First-In Risk Evaluation System (F.I.R.E. System)

UAV Control and Autonomous Flight

Final Report

Author:
Erik C.M. Johnson

Supervisor:
Alan Steele, Ph.D.

Sunday 9th April, 2017

A fourth-year capstone project with:
Fizza Ahmad Sheikh (100892415)
Alok Deshpande (100890102)
Calla McClelland (100884066)
Mohamed Jaber (100878656)
Ann Gunaratnam (100885604)

Abstract

Recent advances in robotic systems have opened the door for the pursuit of novel applications. This report presents the development of an autopilot system as part of a fourth-year capstone project at Carleton University. The objective of the project was the realization of a quadcopter UAV system for application to fire response operations.

The autopilot system designed allows for high-level flight control so that no operator is occupied. This high-level flight control can be overridden through a manual control safety subsystem. The autopilot system exposes access to GPS, accelerometer, and magnetometer data for transmission to the base station.

The autopilot system still requires some software to be written and tested for high-level flight control to be ready for final integration. A flight test also remains for the completed quadcopter UAV system. Future work with photogrammetric processing of received images at the base station would benefit this project in terms of data visualization.

Contents

List of Figures

List of Tables

List of Abbreviations

Analytic Hierarchy Process	AHP
ArduPilotMega	APM
Carleton Undergraduate Engineering Students' Equipment Fund	CUESEF
Controller Area Network	CAN
Digital Elevation Model	DEM
Electronic Speed Controller	ESC
General Public License	GPL
Global Positioning System	GPS
GNU's Not Unix	GNU
Industrial, Scientific, and Medical	ISM
Inertial Measurement Unit	IMU
Inter-Integrated Circuit	I^2C
Lesser General Public License	LGPL
Micro-Aerial Vehicle	MAV
Multiple Criteria Decision Analysis	MCDA
Open Systems Interconnection	OSI
Operating System	OS
Portable Operating System Interchange	POSIX
Printed Circuit Board	PCB
Pull Request	PR
Pulse Position Modulation	PPM
Pulse Width Modulation	PWM
Radio Frequency	RF
Real Time Operating System	RTOS
Robot Operating System	ROS
Serial Peripheral Interface	SPI
Single Board Computer	SBC
Universal Asynchronous Receiver-Transmitter	UART
Unmanned (or Uninhabited) Aerial Vehicle	UAV

1 Introduction

A convergence of technologies has led to the realization of novel robotic systems. This project seeks to address the deficiency of information available to firefighters at the scene of a fire. We have proposed a quadcopter unmanned aerial vehicle (UAV) system that uses a combination of visual and thermal imagery along with data from other sensors to provide firefighters with a new perspective on the scene of a fire. Concept sketches illustrating this idea are shown in figure 1.

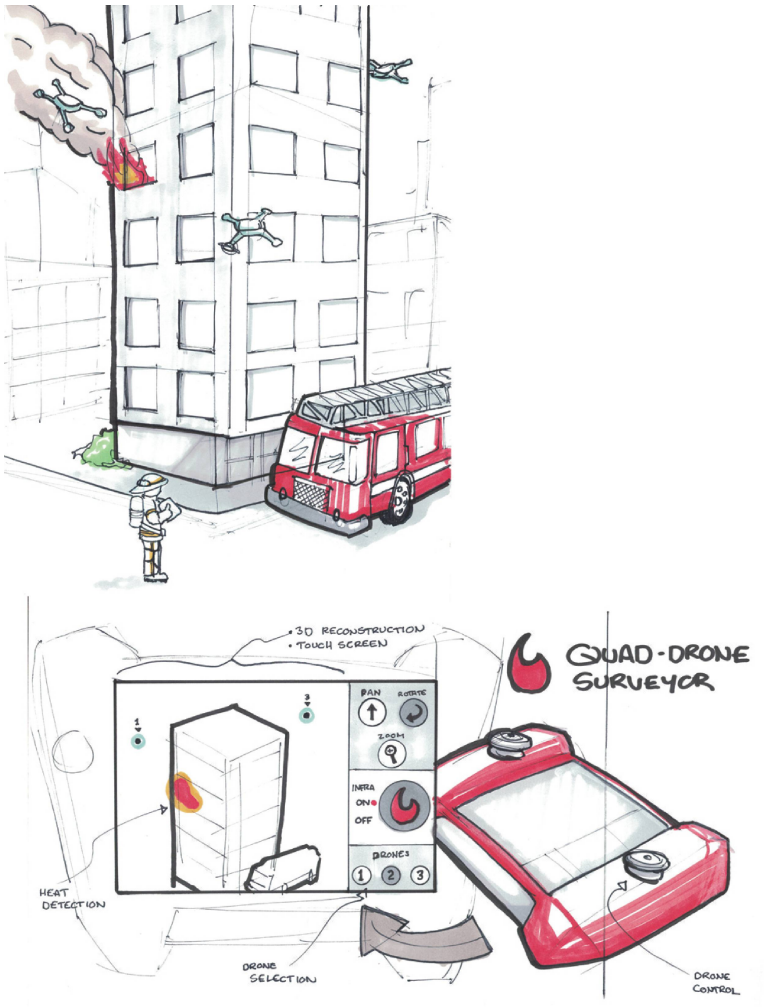


Figure 1: Concept sketch showing system in operation at the scene of a fire (sketch by Maddy DeRueda)

This fourth-year capstone project has made meaningful progress towards the application of a quadcopter UAV to fire response. The objective for the system is to provide firefighters with actionable information that they might not be able to gain without employing UAVs. This project is administered by the Department of Electronics at Carleton University in Ottawa, Canada. This project was supervised by Prof. Alan Steele and completed in conjunction with Alok Deshpande, Fizza Ahmad Sheikh, Ann Gunaratnam, Mohamed Jaber, and Calla McClelland.

This report will present an overview of the system, the objectives of the project as proposed and then present the autopilot system designed to meet these objectives. The autopilot system development involved selecting a suitable autopilot, designing a software architecture, and testing that it would be able to meet all the required objectives. Potential future work towards the improvement of the system is also given.

1.1 Full System Overview

The system diagram with the area of this report identified in blue is shown in figure 2. The system is divided into two separate parts: the UAV and the base station (a PC running the Ubuntu OS). For further information on other components identified in this system diagram refer to table 1 for the individual responsible for that component.

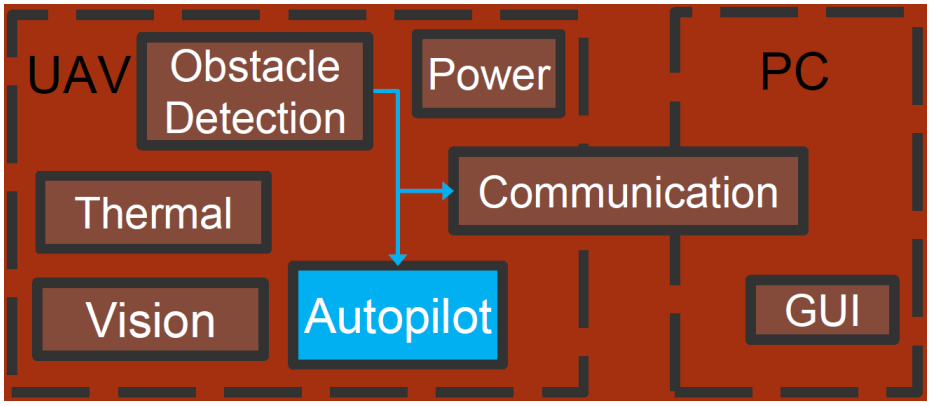


Figure 2: The project system diagram with the area of this report identified in blue

Section	Individual
Obstacle Detection	Alok Deshpande
Power	Mohamed Jaber
Thermal	Ann Gunaratnam
Autopilot	Erik Johnson
Communication	Fizza Ahmad Sheikh
GUI	Calla McClelland
Vision	Calla & Ann

Table 1: Individuals responsible for each section of the project

2 Project and Personal Objectives

The high-level objective of this project was to demonstrate a proof of concept of a flight-capable UAV system for application to firefighter response. The specific objectives in achieving this goal were to be able to:

- control the UAV's flight path from a GUI,
- collect sensor data for display to firefighters and/or for avoiding obstacles in the flight path, and
- present relevant information to firefighters with a GUI.

The objectives of the UAV control and autonomous flight portion of this project were to develop a system capable of high-level control of the flight of a UAV and acquiring data for transmission to a base station. The high-level control of the flight requires sensors such as accelerometer, gyroscope, magnetometer, and GPS. This data, especially the GPS, is valuable for visualization on the base station. Therefore, access to this data is considered as part of the UAV control and autonomous flight system.

