Universal UAV Payload Interface

Nolan Reker	**Computer Engineering**
	nolan@nolanswires.com
Drew Troxell	**Computer Engineering**
	troxellus@gmail.com
David Troy	**Electrical Engineering**
	davidtroyjr@gmail.com

Advisor Dr. Foaad Khosmood

California Polytechnic State University
San Luis Obispo, CA
Computer Engineering Department

June 12, 2015

Abstract

Unmanned Aerial Vehicle (UAV) technology is becoming increasingly accessible for civilian use. Both open-source and commercial-purpose UAVs can be obtained affordably or even built. However, the platforms available are very segmented in their customization to a specific application (i.e. land surveying, payload delivery). This project aims to create a Universal Payload Interface (UPI) mounted to the underside of multi-rotors or other UAVs to enable the attachment of customizable sensor payloads. These payloads allow a single UAV to be rapidly reconfigured to perform a multitude of tasks.

The Universal Payload Interface facilitates communication between the payload, onboard flight controller, and operator ground station as well as providing power to the payload from the UAV DC bus. Both autonomous and manual flight regimes are supported. By using the open-source MAVlink protocol, the UPI is not restricted to one particular flight controller or ground station platform.

Contents

Figures

1 Introduction

For a majority of the 20th century, Unmanned Aerial Vehicle (UAV) technology existed only in the realm of military aeronautics. With recent advancements in microelectronics, the cost of microprocessors and MEMS sensors has decreased enough to allow for the development of low-cost UAV platforms in non-military applications. Studies in UAV controls and automation have boomed in recent years as a result. UAVs are now widely available to researchers, businesses, and hobbyists. In particular, 4 rotor vehicles known as quadcopters have become fairly popular UAV platforms due to their relatively simple mechanical structure − the complex swashplate mechanisms relied upon by single-rotor craft can be disregarded, instead relying upon controls abstractions in software. This software abstraction lends itself to the full automation of flight control processes. With this in mind, companies like Amazon, Google, and Facebook are pioneering the commercial applications of autonomous UAVs.

Following in the tradition of military aeronautical engineering, current UAV designs are often tailored to a specific purpose: payload delivery, aerial photography, recreational flight, etc. Currently lacking, however, are systems that allow for the modular reconfiguration of a UAV to enable a variety of features. A surveyor, for example, may utilize an expensive remote-sensing capable UAV equipped with a Light Detection and Ranging (LIDAR) system. Designed to support an already complex payload, the vehicle cannot physically support many auxiliary features. The surveyor may be out of luck if he wished to use the vehicle to instead perform aerial photography. Such a task could require system rewiring and reprogramming. Simply put, there no commonly-utilized plug-and-play payload handling systems available today.

The goal of this project is the development of a novel Universal Payload Interface (UPI) that facilitates the exchange of data, flight instructions, and power between a payload, its vehicle, and any associated ground-station. The UPI is to allow the rapid coupling and decoupling of payloads to a vehicle. In order to support a large variety of payloads, the UPI must support USB, ethernet, I^2C, UART, and SPI devices. To maintain UAV flight safety and to ensure data validity, the UPI is to prevent circuit shorts and opens that may arise due to flight vibrations. Such a system will enable the owner of a UAV to rapidly reconfigure it to avoid the costs of additional application-tailored vehicles. Integrated with a UPI system, a single UAV can equipped and re-equipped with an infinite arrangement of sensors to fulfil a multitude of needs. Furthermore, this system can be applied to fully autonomous vehicles, allowing for the automated coupling of payloads, collection of data, and delivery of goods.

2 UPI Design

The novel UPI system encompasses three primary elements:

1. An offboard (payload-side) printed circuit board (PCB)
2. An onboard (vehicle-side) PCB
3. An onboard (vehicle-side) single board computer to handle payload traffic

