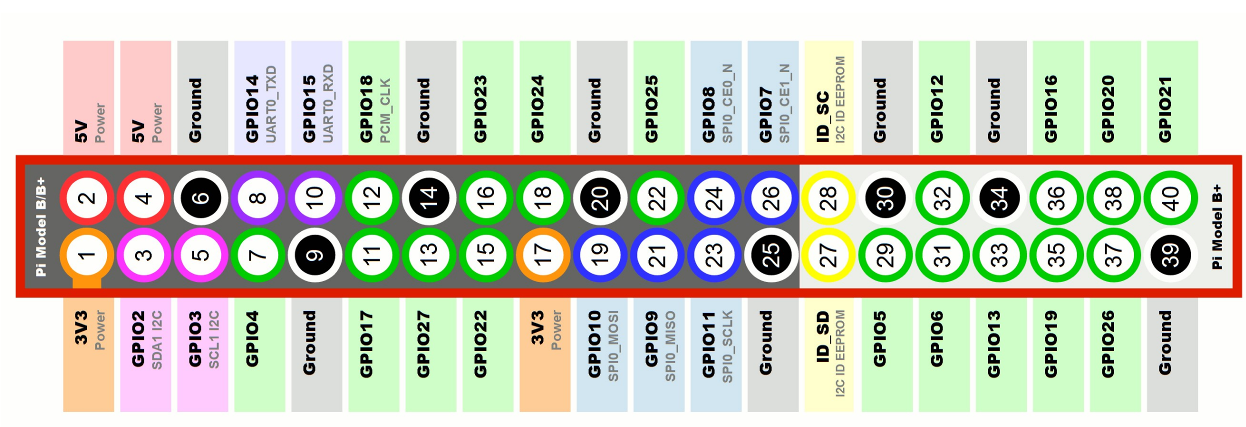

Figure 7.1. Raspberry Pi GPIO Layout Pi 3 Model B. (Hawkins, 2018)

Follow Figure 5.5 in earlier sections for hardware connections enabled with RPI3 and Sonar

Sensor. The sensor's four pins - Ground (GND), Echo Pulse Output (ECHO), Trigger Pulse Input (TRIG), and 5V Supply (Vcc) is connected to Pin 39, Pin 18, Pin 16 and Pin 2 on GPIO respectively.

For software setup and changes to Python code, see section on integration testing for testing of sonar measurement.

CHAPTER 8

INTEGRATION TESTING

For testing the system, we have taken an approach that combines individual modules and tests them as a group. These tests verify the feasibility of the proposed architecture along with the viability of communication links for a future scalable model for development of the Drone System.

8.1 Testing Motor Arm using MavProxy on GCS

This test executes Motor Arm command from GS terminal, which powers up the motor unit remotely. This test verifies two aspects

1. Communication between GS and RPI
2. Communication between RPI and Firmware/Motor Unit.

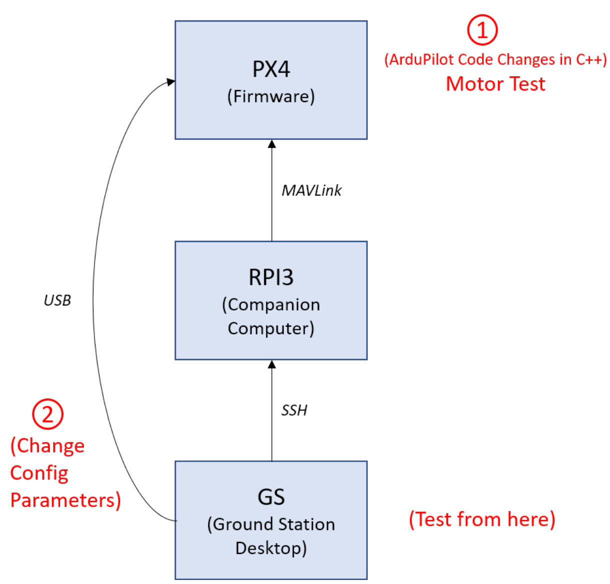

Figure 8.1. Data Flow in *Motor Arm* Test.

There was two ways the test was completed - through change in configuration parameters as shown as 1 in figure 8.1 or setting up boolean flags in the firmware as shown as 2 in figure 8.1, to make the system armable. The reason we had to do two alternative tests is described in detail in following chapter 9 on Challenges, where we were not able to reproduce the results by setting configuration parameters. Following were the steps undertaken:

1. **Connect to firmware:** Open Mission Planner on desktop. Connect to the firmware via USB. Click `Connect` on top right of the windows.

2. **Set Configuration Parameters:** In Mission Planner windows, Click on `Config/Tuning` tab. On left Pane, Select `Full Parameter Tree`. Use the Search Box on the left pane and set the values of the following parameters: `GPS_ENABLE` to 0, `EKF_ENABLE` to 0, `ADHS_ENABLE` to 0, `ARMING_CHECK` to 0

3. **Update Firmware:** On Mission Planner windows on the right pane, click on `Save to file` button. Alternatively, this could be also done on MAVProxy GCS by using `param fetch` and `param set` commands.

4. **Execute Test:** Execute the following commands on MAVProxy GCS:

   ```
   $ sudo mavproxy.py
   $ motortest 1 0 30 2
   ```

   This will power the *motor 1* at *30% of maximum* output for *2 seconds*. The following figure 8.2 shows results on the command prompt

```
pi@raspberrypi:~/test $ sudo mavproxy.py
Auto-detected serial ports are:
/dev/ttyS0
Connecting to /dev/ttyS0
Connect /dev/ttyS0 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from /dev/ttyS0
 MAV> online system 1
STABILIZE> Mode STABILIZE
fence breach
APM: ArduCopter V3.7.0-dev (2834fd80)
APM: ChibiOS: 35ad8a79
APM: Pixhawk1 001E002F 30365104 31323731
APM: Frame: QUAD
Flight battery 100 percent
Received 828 parameters
Saved 828 parameters to mav.parm

STABILIZE> motortest 1 0 30 2
STABILIZE> APM: mavlink_motor_test_start: motor is armed
APM: mavlink_motor_test_start: setting input time for m
APM: mavlink_motor_test_start: motor calibration compas
Got MAVLink msg: COMMAND_ACK {command : 209, result : 0}
APM: motor_test_output: Power to be provided to motor
APM: motor_test_output: motors to be turned on
APM: AP_Motorsmatrix: writing values to motor
APM: motor_test_output: Power to be provided to motor
APM: motor_test_output: motors to be turned on
APM: AP_Motorsmatrix: writing values to motor
APM: motor_test_output: Power to be provided to motor
APM: motor_test_output: motors to be turned on
APM: AP_Motorsmatrix: writing values to motor
APM: motor_test_output: Power to be provided to motor
APM: motor_test_output: motors to be turned on
APM: AP_Motorsmatrix: writing values to motor
APM: motor_test_output: Power to be provided to motor
APM: motor_test_output: motors to be turned on
APM: AP_Motorsmatrix: writing values to motor
ARMED
Arming checks disabled
DISARMED
APM: motor_test_output: motors to be turned on
APM: AP_Motorsmatrix: writing values to motor
APM: mavlink_motor_test_stop: finished motor test
APM: mavlink_motor_test_stop: disaming motors
APM: mavlink_motor_test_stop: motor compass calibration

STABILIZE>
```