The objectives of the UAV control and autonomous flight portion of the project can be summarized as:

- allowing for emergency landing or manual flight control,
- accepting high-level commands (such as takeoff, waypoint navigation and landing), and
- accessing sensor data (namely GPS, magnetometer and accelerometer).

3 Autopilot System Development

A suitable system architecture had to be selected, designed, and implemented so that the system would be capable of meeting the objectives as specified in section 2. This required the selection of autopilot hardware capable of meeting

these objectives. The selected autopilot hardware then required the design of a software architecture. With this hardware and software system, access to data was demonstrated and a high-level control method was implemented.

3.1 Autopilot Selection

Selection of the autopilot hardware to use as a flight controller was approached rigorously as it dictated many aspects or limitations for the entire system. The requirements of the autopilot hardware were that it must be capable of:

- controlling at least four (4) Electronic Speed Controllers (ESCs),
- interfacing with, or contain in the autopilot hardware, a GPS sensor,
- being controlled with a standard RF controller,
- running an autopilot software, such as ArduPilot, and
- interfacing with, or support operation of, the systems to be designed by other group members.

These requirements are hard requirements and dictated what could be considered for the autopilot hardware. In addition to these hard requirements, it would be beneficial for addressing the limited weight payload if the autopilot could support operation of at least a subset of other functions to be developed by other group members. The autopilots considered were:

- a PXFmini mounted on a Raspberry Pi 0,
- a PXFmini mounted on a Raspberry Pi 3,
- a Pixhawk,
- and an APM2.6.

3.1.1 PXFmini

The PXFmini is a sensor PCB assembly compatible with the Raspberry Pi SBC. It mounts on the 40 pin header of a Raspberry Pi and is compatible with all Raspberry Pi models with the 40 pin header layout (i.e. not either model of the Raspberry Pi 1) [1]. We considered using the PXFmini with the Raspberry Pi 0 and the Raspberry Pi 3, both of which are capable of running the ArduPilot software in a Linux environment configured to be an RTOS. The PXFmini contains most of the sensors required for controlling a UAV; however, it does not include an onboard GPS module [1]. It is necessary to connect a GPS module via a UART interface. An image showing the PXFmini along with external connections is shown in figure 3.

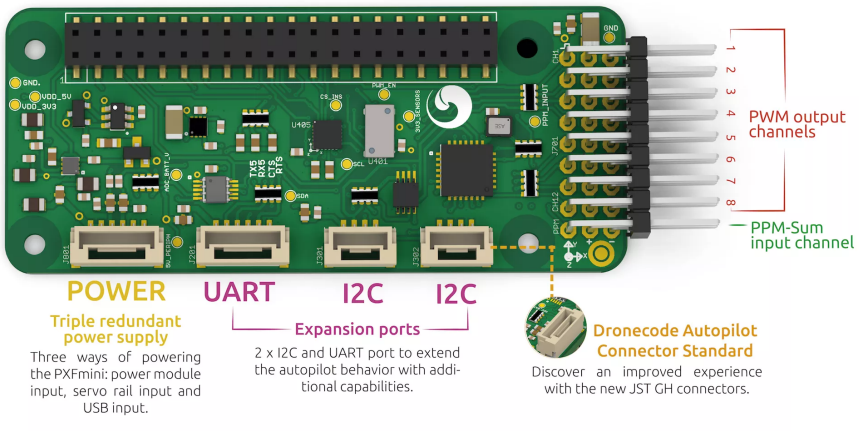


Figure 3: Image of PXFmini board with external connections identified [1]

3.1.2 Pixhawk

The Pixhawk is an open-source, self-contained (not relying on any external computation for flight control) autopilot. The Pixhawk runs a POSIX-compliant RTOS that can use either PX4 or ArduPilot for flight control [2]. The Pixhawk would require an external GPS, but it also exposes many other interfaces such as I^2C, SPI and CANbus [2]. One interesting feature of the Pixhawk is the inclusion of a redundant accelerometer/gyroscope and a failsafe co-processor [2]. An image of the Pixhawk is shown in figure 4.

Figure 4: Image of the Pixhawk autopilot [2]

3.1.3 APM2.6

The APM2.6 is an open-source, self-contained (not relying on any external computation for flight control) autopilot built for use with ArduPilot. The APM2.6 is past end-of-life and no longer supports current builds of ArduPilot (versions > 3.3) [3]. Versions earlier than 3.3 could be used in a simple flight control application; however, significant changes and fixes have been made since version 3.3 [4]. The APM2.6 contains all sensors required for flight, except for the GPS. The APM2.6 would allow for easy editing and reflashing of its code as it runs open-source ArduPilot (version < 3.3) on bare metal (i.e. no OS). A disassembly of the APM2.6 is shown in figure 5.

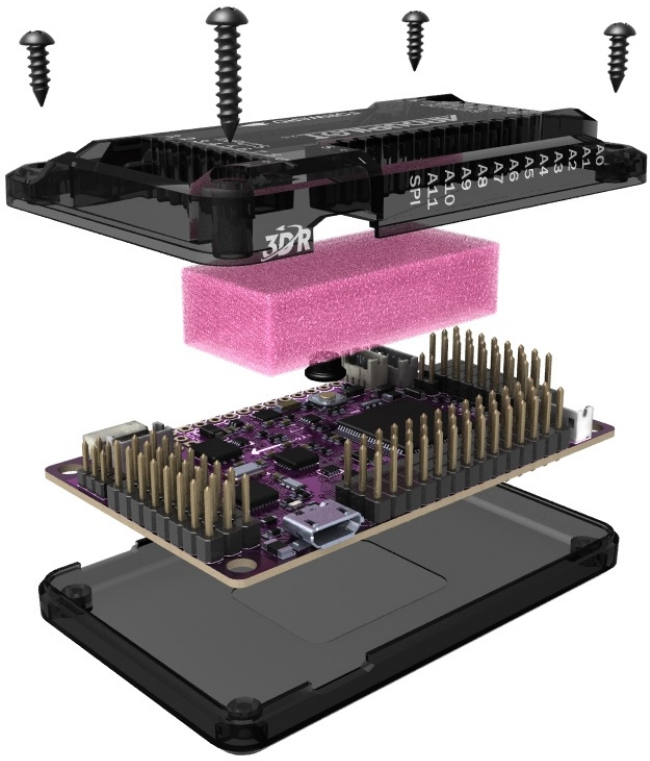


Figure 5: A disassembly showing the APM2.6 [3]

3.1.4 MCDA Methods

Three Multiple Criteria Decision Analysis (MCDA) methods were compared in order to determine the best MCDA method to use for selecting an appropriate autopilot. The three methods considered were: the Pugh method, the Kepner-Tregoe methodology, and the Analytic Hierarchy Process (AHP). These three

MCDA methods were compared in a NASA trade study of decision methodologies and can therefore be considered as mature and accepted MCDA methods [5].

The first MCDA method considered was the Pugh method. This method is a relatively straight forward pro/con method that presents results in an easy to convey table. The Pugh method can consider qualitative factors without assigning them a quantitative value [6]. This method is lacking in accuracy of final decisions [5] and issues have been raised surrounding its ability to converge in iterative applications [6].

The next MCDA method considered was the Kepner-Tregoe methodology. The Kepner-Tregoe methodology is relatively well-known among MCDA methods because of its application during the Apollo 13 disaster [7]. There are two major differences in the Kepner-Tregoe methodology as compared to the Pugh method: a MUSTS/WANTS division of criteria and quantitative consideration for selecting the best option [5]. The MUSTS/WANTS division is not fully applicable to the UAV autopilot selection as the information on decision criteria is relatively static. The MUSTS/WANTS division of criteria supports iterative decision making where the dynamic nature of information may change the viability of an option [7]. The other difference in the Kepner-Tregoe methodology is in its quantitative consideration for selecting the best decision. Each WANTS criteria is assigned a quantitative weight and then the information on the criteria about each possible decision is assigned a quantitative value [7]. The optimality of each decision is determined by the sum of the products between the values and weights. The highest is selected as the best option. An example of the Kepner-Tregoe methodology is shown in figure 6.