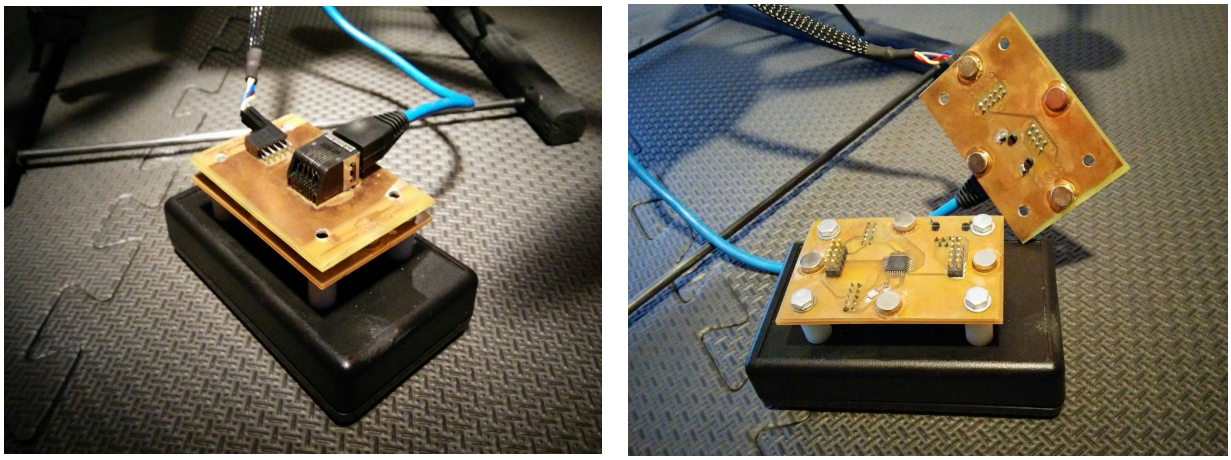


Figure 2.1: Payload interface mated (left) and detached (right)

In figure 2.1 above, elements 1 and 2 are shown. A sample payload enclosure is shown attached to the offboard UPI PCB. The offboard PCB contains an ATmega328P microcontroller to verify proper coupling to the onboard PCB, pass flight instructions to the onboard single board computer, and perform any payload-local processes. Payload electronics connect to this PCB via female headers and/or an RJ-45 jack on its underside.

Prior to flight, this offboard PCB mates to the onboard PCB using 8 permanent neodymium magnets (4 mounted to each board). These magnets are keyed (by polarization) such that they provide the proper alignment of the two PCBs. Furthermore, the magnets provide the mechanical coupling force necessary to affix the payload to the vehicle. Electrical connections between the two PCBs are made by spring-loaded connectors on the offboard PCB. When the PCBs are flight-mated, the force applied by the spring-loaded connectors on the onboard PCB is negligible in relation to the coupling force of the magnets. It is great enough, however, to ensure successful power and high-bandwidth data transfer in a vibratory flight environment.

The onboard PCB interfaces with the on-board computer, a Raspberry Pi. The Raspberry Pi receives flight instructions from the offboard ATmega328P, passes them to the vehicle's flight controller, and returns any data output by the flight controller to the payload. Using an attached USB WiFi dongle, the Raspberry Pi can also receive instructions from an optional ground-station and return flight data. In this fashion, the Raspberry Pi essentially serves as the UPI system's network router. Communications are facilitated using MAVLink, an open-source, header-based UAV communications protocol. Further descriptions of the hardware and software aspects of the system follow.

2.1 System Capabilities

The designed and fabricated UPI system supports the following operating features:

- UART/SPI/I^2C serial payload peripheral support
- 100 Mbit/s Ethernet payload peripheral support
- 5 V_{DC} payload power distribution
- Keyed magnetic coupling to prevent improper flight-mating
- Payload heartbeat message to continually monitor connectivity
- Payload control over flight controller inputs
- Optional ground-station control failover

Current iterations of the UPI design require human intervention for the mating and demating of payloads. See section 4.2 for possible future improvements.