Figure 8.2. Testing Arm Motor command on MaxProxy Terminal.

Alternatively, we can set flags in the firmware source code to make the system armable. Following steps were taken:

1. **Update the source code:** Open the source code using IDE, update the file *ArduCopter/-moter_test.cpp* as shown in following code snippet. Line 14-18 is replaced by line 20-30.

```
1 MAV_RESULT Copter::mavlink_motor_test_start(mavlink_channel_t chan,
      uint8_t motor_seq, uint8_t throttle_type, uint16_t throttle_value,
      float timeout_sec, uint8_t motor_count)
2 {
3   if (motor_count == 0) {
4     motor_count = 1;
5   }
6   // if test has not started try to start it
7   if (!ap.motor_test) {
8     if (!mavlink_motor_test_check(chan, throttle_type != 1)) {
9       return MAV_RESULT_FAILED;
10    } else {
11      ap.motor_test = true;
12      if (!motors->armed()) {
13
14        //Original Code for motor arm
15        //init_rc_out();
16        //enable_motor_output();
17        //motors->armed(true);
18        //---------------------------------------//
19
20        //Newer Code for motor arm
21        //motor arm using srv library - arm and enable motors; for
              more info see Arducopter/esc_calibration.cpp
22        if (motors->get_pwm_type() >= AP_Motors::PWM_TYPE_ONESHOT){
23          motors->set_update_rate(g.rc_speed);
24        } else {
25          motors->set_update_rate(50);
```

```
        }
        motors->armed(true);
        SRV_Channels::enable_by_mask(motors->get_motor_mask());
        hal.util->set_soft_armed(true);
        //-----------------------//
        ...
```

2. **Compile Code:** Use Cygwin Terminal and execute the following command.

   ```
   $ ./waf configure --board PixHawk1
   $ ./waf build copter
   ```

3. **Burn Code:** Connect the PX4 with your Windows System using USB port. Open Mission Planner. Click Initial Setup tab and select Install Firmware. Click Load custom firmware on the bottom right of the window and select the newest arducopter build (usually in build/<yourConfiguredBoard>/bin folder). Follow the instructions during installation.

4. **Execute Test:** Follow the step 4 of above instructed alternative to execute the commands on MAVProxy GCS.

We added another test where we updated the source code to test all four motors with the motortest command. We set the pointer motors->output_test_seq(i, pwm) where i->1 to 4 in **void** Copter::motor_test_output() method in file *ArduCopter/moter_test.cpp*.

8.2 Printing 'Hello New Flight Mode' using MavProxy on GCS

The ardupilot libraries are written in C++ program and hence changing any dynamics of drone system would require updating the cpp code, building and loading the firmware on on-board PX4. This test verifies two aspects

1. Loading of firmware on PX4

2. Testing of addition of a new flight mode

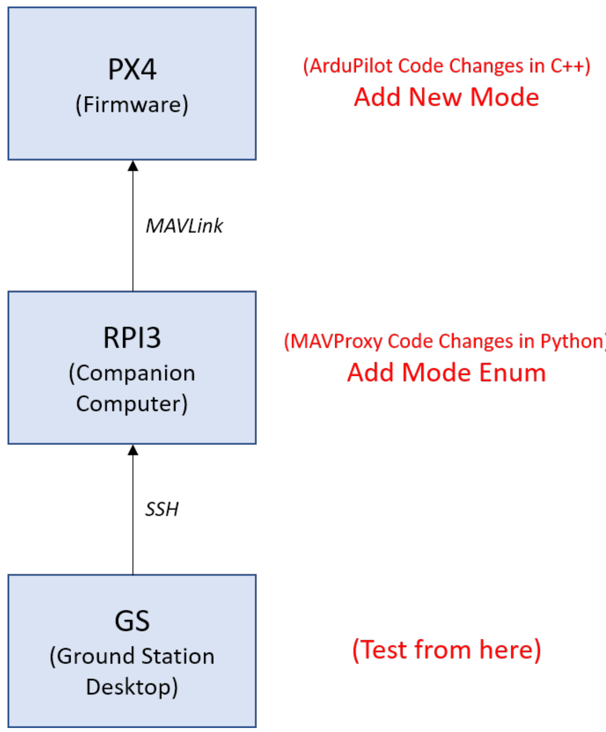

Figure 8.3. Data Flow in *New Mode* Test.

Figure 8.3 explains the high level updates we make to perform this test. We are going to add a new mode on ArduPilot firmware and the corresponding enumeration changes on the companion computer. The test is done through the ground station CLI which is also in our Desktop. Following were the steps undertaken to create a new flight mode in ArduPilot, adding the flight mode in MAVProxy console and executing test commands:

1. **Adding New Flight Mode:** Create a new file *ArduCopter/mode_altholdsonarnogps.cpp* to the ArduPilot library.

```
#include "Copter.h"
//Init and run calls for guided_nogps flight mode
bool Copter::ModeAltholdSonarNoGps::init(bool ignore_checks)
{
  hal.uartC->printf("Hello New Mode\n");
  // start in angle control mode
```

```
  Copter::ModeGuided::angle_control_start();
  return true;
}
// guided_run - runs the guided controller should be called at 100hz or more
void Copter::ModeAltholdSonarNoGps::run()
{
  // run angle controller
  Copter::ModeGuided::angle_control_run();
}
```