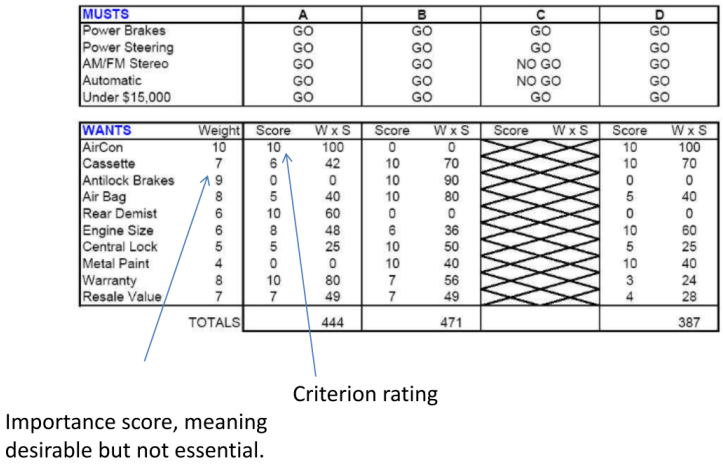


Figure 6: Kepner-Tregoe methodology applied to purchasing a new vehicle [7]

The final MCDA method considered was AHP. AHP is a highly complicated but accurate MCDA method that models the decision to be made as a hierarchy [8]. AHP involves a complicated algorithm for computing the final score of each option (details available in [8]). AHP is considered to be very effective at making complicated decisions with conflicting criteria when the problem can be decomposed into a hierarchy [8], [9]. Some criticisms against AHP are given in [9] and include that the method gives little guidance in the transformation of problems into a hierarchy suitable for application of the method. An example application of AHP is shown in figure 7.

	DECISION CRITERIA					
PAIRWISE COMPARE	**Emissions**	**Fuel Cost**	**Range**	**Vehicle Cost**		
Emissions	1	1	5	5		
Fuel Cost	1	1	5	5		
Range	0.2	0.2	1	3		
Vehicle Cost	0.2	0.2	0.33	1		
CRITERIA MATRIX	**Emissions**	**Fuel Cost**	**Range**	**Vehicle Cost**		**PREFERENCE VECTOR**
Propane	0.07	0.08	0.24	0.65		0.41 **Emissions**
Hybrid Electric	0.15	0.19	0.70	0.29	×	0.41 **Fuel Cost**
Electric	0.78	0.72	0.06	0.06		0.12 **Range**
	GREEN VEHICLE OPTIONS					0.07 **Vehicle Cost**
MATRIX COMPUTATIONS	**Propane**	**Hybrid Electric**		**Electric**		
Final Scores	**0.13**	**0.24**		**0.62**	=	1 **Sum of Scores**
SELECTED VEHICLE				✓		

Figure 7: AHP applied to selecting a green vehicle [5]

3.1.5 Application of the Pugh Method

The selected MCDA method was the Pugh method. This method was selected due to its simple and intuitive matrix communication. When applied to the task of selecting a UAV autopilot, it gave the results shown in figure 8. The final autopilot selected was the PXFmini and Raspberry Pi 3 even though the result of the Pugh method was PXFmini and Raspberry Pi 0. The biggest factor in this decision was that the ease of developing on the Raspberry Pi 3. The decision to develop with the Raspberry Pi 3 as our platform still allows migration back to the Raspberry Pi 0 if profiling later indicates that the Raspberry Pi 0's lower computation could support the developed software.

3.2 Software

The software developed needed to be capable of meeting the objectives specified in section 2. The selected autopilot hardware is capable of meeting all these objectives; however, it is the software architecture that implements the ability to meet these objectives.

Criterion	PXFmini + RPi0	PXFmini + RPi3	Pixhawk	APM2.6
Power	S	-	S	S
Weight	S	-	S	S
Processing	+	++	S	S
Peripherals	+	+	S	-
Sensors	S	S	+	S
Cost	S	-	-	S
System Usage	+	++	S	S
Score	**+3**	**+2**	**0**	**-1**

Figure 8: Pugh method applied to selecting a UAV autopilot

The software architecture was largely determined by the selection of the autopilot hardware. The PXFmini selected is dependent on a Raspberry Pi for its computation; therefore, dictating the use of a Linux OS. ErleRobotics, the manufacturer of the PXFmini, provides an OS image. That OS image is Debian-based with a real time patched kernel and including the APM flight stack [10].

3.2.1 Real Time Operating System

A real time system is one that must respond to asynchronous inputs in a specified time [11]. An RTOS is an OS that can perform processes and respond to inputs within a guaranteed time [12]. Three different levels of RTOS are possible: hard, soft, and firm. A hard RTOS must respond within a given time or it is considered a failure [11]. A soft RTOS must respond on average within a given time [11]. A firm RTOS contains both a limit on average response and on absolute response [11].

The OS image provided by ErleRobotics is a Debian-based Linux distribution compiled with RT_PREEMPT [10]. Compiling the kernel with RT_PREEMPT allows for the majority of the kernel's operation to be preemptively interrupted to service the high priority real time tasks and includes high resolution timers in the OS for highly deterministic task scheduling [13]. The RTOS created by compiling with RT_PREEMPT is a hard RTOS [13].

3.2.2 ArduPilot

ArduPilot is an open-source autopilot software developed and distributed in C++. ArduPilot is over 5 years old and has been used in over 1 million vehicles [14]. ArduPilot is operated on hardware systems by running it from its compiled executable. ArduPilot is distributed with the GNU GPLv3 license [14]. The most important feature of the GNU GPLv3 license in comparison with other

popular open-source licenses (e.g. Apache, MIT, or BSD) is that it is copyleft [15]. Copyleft means that any modifications made to the source code and distributed, either as source code or as a compiled version, must be released under the same license [16]. This does not restrict commercial use of the software and a company can still sell copies of software under the GNU GPLv3 license [16].

ArduPilot has been used in recent research. Coombes et al. demonstrated an autopilot system based on ArduPilot for rapid prototyping of high-level control algorithms [17]. Their work demonstrated the use of ArduPilot for handling low-level control while being able to control the UAV at a high-level through Simulink [17]. Also in a recent publication, Ryan et al. used ArduPilot to design a UAV system for studying calving dynamics at Store Glacier [18]. This system flew in a high risk environment and was able to recover enough images to apply photogrammetric methods (discussed in section 4.1) to create a digital elevation model (DEM) of the glacier for studying calving dynamics [18].

3.2.3 ROS

The Robot Operating System (ROS) is a communication framework for multilingual software components across potentially heterogeneous hardware [19]. Over 3000 ROS packages are available and the ROS wiki contains over 22 000 pages with a Q&A website containing over 13 000 questions (with over 70% answered) [20]. The core of ROS is licensed under the 3 clause BSD license [20]. This is not a copyleft license and specifies no restrictions on the licensing of modified versions of the software [21]. The most important feature of the 3 clause BSD license is the liability protection it provides to the copyright holder and contributors [21].

A system using ROS divides processing across nodes that communicate with each other using messages published to topics or by subscribing to relevant topics [19]. Tools included in ROS also allow for visualization of all nodes and topics in the ROS system [19]. ROS is widely used both in research and in industry. One of the most famous robots running ROS is Willow Garage's PR2 [22]. This expensive robot uses a stereo vision system with two manipulators and a mobile base developed for research purposes [23]. The PR2 robot is shown in figure 9. Another example of the wide-spread use of ROS is at Clearpath Robotics, a leader in the robotics industry [24], where it is used in industrial applications. Clearpath Robotics also supports the development of ROS and provides information on using the ROS ecosystem for research and products [25].

3.2.4 MAVLink and MAVROS

MAVLink is a communication protocol designed for the specific use case of

Figure 9: The PR2 robot developed by Willow Garage [22]

Micro-Aerial Vehicles (MAVs) [26]. The design of MAVLink is inspired in part by CAN bus [26] and has a simple packet structure at OSI level 2 [27]. A detailed description of the packet structure is given in [27]. MAVLink is designed specifically for the use of a MAV communicating back to the ground control station [26]. In this architecture it will be used internally in the autopilot system for communicating from Python to the ArduPilot process using MAVROS.

In order to communicate with ArduPilot over MAVLink, MAVROS is used. MAVROS is a ROS package that creates a ROS node creatively named `mavros_node`. The existence of the MAVROS node in the system allows for multiple benefits in interacting with the ArduPilot process. The first benefit it provides is publishing all MAVLink data received from ArduPilot to ROS topics. A list of the topics available is given at [28] and it includes GPS, IMU, and magnetometer data. Another benefit given by the MAVROS node is the ability to use shell commands for interacting with the ArduPilot process [28]. The most important of these are `mavcmd`, `mavsafety`, and `mavwp`. Using ROS topic publications for

data access and ROS shell commands for high-level control will be discussed in sections 3.3 and 3.4 respectively.