2.2 Software & Firmware Overview

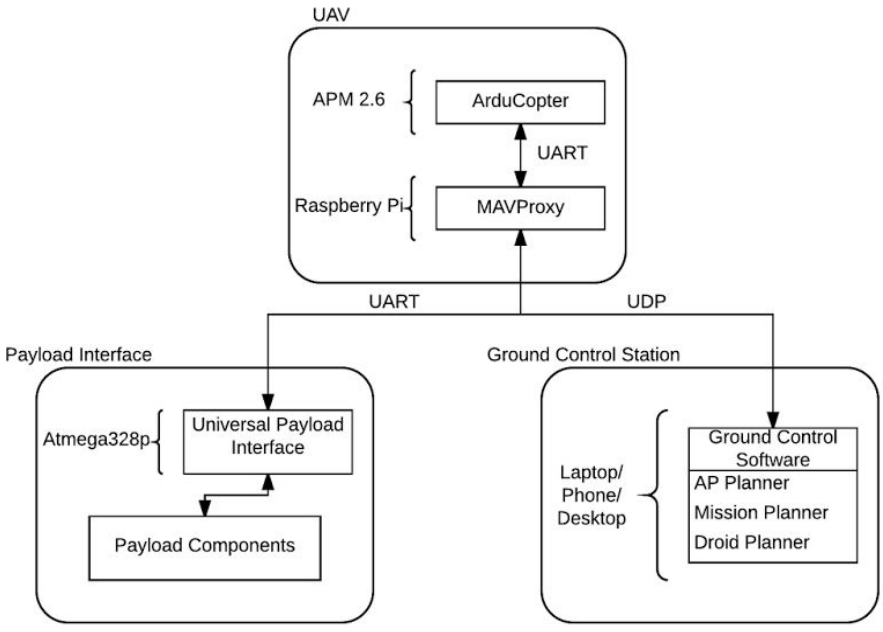


Figure 2.2: Software Block Diagram

2.2.1 Subsystem States

The firmware onboard the payload ATmega328P operates as a finite state machine. The firmware provides for transitions between disconnected, connected, waypoint upload, and visual acknowledgement states. The full state diagram is shown in the figure below.

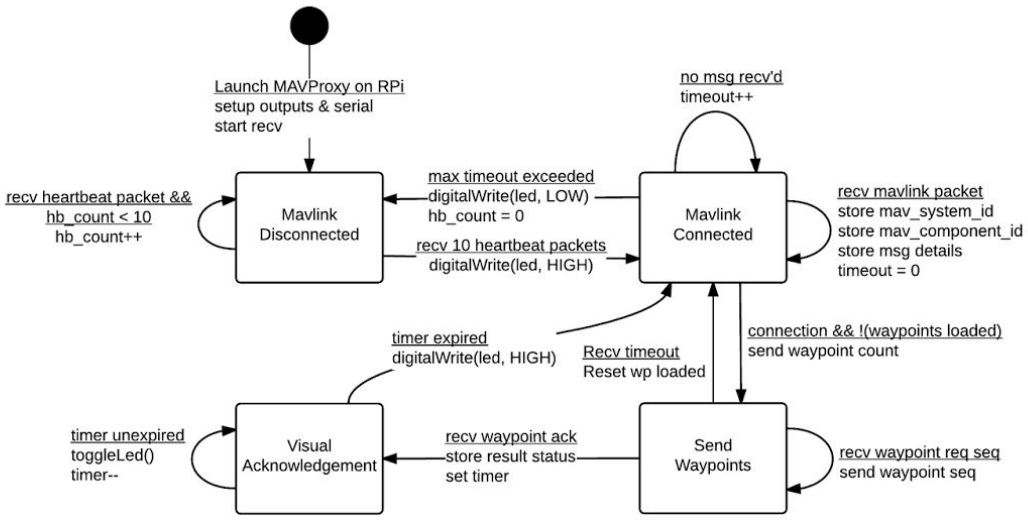


Figure 2.3: Payload ATmega328P firmware state diagram

When the system is first powered, the payload operates in the Mavlink Disconnected state. While in this state, the payload reads serial data from the autopilot until receiving 10 heartbeat messages. Once ten messages are successfully decoded by the payload, the LED signifying a connection is turned on and the system enters MAVLink Connected state.

While in Mavlink Connected state, the system reads serial data and parses MAVLink packets from the autopilot. If the system does not interpret a packet for a predefined maximum timeout, the LED signifying a connection is turned off and the state of the system is set back to disconnected (any other necessary clean-up also occurs, such as resetting the heartbeat counter). If a packet is received, the system simply stores the system id and component id of the sending autopilot. In this state, the system can read in and interpret any of the various status messages the autopilot sends periodically, but for the purposes of this demo application only identifiers and a handful of other status variables are stored. While in Mavlink Connected state, if the system has not yet successfully uploaded waypoints to the autopilot it will enter Send Waypoints state.