2. **Update header file:** Add the following code in *ArduCopter/copter.h* file:

```
#if MODE_ALTHOLD_SONAR_NOGPS_ENABLED == ENABLED
  ModeAltholdSonarNoGps mode_althold_sonar_nogps;
  float groundSonarDistance;
#endif
```

Add the following code in *ArduCopter/config.h* file as well:

```
// ALTHOLD_Sonar_NoGPS - no baro no gps for indoor position
#ifndef MODE_ALTHOLD_SONAR_NOGPS_ENABLED
  # define MODE_ALTHOLD_SONAR_NOGPS_ENABLED ENABLED
#endif
```

Also, we need to add a new mode class in *ArduCopter/mode.cpp* file:

```
class ModeAltholdSonarNoGps : public ModeGuided {
public: // inherit constructor
using Copter::ModeGuided::Mode;
bool init(bool ignore_checks) override;
void run() override;
bool requires_GPS() const override { return false; }
bool has_manual_throttle() const override { return false; }
bool allows_arming(bool from_gcs) const override { return true; }
```

```
bool is_autopilot() const override { return true; }

protected:
const char *name() const override {return "ALT_HOLD_SONAR_NOGPS"; }
const char *name4() const override {return "AHSN"; }
};
```

Reference the above class to the select mode switch statement in the same file *ArduCopter/-mode.cpp*:

```
// return the static controller object corresponding to supplied mode
Copter::Mode *Copter::mode_from_mode_num(const uint8_t mode)
{
  Copter::Mode *ret = nullptr;

  switch (mode) {
    ...
    #if MODE_ALTHOLD_SONAR_NOGPS_ENABLED == ENABLED
      case ALTHOLD_SONAR_NOGPS:
      ret = &mode_althold_sonar_nogps;
      break;
    #endif
    ...
}
```

3. **Compile Code:** Use Cygwin Terminal and execute the following command.

```
$ ./waf configure --board PixHawk1
$ ./waf build copter
```

4. **Burn Code:** Connect the PX4 with your Windows System using USB port. Open Mission Planner. Click `Initial Setup` tab and select `Install Firmware`. Click `Load custom firmware` on the bottom right of the window and select the newest arducopter

build (usually in build/<yourConfiguredBoard>/bin folder). Follow the instructions during installation. In cases when the software is not able to find version changes, use Force Bootloader first and then click Load custom firmware.

5. **Updating MavProxy:** Execute the following commands in Command prompt on Desktop:

```
$ ssh pi@<IP Address Here>
$ pip remove mavproxy
$ pip remove pymavlink
```

Change the following enumeration *mode_mapping_acm* in *pymavlink/mavutil.py* file:

```
mode_mapping_acm = {
  0 : 'STABILIZE',
  1 : 'ACRO',
  2 : 'ALT_HOLD',
  3 : 'AUTO',
  4 : 'GUIDED',
  5 : 'LOITER',
  6 : 'RTL',
  7 : 'CIRCLE',
  8 : 'POSITION',
  9 : 'LAND',
  10 : 'OF_LOITER',
  11 : 'DRIFT',
  13 : 'SPORT',
  14 : 'FLIP',
  15 : 'AUTOTUNE',
  16 : 'POSHOLD',
  17 : 'BRAKE',
  18 : 'THROW',
  19 : 'AVOID_ADSB',
  20 : 'GUIDED_NOGPS',
  21 : 'SMART_RTL',
```

```
    22 : 'FLOWHOLD',
    23 : 'FOLLOW',
    25 : 'ALTHOLD_SONAR_NOGPS'
}
```

6. **Reinstalling pymavlink and mavproxy:** Execute the following command:

```
$ pip remove mavproxy
$ pip remove pymavlink
$ cd ./pymavlink
$ sudo python setup.py build
$ sudo python setup.py install
$ cd ./MAVProxy
$ sudo python setup.py build
$ sudo python setup.py install
```

7. Executing the following test commands and the figure 8.4 shows the output on console:

```
$ sudo mavproxy.py
$ mode 25 or mode ALTHOLD_SONAR_NOGPS
```