The generator for MAVLink messages is released under the GNU LGPL license and any generated output is licensed under the MIT license [26]. The key difference between the LGPL license and the GPL license is in use of licensed libraries. The LGPL license provides specific instructions for combined works and use of LGPL licensed libraries [29]. The MIT license provides no limitations on use of the original software or licensing of derivative works [30]. The MIT license does provide similar liability protection to that given under the BSD license previously discussed [30]. MAVROS contains components licensed under GPL, LGPL and BSD licenses, all of which have been previously discussed.

3.2.5 Architecture

The software architecture created with the components discussed above is shown in figure 10. This architecture allows the autopilot system to meet the objectives specified in section 2. The software developed to meet the objectives is written exclusively in Python and run as a background process in user space. The software developed will be discussed in sections 3.3 and 3.4.

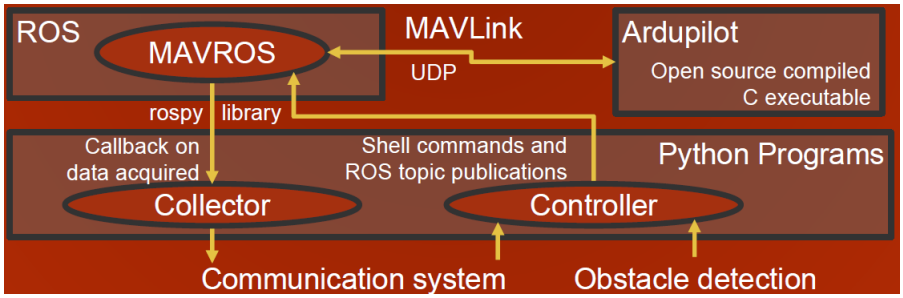


Figure 10: Diagram showing the interaction between software components

The important interactions are highlighted with gold arrows showing the direction of data flow. The Python programs interact with the MAVROS node through use of the `rospy` library and shelling MAVROS commands. The `rospy` library allows Python programs to create nodes and publish to topics, as well as receive callbacks when other nodes publish to subscribed topics [31]. This method is used to access the required data from ArduPilot through MAVROS and will be discussed in detail in section 3.3. Another way that the Python programs interact with the MAVROS node is through the execution of shell commands. This is used exclusively for high-level control and as such will be discussed in section 3.4.

3.3 Data Access

One of the objectives from section 2 for the autopilot system is to be able to acquire sensor data. The critical data required is GPS, but accelerometer and magnetometer were also deemed to be important data for collection. This section will present the ability to access the relevant data from the ArduPilot process through the use of MAVROS. The developed data access library is shown in appendix A and an example of using this library is given in appendix B.

3.3.1 Implementation

As discussed earlier, data access requires use of the `rospy` Python library. This library allows for the creation of a ROS node and for that node to subscribe to a topic. When a subscribed topic receives data, a callback is issued to a function supplied during initialization. The Python function implements the saving of data and alerting the program using the library that data has been updated. This process can be decomposed into two parts: the initialization and the callback function.

A flowchart of the initialization for the data access subsystem is shown in figure 11. The initialization does two tasks on system startup: it creates a node in the ROS system and then subscribes that node to the desired topic(s) passing callback functions. Using the `rospy` library, creation of a node is completed with:

```
rospy.init_node('node_name', anonymous=True)
```

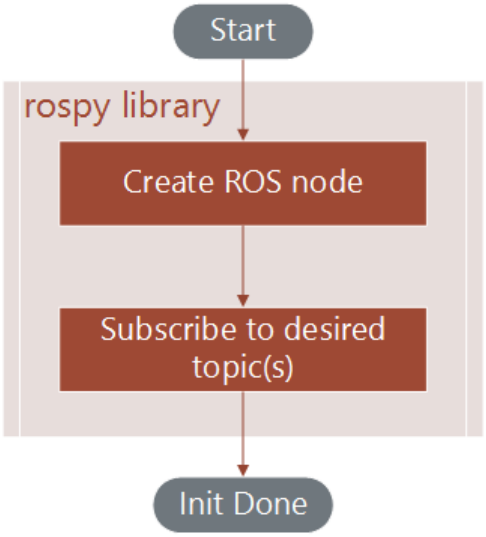


Figure 11: Flowchart for initialization of the data access subsystem

This creates a new node in the ROS system named `node_name`, unless a node with the same name already exists. In this case setting `anonymous=True` allows the library to append a unique identifier to the end of the name. Subscribing this node to a topic is achieved with:

```
rospy.Subscriber(
        'mavros/global_position/global',
        NavSatFix,
        callback_fnc
)
```

This example line subscribes the node to the topic `mavros/global_position/global` which gives data of type `NavSatFix`. Every time there is a publication to the `mavros/global_position/global` topic the function `callback_fnc` will be called with the data as the first argument. One node can be subscribed to multiple topics.

A flowchart for the actions taken when receiving a callback is shown in 12. A prototype for a callback function is:

```
def callback_fnc(data):
        pass
```

The `data` parameter is the message published to the topic. Its type and structure can be determined from its documentation. For example, data of type `NavSatFix` has the following definition [32]:

```
std_msgs/Header header
sensor_msgs/NavSatStatus status
float64 latitude
float64 longitude
float64 altitude
float64[9] position_covariance
uint8 position_covariance_type
```

A global counter is used for tracking the number of calls the callback function has received. This counter is used to arbitrate the update rate of the global variables containing the data. In Python this can be accomplished with:

```
def callback_fnc(data):
        global cb_rx
        cb_rx += 1
        if cb_rx%10 == 0:
                print(str(data.latitude) + '\n')
```

In this example, the latitude would be printed to the screen every tenth publication. This arbitration helps reduce the computational load required when intensive tasks and high rate publications are combined, such as file operations to save accelerometer data. This was determined to be necessary when file corruption occurred without arbitration for logging accelerometer data. In the data

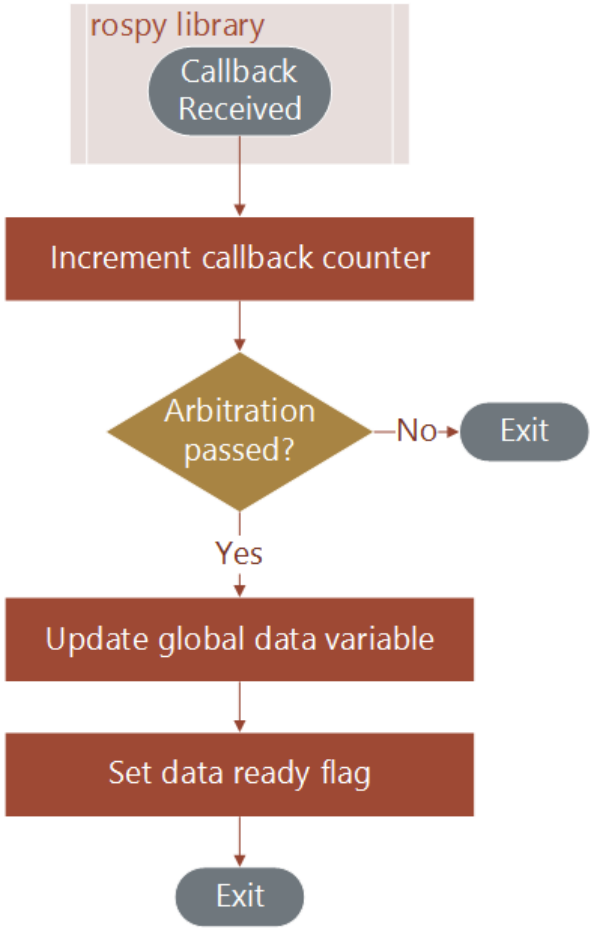


Figure 12: Flowchart for servicing callbacks received for messages published to topics

access library code the arbitration block contains code for copying the local data to a global variable and setting a data ready flag for use by the transmission subsystem.

3.3.2 Testing

This method of accessing data from the ArduPilot process using MAVROS and `rospy` was implemented and tested. The ability to access accelerometer and magnetometer data is shown in figures 13 and 14 respectively. This test involved placing the autopilot system in three different orientations, each for approximately the same amount of time. Since the direction of the gravitational

field and magnetic field does not change, this should (if the data is valid) create a different linear combination of values for the axes and this is what is shown in figures 13 and 14. A test was also performed to record GPS data during a walk around Carleton University outside Mackenzie Building and Canal Building. This GPS data is shown overlaid on a Google Earth satellite image in figure 15.