While in Send Waypoint state, the system initially sends a waypoint count message to inform the autopilot that a number of waypoints are about to be provided. The system then waits for a request for each waypoint up to the number of waypoints defined in the initial count message. For each received request, the system sends the waypoint details of that waypoint sequence number. Upon

completion, the system will receive a mission acknowledgement from the autopilot, at which time the system enters Visual Acknowledgement state.

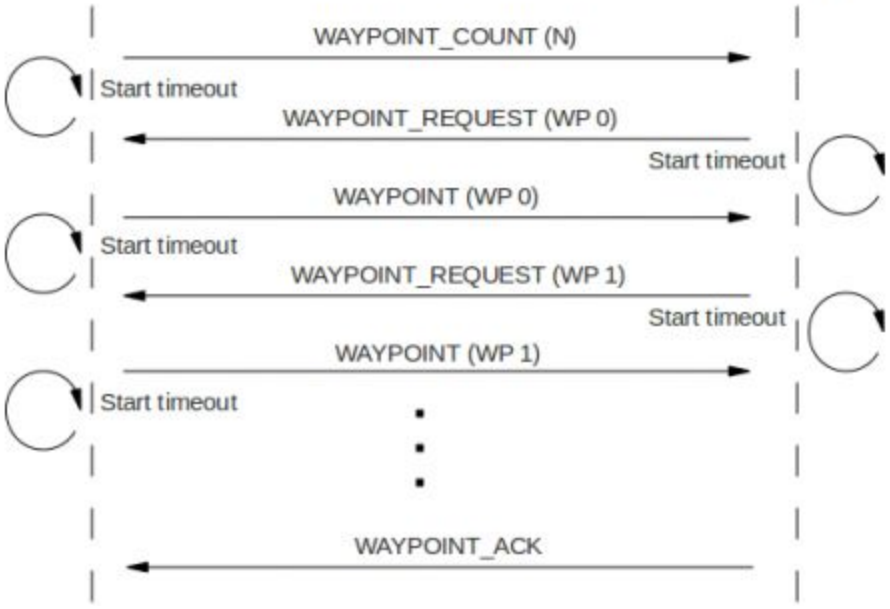


Figure 2.4: Waypoint sending protocol between a ground station & autopilot

The Visual Acknowledgement state simply acts as an external visual to signify to the user that waypoints have been successfully uploaded to the autopilot. In this state, the onboard LED is toggled for a short period until re-entering MAVLink Connected state and listening for more incoming messages.

Once the payload's mission waypoints have been successfully received by the autopilot, the ground control station can send a start mission command to run the mission.

2.2.2 MAVLink Protocol Overview

To maximize the potential of the UAV payload management platform, the payload communicates with the autopilot using the open source MAVLink protocol. MAVLink is a protocol for communicating with unmanned vehicles. The protocol is openly available as a C header library designed for packing structures necessary for communication between an unmanned vehicle, a ground control station, and any internal components to the unmanned vehicle.

Each MAVLink packet contains a header, a message, and a CRC trailer. The header contains a start of frame identifier, the message length, the packet sequence number, the system ID of the sending system, the component ID of the sending system, and the ID of the incoming message. The message varies depending on the message id. Common messages on all autopilots include heartbeat, command, and waypoint management messages, but the overall message set depends on the

autopilot in use. The trailing CRC uses a ITU X.25/SAE AS-4 hash of the bytes in the packet to confirm the integrity of the message.

Byte #	0	1	2	3	4	5	6	n + 5	n + 6	n + 7
Value	0xFE	Msg Len	Seq #	Sys ID	Comp ID	Msg ID	Payload		CRC	

Figure 2.5: MAVLink packet structure

The protocol is supported by an assortment of autopilots and ground control softwares including Ardupilot, Parrot AR, Pixhawk, QGroundControl, APM Planner, and more. By utilizing this protocol, the payload firmware can seamlessly interface with a wide variety of existing autopilot systems.

2.2.3 ATmega328P Payload Firmware

Including MAVLink Headers

For improved serial communication, common practice for MAVLink protocol on microcontrollers is to include an input driven serial library such as FastSerial. This file must be included before Arduino.h since it will be replacing the functionality of Arduino's HardwareSerial library.