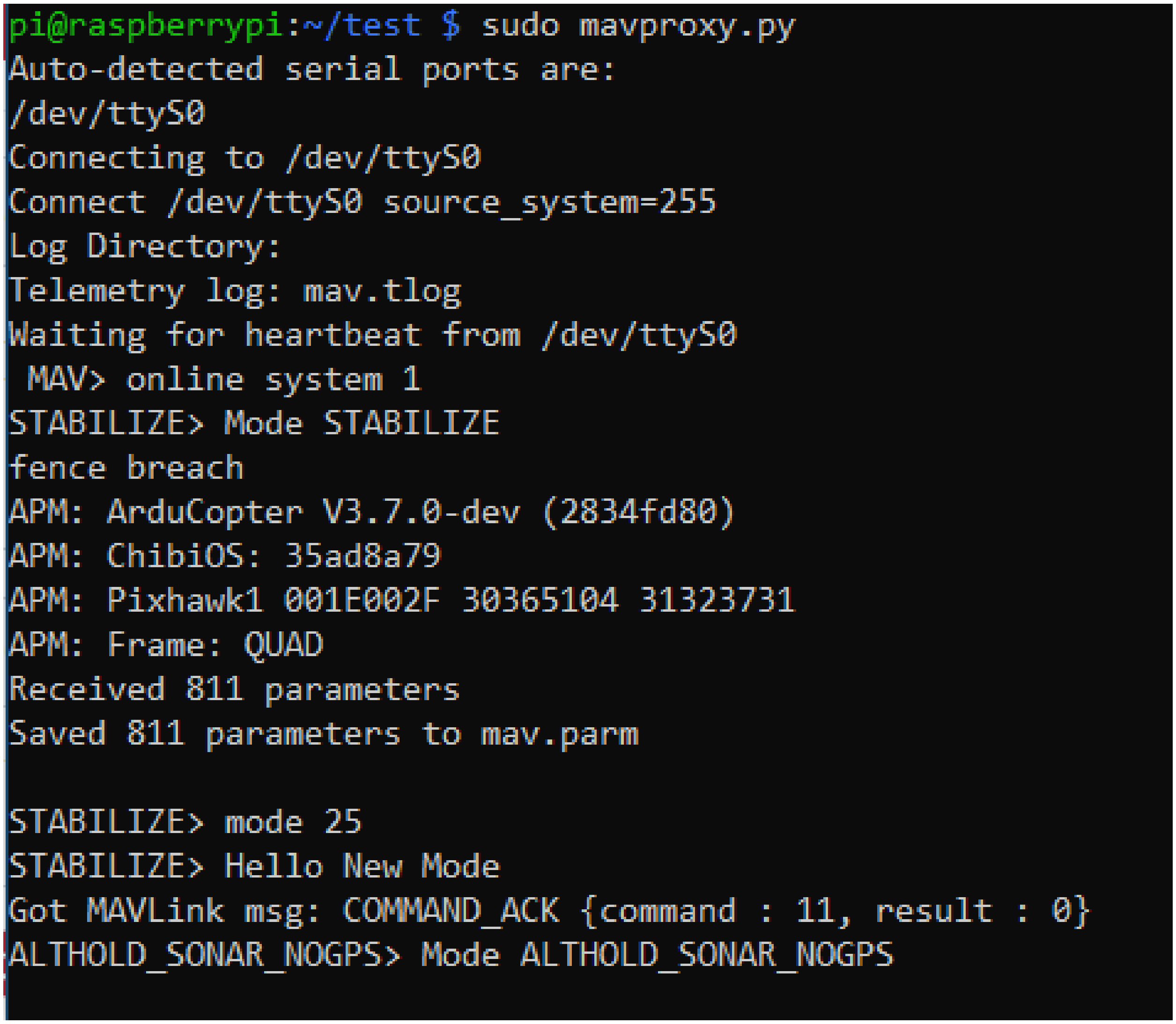

Figure 8.4. Printing 'Hello New Mode' on MavProxy terminal.

8.3 Testing Sonar Altitude Measurements using MavProxy on GCS

As stated earlier, we are using Sonar to get altitude data for drone. Although the sonar sensor will not be accurate outdoors, it is suitable for indoor use as it is accurate up to 4 meters. For even higher accuracy, more of such ultrasonic sonar sensor could be used, especially facing upward to calculate distance from roof. This integration test verifies three aspects:

1. Correctness of Sonar readings

2. Correctness of Send socket (written in Python based MAVProxy) and sending of sonar read-

ings on Mavlink Communication Protocol

3. Correctness of Receive socket (written in C++ based ardupilot) and receiving of sonar readings on Mavlink Communication Protocol

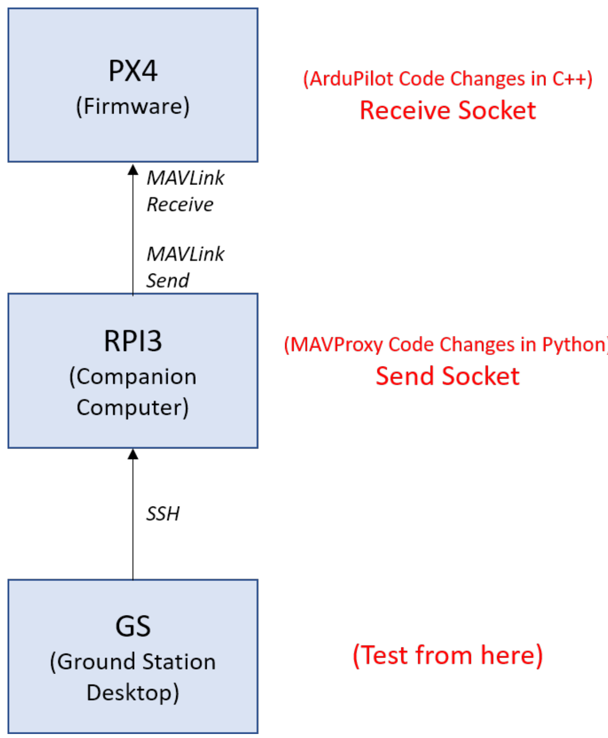

Figure 8.5. Data Flow in *Sonar Integration* Test.

Figure 8.5 explains the updates we make to perform this test. We are going to create a send socket on companion computer and a receive socket on ArduPilot firmware. The test is done through the ground station CLI which is also in our Desktop. A sample python code for reading of sonar sensor is located in *home/sonar* folder of RPI3. Execute the python file *test_sonar.py* to test correctness of sensor measurements.

```
import RPi.GPIO as GPIO
import time
try:
  GPIO.setmode(GPIO.BOARD)
  PIN_TRIGGER = 7
  PIN_ECHO = 11
```

```
  GPIO.setup(PIN_TRIGGER, GPIO.OUT)
  GPIO.setup(PIN_ECHO, GPIO.IN)
  GPIO.output(PIN_TRIGGER, GPIO.LOW)

  print "Waiting for sensor to settle"
  time.sleep(2)
  print "Calculating distance"
  GPIO.output(PIN_TRIGGER, GPIO.HIGH)
  time.sleep(0.00001)
  GPIO.output(PIN_TRIGGER, GPIO.LOW)

  while GPIO.input(PIN_ECHO)==0:
    pulse_start_time = time.time()
  while GPIO.input(PIN_ECHO)==1:
    pulse_end_time = time.time()

  pulse_duration = pulse_end_time - pulse_start_time
  distance = round(pulse_duration * 17150, 2)
  print "Distance:",distance,"cm"
finally:
  GPIO.cleanup()
```

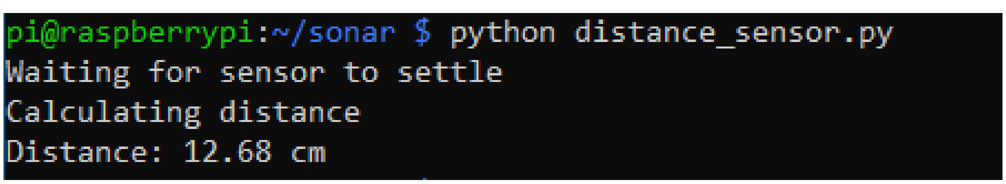

Figure 8.6. Testing sonar measurement using a simple Python Code.

To test send and receive socket, first we need to create a new message in MavLink protocol. Execute the following steps to create a new message:

1. **Update the Message Definition:** Open the xml file *./pymavlink/ message_definition /ardupilotmega.xml* and add an another *message* node in parent *messages* node to the xml definition:

```xml
<!-- SONAR and SLAM Based messages -->
<message name="SONAR_GROUND_DISTANCE" id="11050">
  <description>send the ground distance (measured via sonar attached to GS) in cm</description>
  <field name="time_usec" type="uint64_t">Timestamp (microseconds since UNIX epoch or microseconds since system boot)</field>
  <field name="ground_distance" type="float">Ground distance in centimeters. Positive value: distance known. Negative value: Unknown distance</field>
</message>
```

2. **Uninstall mavproxy and pymavlink**: Its possible multiple copy of the libraries are installed. To resolve this, keep uninstalling until we get a message no libraries available for uninstall. Execute the following commands to uninstall.