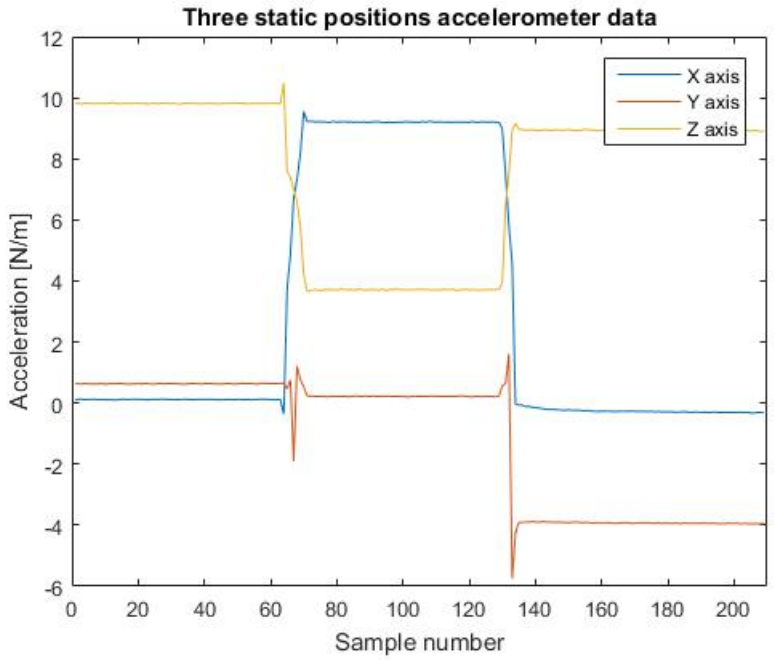


Figure 13: Plot showing accelerometer data for three clearly distinguishable orientations

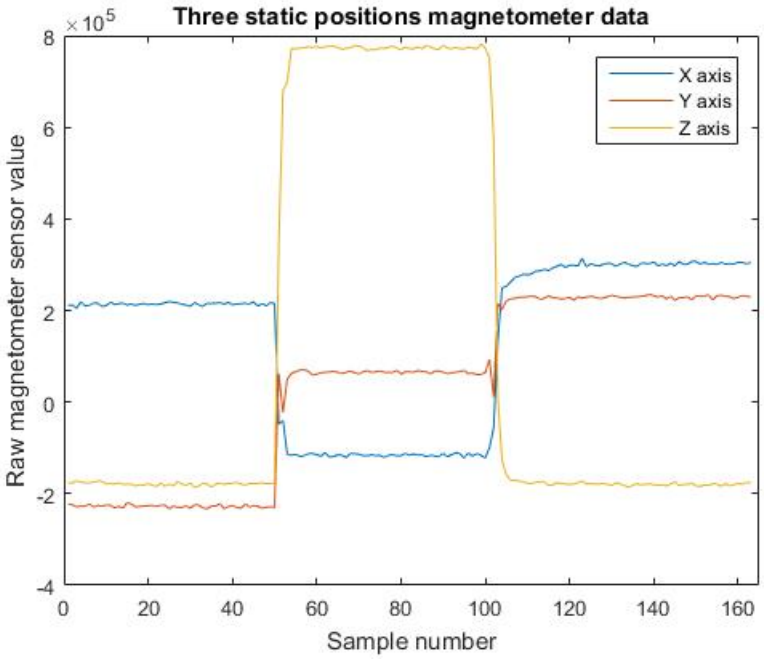


Figure 14: Plot showing magnetometer data for three clearly distinguishable orientations

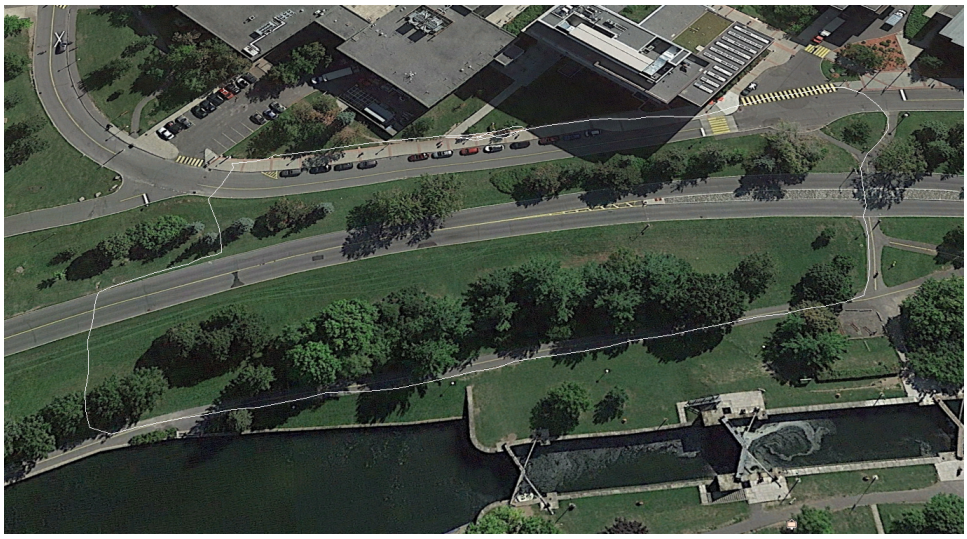


Figure 15: Overlay of walk GPS data (white line) on satellite image of Carleton University outside Mackenzie Building and Canal Building

3.4 High-Level Control

The high-level control subsystem of the autopilot must meet the following two objectives described in section 2: accept high-level commands (such as takeoff, waypoint navigation, and landing) and allow for emergency landing or manual flight control. This section will present the ability to control the flight through high-level commands. The manual control safety subsystem will be shown in section 3.4.3.

3.4.1 Implementation

The MAVROS node allows for ROS shell commands to be used for high-level flight control. In presenting the available commands exposed by the MAVROS node, the flight is considered to consist of four possible states: pre-flight, flight, landing, or emergency. The available commands and the flight states to which they correspond are summarized in table 2. Note that all the MAVROS shell commands must begin with `rosrun mavros`. The format of the waypoint file will be presented in section 3.4.2. An entire mission of the UAV can be controlled using these commands; however, manual control is still necessary for safety reasons. The design of the subsystem implementing the ability to assume manual control will be shown in section 3.4.3.

Flight State	Purpose of Command	MAVROS Shell Command
Pre-flight	Set launch point as home	`mavcmd sethome`
Pre-flight	Set UAV flight path	`mavwp load file.wp`
Pre-flight	Enable motor control	`mavsafety arm`
Pre-flight	Takeoff at current location	`mavcmd takeoff`
Flight	Change active waypoint	`mavwp goto i`
Flight	Change UAV flight path	`mavwp load file.wp`
Flight	Clear UAV flight path	`mavwp clear`
Landing	Land at home	`mavcmd land`
Landing	Land on current location	`mavcmd landcurr`
Emergency	Make emergency landing	`mavcmd landcurr`
Emergency	Hover at current position	`mavwp clear`

Table 2: Available high-level commands using the MAVROS node [28] (shell commands omit the necessary `rosrun mavros`)

All of these commands can be executed from a Python program by making use of the `subprocess` library available in all installations of Python [33]. The `subprocess` library is included in all currently supported Python versions [34]. An example of using the `subprocess` library is shown here:

```
subprocess.call("rosrun␣mavros␣mavcmd␣sethome")
```

This example will execute the `rosrun mavros mavcmd sethome` command as if it had been entered as a regular shell command. One important fact about the

use of this library is that the calls made using `subprocess.call()` are blocking. That is, the flow of the program will be blocked until the completion of the command [33]. All of the commands in table 2 return after completing their tasks; however, in some cases (notably loading a large waypoint file) this may take on the order of seconds. One possible way to avoid this limitation would be to use `subprocess.Popen` which runs shell commands in a new process [35].

3.4.2 Waypoint File Format

The format of the waypoint file used in the `rosrun mavros mavwp load file.wp` command required research to determine, as no documentation of the new version is available yet. The format does not appear to be handled differently in the actual MAVROS code where it is parsed and passed to ArduPilot [36]. The format of the old format for waypoints is given at [37]. The fields of an entry in the waypoint file is shown in table 3. A graphic created for showing the new format for waypoint files is included as appendix C.