When using a microcontroller with limited memory (such as the ATmega328P 8 MHz), the number of communication buffers allocated by MAVLink can be limited by defining MAVLINK_COMM_NUM_BUFFERS before including mavlink.h. If left undefined, MAVLink will attempt to utilize more memory than is available to the ATmega328P, causing the microcontroller to lock-up.

```
/* Make sure this is included before Arduino's HardwareSerial */
#include <FastSerial.h>

/* Make sure this is defined before including mavlink.h */
#define MAVLINK_COMM_NUM_BUFFERS 1
#include <mavlink.h>
```

Reading MAVLink Packets from APM

Messages can be received from an autopilot utilizing MAVLink protocol via the serial connection. To parse a MAVLink message, read in characters are passed into the mavlink_parse_char function. This function will return true when an incoming message successfully passes the CRC check and identifies as a MAVLink message. The message is then stored in the passed in mavlink_message_t struct pointer and can be handled based on its message ID. The code below also updates the system and component IDs of the autopilot based on the values in the received message header. These will be important when sending messages to the autopilot.

```
void comm_receive() {
        mavlink_message_t recv_msg;
        mavlink_status_t recv_status;

        // Receive data over serial
        while (Serial.available() > 0) {
                uint8_t c = Serial.read();

                // Try to get a new message
                if (mavlink_parse_char(0, c, &recv_msg, &recv_status)) {
                        // Update system and component id
                        mav_system_id = recv_msg.sysid;
                        mav_component_id = recv_msg.compid;

                        // Handle message
                        switch(recv_msg.msgid) {
                                case MAVLINK_MSG_ID_HEARTBEAT:
                                        handle_heartbeat(&recv_msg);
                                        break;
                                case MAVLINK_MSG_ID_MISSION_CURRENT:
                                        handle_mission_current(&recv_msg);
                                        break;
                                case MAVLINK_MSG_ID_MISSION_REQUEST:
                                        handle_mission_request(&recv_msg);
                                        break;
                                case MAVLINK_MSG_ID_MISSION_ACK:
                                        handle_mission_ack(&recv_msg);
                                        break;
                                default:
                                break;
                        }
                }
                delayMicroseconds(200);
        }
}
```

Decoding Received Messages

When handling received messages, the MAVLink header library provides convenience functions to extract elements from a message. Each message contains functions following the style `mavlink_msg_<message type>_get_<message element>(mavlink_message_t *msg)`. The library also provides general decode functions to decode a received mavlink_message_t into a specific message struct (such as mavlink_msg_heartbeat_t).

```
void handle_heartbeat(mavlink_message_t *msg) {
        hb_count++;
        mav_type = mavlink_msg_heartbeat_get_type(msg);
        mav_autopilot = mavlink_msg_heartbeat_get_autopilot(msg);
        mav_base_mode = mavlink_msg_heartbeat_get_base_mode(msg);
        mav_system_status = mavlink_msg_heartbeat_get_system_status(msg);
}
```

Sending MAVLink Packets to APM

In order for a sent message to successfully be read by the autopilot, the CRC trailer must match what the autopilot is expecting. This requires the identifiers in the header to be correctly set. To send a Set Waypoint message, the code below creates a local copy of the mavlink_mission_set_current_t message struct. The seq value of the struct is set to the desired waypoint, and the target_system and target_component values are set to that of the target autopilot as received in the receive sample above. MAVLink provides helper functions to encode a MAVLink message when given the message structure. Once encoded, the message can be written to serial.

```
void send_set_waypoint(uint16_t wp) {
        mavlink_message_t msg;
        mavlink_mission_set_current_t set_point;

        set_point.seq = wp;
        set_point.target_system = mav_system_id;
        set_point.target_component = mav_component_id;

        mavlink_msg_mission_set_current_encode(pl_system_id, pl_component_id, &msg,
&set_point);

        send_message(&msg);
}

void send_message(mavlink_message_t* msg) {
        uint8_t buf[MAVLINK_MAX_PACKET_LEN];

        uint16_t len = mavlink_msg_to_send_buffer(buf, msg);

        for(uint16_t i = 0; i < len; i++) {
                Serial.write(buf[i]);
        }
}
```

2.2.4 Raspberry Pi Payload Data Network

The MAVlink protocol is supported by several ground station applications allowing support for almost any platform. During testing APM Planner, QGroundControl, and DroidPlanner were able to connect to the UAV and see its status, any waypoints or commands sent by the payload, and could arm and disarm the UAV for flight. Communications between the payload, flight controller, and ground station are all routed through a Raspberry Pi B+ onboard the test UAV via a Python application called MAVproxy. This application allows routing of multiple MAVlink data streams between the flight controller and various endpoints. In addition, this allows a payload to use either UART or ethernet/IP to communicate with the flight controller enabling wide ranges of payload sophistication.