   ```
   $ pip remove mavproxy
   $ pip remove pymavlink
   ```

3. **Install pymavlink** Execute the following commands:

   ```
   $ cd ./pymavlink
   $ sudo python setup.py build
   $ sudo python setup.py install
   ```

4. **Verify the Header Files:** Check for the cpp header files (*.h extension*) with the same name as the earlier newly created message at */.local /lib/ Python2.7 /site-packages/* path. Use `pip show pymavlink` to get this installed path. If a *python .egg package* is created, then use unzip command to list the content of the package `unzip -l /path/to/file.egg`. See Section 6.2 and 6.3 for details on library dependencies and mavgen tools.

In order to create a send socket on MAVProxy we need to first create a new module in MAVProxy software and reinstall the mavproxy libraries. Follow these steps:

1. **Create new module:** go to folder location *home/MAVProxy/MAVProxy/modules/* and create a new file and *mavproxy_sonarInput.py*. Copy these lines to the file.

```
'''
support for Sonar_INPUT message
'''
import socket, errno
import json
from pymavlink import mavutil
from MAVProxy.modules.lib import mp_module
import RPi.GPIO as GPIO
import time

class SonarInputModule(mp_module.MPModule):
  def __init__(self, mpstate):
    super(SonarInputModule, self).__init__(mpstate, "SonarInput", "Sonar_INPUT message support")
    self.data = {
    'time_usec' : 0,    # (uint64_t) Timestamp (micros since boot or Unix epoch)
    'ground_distance' : 0,  # (float) ID of the GPS for multiple GPS inputs
    }
    PIN_TRIGGER = 16
    PIN_ECHO = 18
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(PIN_TRIGGER, GPIO.OUT)
    GPIO.setup(PIN_ECHO, GPIO.IN)
    GPIO.output(PIN_TRIGGER, GPIO.LOW)
    print ("Waiting for sonar sensor to settle")
    time.sleep(2)
```

```

def idle_task(self):
  '''called rapidly by mavproxy'''
  PIN_TRIGGER = 16
  PIN_ECHO = 18
  try:
    GPIO.output(PIN_TRIGGER, GPIO.HIGH)
    time.sleep(0.00001)
    GPIO.output(PIN_TRIGGER, GPIO.LOW)
    while GPIO.input(PIN_ECHO)==0:
      pulse_start_time = time.time()
    while GPIO.input(PIN_ECHO)==1:
      pulse_end_time = time.time()
    pulse_duration = pulse_end_time - pulse_start_time
    distanceData = round(pulse_duration * 17150, 2)
    millis = int(round(pulse_end_time * 1000))
  except Exception as e:
    print("Sonar Input Read Failed:", e)

  try:
    self.master.mav.sonar_ground_distance_send(millis, distanceData)
  except Exception as e:
    print("Sonar Input Send Failed:", e)

def init(mpstate):
  '''initialise module'''
  return SonarInputModule(mpstate)
```

2. **Reinstall MAVProxy:** MAVProxy folder execute these commands -

```
$ cd ./mavproxy
$ sudo python setup.py build
$ sudo python setup.py install
```

3. **Test the new module:** execute the following commands. We should get a results on terminal as shown in figure 8.7.

```
$ sudo mavproxy.py
$ module load sonarInput
```