Field Name	**Purpose of Field**
Index	Used to order the execution of waypoints
Current WP	The waypoint to set as current will have a 1 here
Frame	Frame of reference used in specifying coordinates or 2 for a mission command (explained at [38], but limited by use of MAVROS)
Command	Command to be executed (available commands compatible with MAVROS discussed below)
Parameter 1 Parameter 2 Parameter 3 Parameter 4	These parameters differ depending on the command used and the parameters for each command are given at [38] or [39]; however, not all commands presented are available with MAVROS (discussed below)
Latitude	Latitude for waypoint or command
Longitude	Longitude for waypoint or command
Altitude	Altitude for waypoint or command
Autocontinue	1 to continue to next command after this one completes

Table 3: Format of an entry in the QGC WPL 120 format file used for waypoint specification [37], [38], [39]

The waypoint file can specify many different types of commands to be executed during the mission. For example, a specified delay can be inserted at each waypoint or a return to launch location can be given as a line in the waypoint file. Ordinarily ArduPilot would support a wide range of available commands or frames of reference to be set in the waypoints file; however, MAVROS limits the set of available commands and frames of reference [36]. The limited set of frames of reference are shown in table 4 and the limited set of commands are shown in table 5. This limitation is due to the hardcoded nature of lines 21

through 49 in `mission.py` of MAVROS's source code, and could easily be extended to support more commands or frames of reference should they be deemed necessary for this project (a pull request (PR) may even be welcomed) [36].

Frame of Reference	Code	Description
FRAME_GLOBAL	0	WGS84 coordinate system with altitude over mean sea level
FRAME_GLOBAL_REL_ALT	3	WGS84 coordinate system with altitude over ground at home position
FRAME_LOCAL_ENU	4	Local coordinate frame: x is east, y is north, and z is up
FRAME_LOCAL_NED	1	Local coordinate frame: x is north, y is east, and z is down
FRAME_MISSION	2	Indicates a mission command

Table 4: Valid frames of reference in waypoint files loaded with MAVROS in the order listed in the source code [36], [38]

Command	Code	Description
LAND	21	Land at specified location
LOITER-TIME	19	Remain at specified location for specified time
LOITER-TURNS	18	Remain at specified location for specified rotations
RTL	20	Return to launch location
TAKEOFF	22	Takeoff from current location
WAYPOINT	16	Fly to specified location
COND-DELAY	112	Delay execution of waypoint line by specified time
COND-CHANGE-ALT	113	Change altitude at the specified speed
COND-DISTANCE	114	Waits for UAV to be within a specified distance of waypoint
COND-YAW	115	Change yaw to specified heading
DO-JUMP	177	Change next command to execute to the specified one (by index)
DO-CHANGE-SPEED	178	Set the speed of the UAV to the specified value
DO-SET-RELAY	181	Set a pin on the autopilot to specified binary value
DO-REPEAT-RELAY	182	Set a pin on the autopilot to toggle at the specified rate
DO-SET-SERVO	183	Set a servo with a given PWM value
DO-REPEAT-SERVO	184	Set a servo to transition between values at the specified rate
DO-SET-ROI	201	Set a region of interest for the UAV to continuously face

Table 5: Valid commands in waypoint files loaded with MAVROS in the order listed in the source code [36], [39]

3.4.3 Manual Control Safety Subsystem

For the purposes of safety, a manual control override system was designed. This manual override system uses a standard hobby RF transmitter/receiver pair to assume control over the UAV should it be deemed necessary. Possible cases requiring manual control include, but are not limited to, loss of XBee communication, base station PC issues, GUI errors (either human or programming), and failure of Python autopilot programs. The manual override system is shown in figure 16.

Figure 16: System for implementing the manual control override

An RF transmitter/receiver pair operating in the 915MHz ISM band is used for the manual control. This will not interfere with the main XBee communication channel operating at 2.45GHz. The output of the receiver radio is a set of pulse width modulation (PWM) channels; however, the PXFmini requires the input to be a pulse position modulated (PPM) signal [1]. The PPM signal represents the multiple PWM channels over a single wire in a compressed format [40]. A signal converter was acquired and used for converting the PWM channels to a PPM signal.

This system implementation was tested by using an oscilloscope attached to the motor driver pins on an armed PXFmini while manipulating the RF transmitter controls. Variation in the motor drive PWM signals were observed as yaw, pitch, and throttle controls were changed. This shows that the UAV is theoretically able to be controlled manually through the manual control override system; however, without a manual flight test this ability is not fully confirmed. No autonomous operation should be attempted without flight verification of the manual control safety subsystem.

3.4.4 Remaining Work on High-Level Control

There is remaining work on the high-level control subsystem for the autopilot. The manual control safety subsystem requires testing as mentioned in section 3.4.3. A library wrapping the command line calls to interact with the ArduPilot process is necessary for interfacing with the communication system. Also, a generator for waypoint files remains to be written so that the waypoints transmitted by the ground station can be combined into a waypoint file in the MAVROS format.

4 Future Work

The most significant future work for this project is final integration and flight testing. A sufficiently large frame quadcopter UAV has been acquired using CUESEF funding that can accommodate the designed system. After the completion of the outstanding work required on the high-level control subsystem (see discussion in section 3.4.4), all system components will be ready for a final integration and flight testing.

During final integration, the performance of the Raspberry Pi 3 should be profiled and a decision made whether the Raspberry Pi 0 would be able to support the computational load of the system. This would allow for a significant reduction in total payload weight that would increase flight time. It would result in a 6.44W reduction in lift power required, using the $200\frac{\text{W}}{\text{kg}}$ power estimate from [41]. This profiling could be accomplished by logging the output of Linux's `top` profiling program [42].

The autopilot system can accomplish missions specified in waypoint files autonomously; however, currently no collision avoidance is implemented using the data produced by the collision detection system. Using this data to modify the path followed would decrease the risk of a collision caused by incorrect path entry in the GUI. At the very least, stopping the UAV and alerting an operator would considerably increase the safety of the system.

One significant section for future work would be to use the visual and infrared images to produce a 3D model of the affected building. This represents the most intuitive method for providing the information gathered by the UAV system to a non-technical user. The method for producing 3D models from images will be discussed further in the section 4.1.

4.1 Photogrammetry

Photogrammetry is the science of extracting measurements from images [43]. Photogrammetry is a very popular technique used with UAVs for building 3D models or digital elevation models (DEMs) [18], [44], [45], [46], [47]. An especially interesting photogrammetric processing algorithm for a set of unlabeled images is called Structure from Motion (SfM) [45]. An example use of photogrammetry from a UAV is given in [47] and one of the constructed 3D models is shown in figure 17.

SfM has recently risen in prominence given its simple application and largely automated workflow [48]. The SfM algorithm involves solving for the orientation and location of each image's camera by matching features between images [49]. The popularity of this technique has led to the development of software for photogrammetric processing of images, both proprietary (e.g. Agisoft's Photo-

Figure 17: Digital 3D model of a Mayan ruin in Copan, Honduras created with the use of UAVs [47]

Scan and Pix4Dmodel) and open-source (e.g. OpenSfM and Bundler).

The major challenges of using SfM in this application are the dynamic nature of the environment to be made into a 3D model and the speed at which the firefighters need to see the initial 3D model. The challenge posed by the dynamic nature of the firefighter response environment can be addressed through the use of a real time updating variant of the SfM algorithm (e.g. [50], [51]). The speed for the initial model creation is determined in part by the time to gather a set of images from the UAVs and in part by the run-time of the SfM algorithm on those images. A set of images could by gathered quicker through the use of multiple UAVs. Also, the run-time of the SfM algorithm is decreasing as improvements are made in the dependent algorithms and GPU computation becomes cheaper [52]. Increasing the computational power of the base station would also decrease the time to produce the initial 3D model.

5 Conclusion

This report has presented the progress on the UAV control and autonomous flight subsystem for a quadcopter UAV fire response system undertaken as a fourth-year capstone project at Carleton University in Ottawa, Canada. The development of the autopilot system was presented, along with examples of data access from the autopilot and high-level control of the UAV's flight. Potential future work on the project was also discussed.

It is the author's hope that a system such as described in this report will someday become a tool for firefighters to increase their effectiveness and improve their safety. In the current era of strong political divides, a demonstration that robotics can exist outside class-based fears of automation targeting lower wage jobs is necessary. The promising nuclear industry of the 1970's did not live up to its potential not because of technological hurdles, but rather a strong social counter movement. The robotics industry should take actions so that such an unwarranted fate may remain unique to the nuclear industry. Robotic systems need only replace the hole existing prior to their realization.