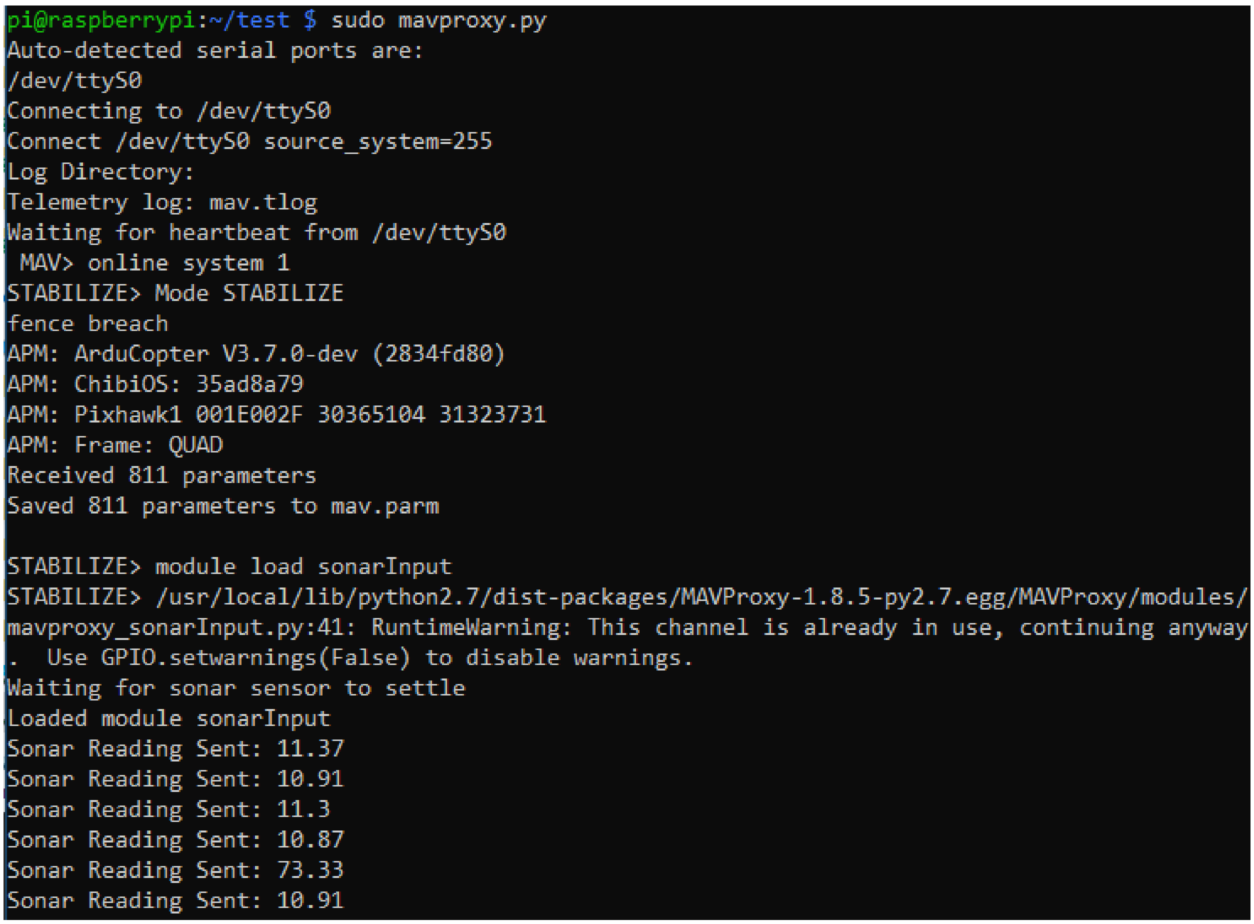

Figure 8.7. Testing sending of sonar measurement.

To create a receive socket on ArduPilot, we need to update the *handle_message()* function in ardupilot c++ based source code. Follow these steps:

1. **Create new packet decoder:** Copy a new switch statement to *void GCS_MAVLINK_Copter :: handleMessage(mavlink_message_t* msg)* function in *ardupilot/ArduCopter/GCS_MavLink.cpp* as shown below.

```
void GCS_MAVLINK_Copter::handleMessage(mavlink_message_t* msg)
{
  switch (msg->msgid) {
    ...
    #if MODE_ALTHOLD_SONAR_NOGPS_ENABLED == ENABLED
      case MAVLINK_MSG_ID_SONAR_GROUND_DISTANCE:
      {
        sonar_altitude = GCS_MAVLINK_Copter::sonar_ground_distance_rcv(msg);
        hal.uartC->printf("Sonar Reading Received: %f\n", sonar_altitude);
        break;
      }
    #endif
    ...
}
```

2. **Compile Code:** Use Cygwin Terminal and execute the following command.

   `$ ./waf build copter`

3. **Burn Code:** Connect the PX4 with your Windows System using USB port. Open Mission Planner. Click `Initial Setup` tab and select `Install Firmware`. Click `Load custom firmware` on the bottom right of the window and select the newest arducopter build (usually in `build/PixHawk1/bin` folder). Follow the instructions during installation.

4. Test the new decoder: Execute the commands below. We should get a results on terminal as shown in figure 8.8.

   `$ sudo mavproxy.py`

   `$ module load sonarInput`

```
pi@raspberrypi:~/test $ sudo mavproxy.py
Auto-detected serial ports are:
/dev/ttyS0
Connecting to /dev/ttyS0
Connect /dev/ttyS0 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from /dev/ttyS0
 MAV> online system 1
STABILIZE> Mode STABILIZE
fence breach
APM: ArduCopter V3.7.0-dev (2834fd80)
APM: ChibiOS: 35ad8a79
APM: Pixhawk1 001E002F 30365104 31323731
APM: Frame: QUAD
Received 811 parameters
Saved 811 parameters to mav.parm

STABILIZE> module load sonarInput
STABILIZE> /usr/local/lib/python2.7/dist-packages/MAVProxy-1.8.5-py2.7.egg/MAVProxy/modules/
mavproxy_sonarInput.py:41: RuntimeWarning: This channel is already in use, continuing anyway
.  Use GPIO.setwarnings(False) to disable warnings.
Waiting for sonar sensor to settle
Loaded module sonarInput
Sonar Reading Sent: 11.35
Sonar Reading Sent: 11.32
Sonar Reading Received: 11.35000
Sonar Reading Sent: 11.29
Sonar Reading Received: 11.32000
Sonar Reading Sent: 11.32
Sonar Reading Received: Sonar Reading Sent: 10.91
11.29000
Sonar Reading Sent: 11.3
```

Figure 8.8. Testing receiving of sonar measurement.

CHAPTER 9

CHALLENGES

We faced many challenges during the study pertaining to power management, understanding available open source software system, and code development.

Apart from drone technology and its control system, an important aspect of a drone is its battery system. It is important to understand how much power is delivered to the drone and its component for smoother development process. We faced a LiPo battery damage as the battery swelled-up due to incorrect charging procedures. We were charging the battery for 15 to 20 min, instead we should have charged it for 30 to 60 min for a full charge. Our battery charger pack has inbuilt switch program which beeps and disconnects the battery when it is fully charged. If the LiPo battery is discharged below 3.2V per cell (in our case, a total of 12.8V for 4 cells), it will cause a permanent damage to battery maximum capacity. We also powered the PX4 Autopilot controller from two sources — battery as well as the USB. This too discharged and damaged the battery significantly. As a rule of thumb, any embedded system should be charged from one power source only. It is possible, due to its smaller form factor or economic factor, the embedded system does not have an inbuilt power management module to support higher or parallel power ratings.

One of the major challenges of the project was to understand the open source software which we were using. The communication protocol MAVLink is hidden and works under application layers and, hence, during the development process we had to deep dive the underlying networking layer of the system. To articulate and test our networking changes, we had to use the MAVProxy GCS on application layer, which itself had its learning curve. For the firmware development, there was overwhelming amount of code in ArduPilot to investigate, around more than 150 MB of code or more than 250K lines of code. We had to inspect the firmware's open and closed loops and Booleans flags for the execution of functions on a case-by-case basis. Setting up

Eclipse-Cygwin IDE was also demanding as there were permission errors, code analysis errors, and the code took a considerable compilation time.

For code development, we found that testing was cumbersome. We had to use the process of build, burn and print. Nowadays, developers use high level languages and testing frameworks, and therefore could find such techniques arduous. As stated earlier, the vast amount of firmware's code makes it challenging to remove module/sensor libraries from the system. There is a possibility, that forcing GPS and barometer disable causes side-effects or issues with the open loop which repetitively executes methods. If we remove one method, other methods may not work as expected. Also, due to high dependence on the GPS, disabling GPS caused arming issues. We had to set an additional Boolean *soft_arm* apart from regular *arm* flag to power the motors. We also calibrated ESC by reusing the calibration method available in the firmware source code.

Other minor issues we faced were related to fragile hardware. We had a broken USB cable which was powering the system but was unable to send data, and we ended up spending some time on driver installation on Windows OS. Also, every time we test the system, we make sure of open or short connections that can cause test fails or smoking of wires, respectively. Also we used a higher Baud Rate to communicate to PX4 and realized that the firmware is rated to work at lower values.