A Data Access Library for the F.I.R.E. System (fire_data.py)

```
import rospy
from sensor_msgs.msg import Imu, MagneticField, NavSatFix

# global variables (modified asynchronously in callbacks)
# imu data
accel_x = 0
accel_y = 0
accel_z = 0
accel_flag = False
accel_rx = 0
# mag data
mag_x = 0
mag_y = 0
mag_z = 0
mag_flag = False
mag_rx = 0
# gps data
gps_lat = 0
gps_long = 0
gps_alt = 0
gps_flag = False
gps_rx = 0

class data_access(object):
    """Class for F.I.R.E. system data collection from ArduPilot"""

    def __init__(self, accel_mod=25, mag_mod=10, gps_mod=5):
        # imu modulus
        self.accel_mod = accel_mod
        # mag modulus
        self.mag_mod = mag_mod
        # gps modulus
        self.gps_mod = gps_mod

        self.ros_init('fire_node')

    def ros_init(self, node_name):
        rospy.init_node(node_name, anonymous=True)
        rospy.Subscriber(
            'mavros/imu/data',
            Imu,
            self.accel_cb
```

```
        )
        rospy.Subscriber(
            'mavros/imu/mag',
            MagneticField,
            self.mag_cb
        )
        rospy.Subscriber(
            'mavros/global_position/global',
            NavSatFix,
            self.gps_cb
        )

    def accel_cb(self, data):
        global accel_rx, accel_x, accel_y, accel_z, accel_flag
        accel_rx += 1
        if accel_rx%self.gps_mod == 0:
            accel_x = data.linear_acceleration.x
            accel_y = data.linear_acceleration.y
            accel_z = data.linear_acceleration.z
            accel_flag = True

    def mag_cb(self, data):
        global mag_rx, mag_x, mag_y, mag_z, mag_flag
        mag_rx += 1
        if mag_rx%self.mag_mod == 0:
            mag_x = data.magnetic_field.x
            mag_y = data.magnetic_field.y
            mag_z = data.magnetic_field.z
            mag_flag = True

    def gps_cb(self, data):
        global gps_rx, gps_x, gps_y, gps_z, gps_flag
        gps_rx += 1
        if gps_rx%self.gps_mod == 0:
            gps_lat = data.latitude
            gps_long = data.longitude
            gps_alt = data.altitude
            gps_flag = True
```

B Example Using the Data Access Library

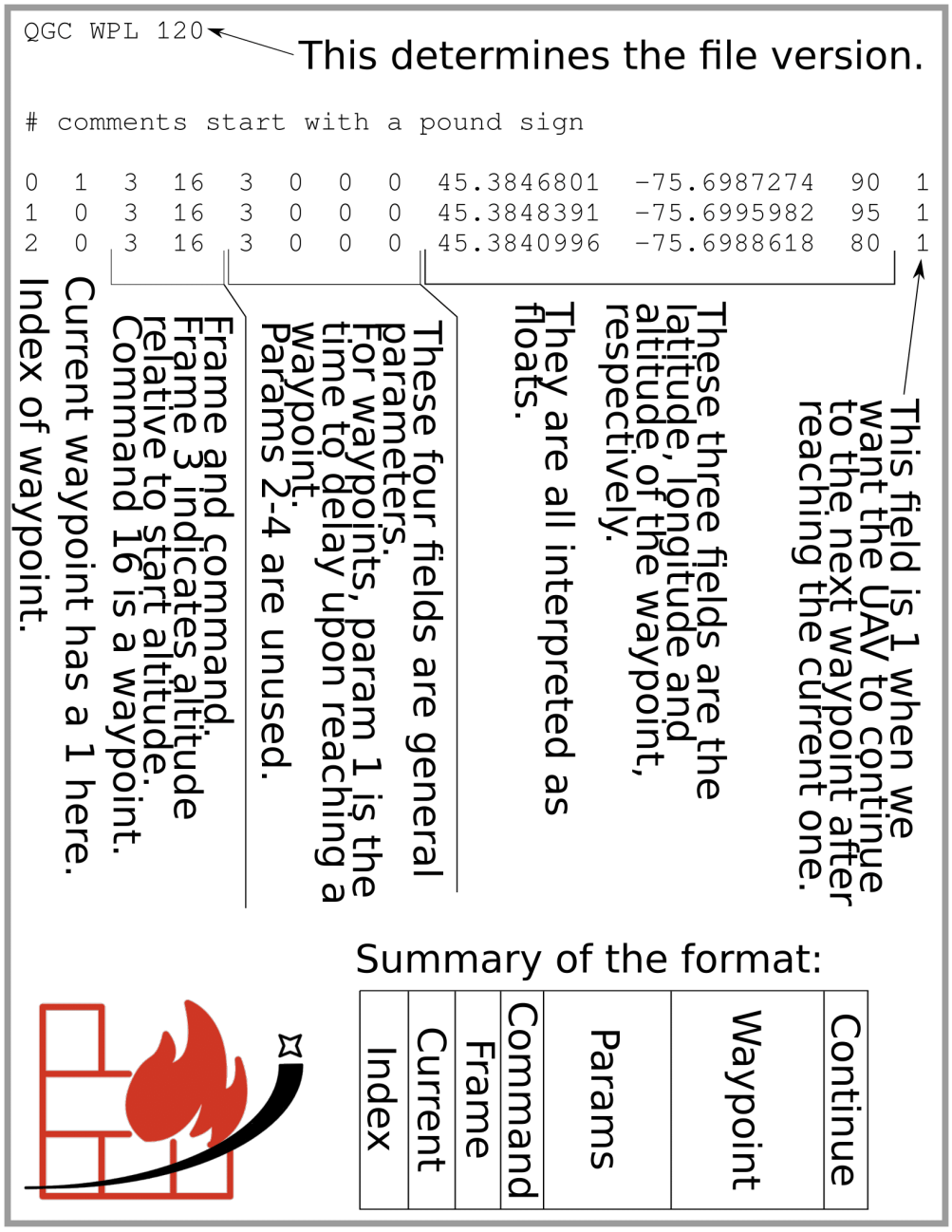
```
import time

# load the data access library
import fire_data

# call the data_access constructor
fire_data.data_access()

# now the data is available to access
# this shows an example of printing the data to the screen
while True:
    if fire_data.accel_flag:
        print('x axis: ' + str(fire_data.accel_x) + '\n')
        print('y axis: ' + str(fire_data.accel_y) + '\n')
        print('z axis: ' + str(fire_data.accel_z) + '\n')
        time.sleep(5)
```

C Graphic Showing the QGC WPL 120 Format



References

[1] "PXFmini — Erle Robotics", Erlerobotics.com, 2017. [Online]. Available: http://erlerobotics.com/blog/product/pxfmini/. [Accessed: 26- Mar- 2017].

[2] "Home - Pixhawk Flight Controller Hardware Project", Pixhawk.org, 2017. [Online]. Available: https://pixhawk.org/. [Accessed: 31- Mar- 2017].

[3] "Archived:APM 2.5 and 2.6 Overview Copter documentation", Ardupilot.org, 2017. [Online]. Available: http://ardupilot.org/copter/docs/common-apm25-and-26-overview.html. [Accessed: 30- Mar- 2017].

[4] "ArduPilot Release Notes", GitHub, 2017. [Online]. Available: https://github.com/ArduPilot/ardupilot/blob/master/ArduCopter/ReleaseNotes.txt. [Accessed: 29- Mar- 2017].

[5] T. Studies and D. Analysis, "SURVEY OF TRADE STUDY METHODS FOR PRACTICAL DECISION-MAKING Sample Application of Decision Analysis Methods," p. 2010, 2010.

[6] G. A. Hazelrigg, "Letter to the Editor re The Pugh controlled convergence method: model-based evaluation and implications for design theory," Res. Eng. Des., vol. 21, no. 3, pp. 143144, 2010.

[7] Skorkovsk, "Kepner-Tregoe Methodology," 2013.

[8] M. Alexander, "Decision-Making using the Analytic Hierarchy Process (AHP) and SAS/ IML," United States Soc. Secur. Adm. Balt., pp. 112, 2012.

[9] F. Hartwich, "Weighting of Agricultural Research Results: Strength and Limitations of the Analytic Hierarchy Process (AHP)," pp. 118, 1999.

[10] "Debian — Erle Robotics Docs", Docs.erlerobotics.com, 2017. [Online]. Available: http://docs.erlerobotics.com/brains/os_images/debian. [Accessed: 02- Apr- 2017].

[11] L. Madan and K. A. B. Bhushan, "REAL-TIME OPERATING SYSTEM," vol. 4, no. 3, pp. 3950, 2014.