CHAPTER 10

POTENTIAL DIRECTIONS

As every research study does, this study on achieving higher autonomy also faced time constraints. Even though we were able to draft and test a software architecture and a development pattern to add more sensors and a control system corresponding to it, fully disabling existing barometer and GPS sensor are still on board. As stated earlier, removing these libraries can cause side effects on other methods and hence, we need to go through each removed code to check variable dependence on other classes/methods and appropriately handle each scenario. In this case, we need to bite the bullet as there are no silver bullets.

Another parameter we need to look is on indoor localization improvements. This could be done by adding more sensors, specially the available SLAM (Simultaneous Localization and Mapping) system in our lab. The stabilization of the drone can be improved by interpreting and analyzing the 360-point dataset generated per time frame from the SLAM system. The stabilization could also be improved by researching granularity and delay analysis of sonar data. This would be a comparison and fine tuning between streaming and individual message.

This approach also gives a platform for executing autonomous operations from ground station by enabled input/input-stream. Future development could be done to integrate machine learning and artificial intelligence algorithms on input data and sent it through the ground station.

There is also a scope to implement SITL (Software in the loop) for simulated drone testing without enabling any hardware. SITL works on x86 compatible system and hence, with current architecture it can be only installed and used on desktop module (Dev Box/Test Box).

We also could not try on the inbuilt logging mechanism provided by the ArduPilot team. We expect this would get us significant help in testing and debugging.

This study also gives room to multi-developer/team development on the same drone system due to its support modular changes. Future iteration of this study could move towards a partial SOA (Service-Oriented Architecture) implementation to streamline software development.

CHAPTER 11

CONCLUSIONS

This thesis describes a new modular architecture for developing, deploying, and testing a drone system. The thesis gives an in-depth instruction on how to set-up and verify the needed software stack. This new architecture improves the autonomy of the drone system by removing the radio controller transmitter and controlling it remotely over Wi-Fi system. We also integrated the sonar sensor technology into the firmware using customized messages over the MAVLink communication network. Overall, we have successfully expanded an existing Drone System at the Heterogeneous Systems Research Lab in the areas of software implementation and integration.

BIBLIOGRAPHY

BIBLIOGRAPHY

3D Robotics (2019a), About DroneKit, http://python.dronekit.io/about/overview.html, online; accessed 01 March 2019.

3D Robotics (2019b), Optional hardware, http://ardupilot.org/copter/docs/common-optional-hardware.html, online; accessed 29 March 2019.

ArduPilot (2016a), Software, http://ardupilot.org/about, online; accessed 04 April 2019.

ArduPilot (2016b), Mavproxy, http://ardupilot.github.io/MAVProxy/html/index.html, online; accessed 04 April 2019.

ArduPilot Dev Team (2019a), Setting up the waf Build Environment on Windows using Cygwin, http://ardupilot.org/dev/docs/building-setup-windows-cygwin.html, online; accessed 01 February 2019.

ArduPilot Dev Team (2019b), Learning the ArduPilot Codebase, http://ardupilot.org/dev/docs/learning-the-ardupilot-codebase.html, online; accessed 25 March 2019.

ArduPilot Dev Team (2019c), Communicating with Raspberry Pi via MAVLink, http://ardupilot.org/dev/docs/raspberry-pi-via-mavlink.html, online; accessed 01 January 2019.

ArduPilot Dev Team (2019d), Setup eclipse on windows, http://ardupilot.org/dev/docs/building-setup-windows-eclipse.html, online; accessed 06 April 2019.

ArduPilot Dev Team (2019e), Flight modes, http://ardupilot.org/copter/docs/flight-modes.html, online; accessed 04 April 2019.

ArduPilot Dev Team (2019f), License (gplv3), http://ardupilot.org/dev/docs/license-gplv3.html, online; accessed 04 April 2019.

ArduPilot Dev Team (2019g), Installing mission planner (windows), http://ardupilot.org/planner/docs/mission-planner-installation.html, online; accessed 05 April 2019.

Balena (2019), Etcher, https://www.balena.io/etcher, software; version 1.5.19.

Batard, P. (2019), Rufus, https://rufus.ie/, software; version 3.5.

Canis, B. (2015), Unmanned aircraft systems (uas): Commercial outlook for a new industry.

Cygwin Authors (2019), Cygwin, https://www.cygwin.com/install.html, software; version 3.0.5.

davids5 (2017), Fmuv2 hardware identification some mpu9250 fixes, mpu9250 deprecated on pxmini 7618, https://github.com/PX4/Firmware/pull/7618, online; accessed 02 April 2019.

Davis, T. (2019), Win32 disk imager, https://sourceforge.net/projects/win32diskimager/, software; version 1.0.

Dedes, G., and A. G. Dempster (2005), Indoor gps positioning-challenges and opportunities, in *VTC-2005-Fall. 2005 IEEE 62nd Vehicular Technology Conference, 2005.*, vol. 1, pp. 412–415, Citeseer.

DJI (2014), E310 user manual v1.0, http://dl.djicdn.com/downloads/e310/en/E310_User_Manual_v1.0_en.pdf, online; accessed 02 April 2019.

Dronecode Project (2018), Mavlink developer guide, https://mavlink.io/en/, online; accessed 04 April 2019.

Gossett, H. (2018), Building an Autonomous Indoor Drone System, Master's thesis, Department of Computer and Information Science, The University of Mississippi.

Hawkins, M. (2018), Simple guide to the raspberry pi gpio header and pins, https://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/, online; accessed 05 April 2019.

Hock-Chuan, C. (2013), Eclipse 4.3 (kepler) for c/c++ programming: How to install eclipse cdt 8.2 and get started, https://www3.ntu.edu.sg/home/ehchua/programming/howto/EclipseCpp_HowTo.html, online; accessed 06 April 2019.

Li, B., B. Harvey, and T. Gallagher (2013), Using barometers to determine the height for indoor positioning, in *International Conference on Indoor Positioning and Indoor Navigation*, pp. 1–7, IEEE.

Parnas, D. L. (1972), On the criteria to be used in decomposing systems into modules, *Commun. ACM*, 15(12), 1053–1058, doi:10.1145/361598.361623.

Proud, R. W., J. J. Hart, and R. B. Mrozinski (2003), Methods for determining the level of autonomy to design into a human spaceflight vehicle: a function specific approach, *Tech. rep.*, NATIONAL AERONAUTICS AND SPACE ADMINISTRATION HOUSTON TX LYNDON B JOHNSON .

PX4 Dev Team (2019), Pixhawk mini (discontinued), https://docs.px4.io/en/flight_controller/pixhawk_mini.html, online; accessed 02 April 2019.

QuadcopterGarage.com (2018), Dji flame wheel f450 review, http://quadcoptergarage.com/dji-flame-wheel-f450-review, online; accessed 02 April 2019.

Raspberry Pi Foundation (2018), Debian stretch with raspberry pi desktop, https://www.raspberrypi.org/downloads/raspberry-pi-desktop/, software; version November 2018.

Raspberry Pi Foundation (2019a), The raspberry pi uarts, https://www.raspberrypi.org/documentation/configuration/uart.md, online; accessed 05 April 2019.

Raspberry Pi Foundation (2019b), Gpio, https://www.raspberrypi.org/documentation/usage/gpio/, online; accessed 05 April 2019.

SD Association (2019), Sd memory card formatter, https://www.sdcard.org/downloads/formatter/, software; version 5.0.1.

Sopasakis, P. (2017), Attitude control and autonomous navigation of quadcopters, https://alphaville.github.io/files/quadcopter-slides.pdf, online presentation; accessed 24 March 2019.

Watts, A. C., V. G. Ambrosia, and E. A. Hinkley (2012), Unmanned aircraft systems in remote sensing and scientific research: Classification and considerations of use, *Remote Sensing*, 4(6), 1671–1692.

VITA

SHOBHAN SINGH

shobhansingh1@gmail.com

PROFESSIONAL SUMMARY

- IT experience for more than 3 years of developing applications using Microsoft .NET Technologies and Microsoft BizTalk Server Technologies.
- Experience in Design, Development, Integration, Implementation and provided solutions throughout all the phases of Software Development Life Cycle (SDLC)
- Solid .NET experience on Backend Development; and implementation of Backend Business processes and Server-Side Integration techniques.
- Experience in implementing Service Oriented Architecture (SOA) based solutions using .NET/ BizTalk Server Framework.
- Solid experience in Consuming and Deploying WCF and Web services in .NET Framework.
- Experience in Deploying Builds to Servers and Configure them as per the requirement and performing Build verification tests.
- Hands on experience writing, testing and implementation of Queries, Stored Procedures, Triggers and Functions at database level using MS SQL Server and Oracle RDBMS.
- Highly motivated with excellent analytical and communication skills, ability & enthusiasm to learn new systems and technologies.

EDUCATION

University of Mississippi, University, MS *May 2019*
Master of Science in Engineering Science
Emphasis - Computer Science

TECHNICAL SUMMARY

Programming Languages	.NET C#, Java, SQL, C++/C, Python
Web Technologies	WCF/Web Services, XML, JSONS, ADO.NET, WSDL, XSLT, XPath, IIS
Markup Languages	XML, XSL, XAML, HTML, CSS
Databases	MS SQL Server 2016/2014, Oracle 12c/11g
Middleware Technologies	Microsoft BizTalk, BRE, BAM

RELEVANT WORK EXPERIENCE

Blue Cross Blue Shield of Arizona, Phoenix AZ *Jun 2017 – Aug 2017*
Contract to IT Echelon Corporation, Princeton NJ
Developer

- Responsible for Integration, Development and Testing of Medicare System for SOA Team.
- Created System Tracking Points for EDI 837 (Claim Submission) and corresponding 999 (Response)

submitted by Clients.

- Used SQL Composite Operations to reduce connection time and transmit time to SQL Servers.
- Created Pipeline Component to convert input Excel files from Clients to desired xml format using maps and OleDBDataAdapter libraries
- Created Batch Process using SQL Adapter and Pipeline Components to insert and query records to and from SQL Tables.
- **Environment:** BizTalk Server 2010, Windows Server 2008, Visual Studio 2010, C# (.NET Framework 4.4, SQL Server 2008 R2, EDI/XML/Canonical Schemas, Development Model – Scrum.

Hewlett Packard Enterprise, Newark DE *Jun 2015 – Apr 2017*

Contract to Infologitech Inc, Princeton NJ and IT Echelon Corporation, Princeton NJ

Middleware/Backend Developer

- Responsible for Integration, Development and Testing of Delaware Medicaid Enterprise System (DMES) for SOA Subsystem/Team
- Successfully implemented Backend processes for Customer Service, Pharmacy Claims, Claims, Drug (Health History), Provider, Member, Authorization and Shared functional subsystem.
- Integration of Functional Sub-System like Claims, Financial, Trading Partner, Provider, Member, etc. with UI (Portal) using BizTalk Server 2013R2 using WCF/Web Services.
- Development of Backend codes using .NET/Entity Framework to enable Logical/Business Processing of DMES PHI/PII/Internal data.
- Integration of External Systems/Technologies like Lexis-Nexis (Background Verification), Corticon (Rules Engine), K2 (Workflow) Services, PS2 Processor (Authorization Engine), SOAP Server (Claims Engine) with DMES through ESB/Web Service Technology.
- Implementing Lexis Nexis and Address Doctor Rules for verification during Provider Enrollment and Revalidation based on the responses from these third-party services.
- Created IIS Scripts and MS Excel Reports using LogParser and Excel VBA Macros.
- Created Scheduler task application and assigning privileges to process batch processes.
- Created additional Data Processing techniques using third party libraries including Aspose and Ionic.zip libraries.
- Used Logging Mechanism for Applications using Event Log and log4net libraries.
- Implemented standard security protocols for Backend/Middleware Solution including basicHTTP and WsHTTP protocols.
- Integration techniques involving BizTalk WCF Wizard and using Orchestrations/Schema to Expose WCF services and URL to consume WCF Services.
- Used Oracle SQL Developer to verify testing, create test data and user privileged tables, and executed and incorporated complicated SQL queries in the Backend processes; and Installation and Documentation of Oracle 11g.
- **Environment:** BizTalk Server 2013R2, Windows Server 2008, Visual Studio 2015/2013/2012, C# (.NET Framework 4.5/4.5.1), SQL Server 2008, Oracle DB 11g, SQL Developer 2015, EDI/XML/Canonical Schemas, IIS 7.5, TFS 2010, Development Model - Waterfall.