[12] M. Barabanov, "A Linux-based Real-Time Operating System," New Mexico Institute of Mining and Technology, 1997.

[13] M. Mossige, P. Sampath, and R. G. Rao, "Evaluation of Linux rt-preempt for embedded industrial devices for Automation and Power Technologies-A Case Study," Proc. 9th Real-Time Linux Work., pp. 16, 2007.

[14] "ArduPilot Open Source Autopilot", Ardupilot.org, 2017. [Online]. Available: http://ardupilot.org/. [Accessed: 30- Mar- 2017].

[15] "GNU General Public License version 3 — Open Source Initiative", Opensource.org, 2017. [Online]. Available: https://opensource.org/licenses/GPL-3.0. [Accessed: 02- Apr- 2017].

[16] "Frequently Answered Questions — Open Source Initiative", Opensource.org, 2017. [Online]. Available: https://opensource.org/faq. [Accessed: 02- Apr- 2017].

[17] M. Coombes, O. McAree, W.-H. Chen, and P. Render, "Development of an autopilot system for rapid prototyping of high level control algorithms," Proc. 2012 UKACC Int. Conf. Control, no. September, pp. 292297, 2012.

[18] J. C. Ryan et al., "UAV photogrammetry and structure from motion to assess calving dynamics at Store Glacier, a large outlet draining the Greenland ice sheet," Cryosphere, vol. 9, no. 1, pp. 111, 2015.

[19] M. Quigley et al., "ROS: an open-source Robot Operating System," Icra, vol. 3, no. Figure 1, p. 5, 2009.

[20] "ROS.org — Powering the world's robots", Ros.org, 2017. [Online]. Available: http://www.ros.org/. [Accessed: 03- Apr- 2017].

[21] "The 3-Clause BSD License — Open Source Initiative", Opensource.org, 2017. [Online]. Available: https://opensource.org/licenses/BSD-3-Clause. [Accessed: 03- Apr- 2017].

[22] S. Cousins, B. Gerkey, K. Conley, and W. Garage, "Sharing software with ROS," IEEE Robot. Autom. Mag., vol. 17, no. 2, pp. 1214, 2010.

[23] "Overview — Willow Garage", Willowgarage.com, 2017. [Online]. Available: http://www.willowgarage.com/pages/pr2/overview. [Accessed: 02- Apr- 2017].

[24] P. Villavicencio, "Clearpath Wins 2016 RBR50 Award - Clearpath Robotics", Clearpath Robotics, 2017. [Online]. Available: https://www.clearpathrobotics.com/2016/02/clearpath-wins-rbr50-award-2016/. [Accessed: 02- Apr- 2017].

[25] I. Baranov, "How to Guide: ROS 101 - Clearpath Robotics", Clearpath Robotics, 2017. [Online]. Available: http://www.clearpathrobotics.com/2014/01/how-to-guide-ros-101/. [Accessed: 02- Apr- 2017].

[26] "MAVLink Micro Air Vehicle Communication Protocol - QGroundControl GCS", Qgroundcontrol.org, 2017. [Online]. Available: http://qgroundcontrol.org/mavlink/start. [Accessed: 03- Apr- 2017].

[27] J. A. Marty, "Vulnerability Analysis of the Mavlink Protocol," Air Force Institute of Technology, 2014.

[28] "mavros - ROS Wiki", Wiki.ros.org, 2017. [Online]. Available: http://wiki.ros.org/mavros. [Accessed: 03- Apr- 2017].

[29] "GNU Lesser General Public License version 3.0 — Open Source Initiative", Opensource.org, 2017. [Online]. Available: https://opensource.org/licenses/LGPL-3.0. [Accessed: 04- Apr- 2017].

[30] "The MIT License — Open Source Initiative", Opensource.org, 2017. [Online]. Available: https://opensource.org/licenses/MIT. [Accessed: 04- Apr- 2017].

[31] M. Quigley, B. Gerkey, and W. D. Smart, "Programming Robots with ROS", Early ver3. OReilly Media, Inc., 2010.

[32] "sensor_msgs/NavSatFix Documentation", Docs.ros.org, 2017. [Online]. Available: http://docs.ros.org/api/sensor_msgs/html/msg/NavSatFix.html. [Accessed: 04- Apr- 2017].

[33] "17.1. subprocess Subprocess management Python 2.7.13 documentation", Docs.python.org, 2017. [Online]. Available: https://docs.python.org/2/library/subprocess.html. [Accessed: 07- Apr- 2017].

[34] "17.5. subprocess Subprocess management Python 3.7.0a0 documentation", Docs.python.org, 2017. [Online]. Available: https://docs.python.org/3.7/library/subprocess.html. [Accessed: 07- Apr- 2017].

[35] "17.5. subprocess Subprocess management Python 3.6.1 documentation (subprocess.Popen)", Docs.python.org, 2017. [Online]. Available: https://docs.python.org/3/library/subprocess.html#subprocess.Popen. [Accessed: 07- Apr- 2017].

[36] "mavlink/mavros", GitHub, 2017. [Online]. Available: https://github.com/mavlink/mavros/blob/master/mavros/src/mavros/mission.py. [Accessed: 07- Apr- 2017].

[37] "Waypoint Protocol - QGroundControl GCS", Qgroundcontrol.org, 2017. [Online]. Available: http://qgroundcontrol.org/mavlink/waypoint_protocol. [Accessed: 07- Apr- 2017].

[38] "MAVLINK Common Message set specifications", Pixhawk.ethz.ch, 2017. [Online]. Available: https://pixhawk.ethz.ch/mavlink/. [Accessed: 07- Apr- 2017].

[39] "MAVLink Mission Command Messages (MAV_CMD) Copter documentation", Ardupilot.org, 2017. [Online]. Available: http://ardupilot.org/copter/docs/common-mavlink-mission-command-messages-mav_cmd.html. [Accessed: 07- Apr- 2017].

[40] "PWM, PPM, and Serial RX explained", Quad Me Up, 2017. [Online]. Available: https://quadmeup.com/pwm-ppm-and-serial-rx-explained/. [Accessed: 08- Apr- 2017].

[41] V. Kumar. "Aerial Robotics." Class Lecture, Topic: "Design Considerations." Penn Engineering, University of Pennsylvania.

[42] "top(1) - Linux man page", die.net, 2017. [Online]. Available: `https://linux.die.net/man/1/top`. [Accessed: 08- Apr- 2017].

[43] W. Linder, "Digital Photogrammetry: A Practical Course". Berlin: Springer, 2006.

[44] G. Grenzdrffer, A. Engel, and B. Teichert, "The photogrammetric potential of low-cost UAVs in forestry and agriculture," Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci., vol. 1, pp. 12071213, 2008.

[45] F. Mancini, M. Dubbini, M. Gattelli, F. Stecchi, S. Fabbri, and G. Gabbianelli, "Using unmanned aerial vehicles (UAV) for high-resolution reconstruction of topography: The structure from motion approach on coastal environments," Remote Sens., vol. 5, no. 12, pp. 68806898, 2013.

[46] M. Sauerbier and H. Eisenbeiss, "Uavs for the Documentation of Archaeological Excavations," Proc. Isprs Comm. V Mid-Term Symp. Close Range Image Meas. Tech., vol. 38, no. 5, pp. 526531, 2010.

[47] F. Remondino, L. Barazzetti, F. Nex, M. Scaioni, and D. Sarazzi, "Uav Photogrammetry for Mapping and 3D Modeling Current Status and Future Perspectives," ISPRS - Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci., vol. XXXVIII-1/, no. September, pp. 2531, 2012.

[48] M. J. Westoby, J. Brasington, N. F. Glasser, M. J. Hambrey, and J. M. Reynolds, "Structure-from-Motion photogrammetry: A low-cost, effective tool for geoscience applications," Geomorphology, vol. 179, pp. 300314, 2012.

[49] S. Agarwal, Y. Furukawa, and N. Snavely, "Building rome in a day," Commun. , pp. 105112, 2011.

[50] A. Chiuso, P. Favaro, H. Jin, and S. Soatto, "Structure from motion causally integrated over time," IEEE Trans. Pattern Anal. Mach. Intell., vol. 24, no. 4, pp. 523535, 2002.

[51] N. D. Molton, A. J. Davison, and I. D. Reid, "Locally Planar Patch Features for Real-Time Structure from Motion," Bmvc, p. 90.1-90.10, 2004.

[52] C. Wu, "Towards linear-time incremental structure from motion," Proc. - 2013 Int. Conf. 3D Vision, 3DV 2013, pp. 127134, 2013.