Connection to the ground station while in flight is achieved with a dual band 802.11n USB WiFi adapter, specifically an ASUS N53 which is based on a Ralink RT2800 chipset and has a rated output power of 16.5 ~19.5 dBm (40-80mW). This provides a direct line-of-sight wireless link with devices such as phones and tablets to allow very portable ground station deployments as well as enabling high speed data communication to the payload with a simple IP interface through the Raspberry Pi.

2.3 Hardware Overview

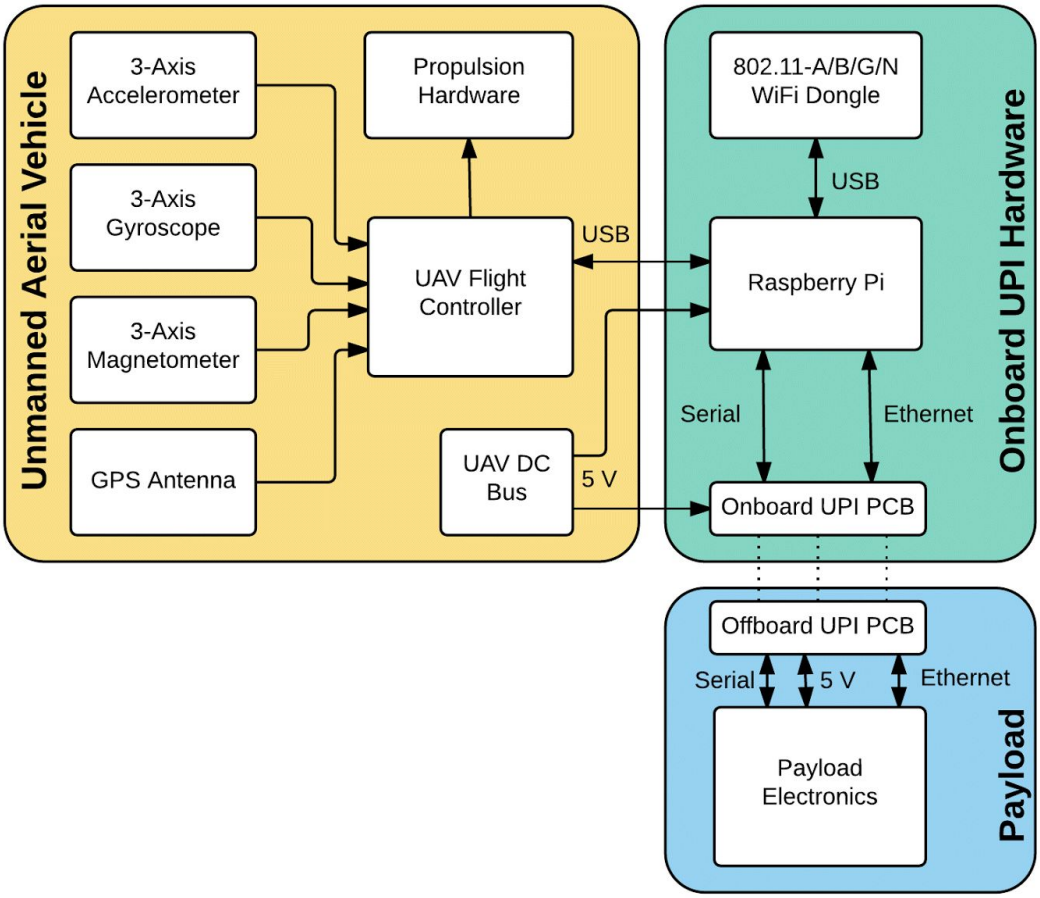


Figure 2.6: Hardware level 1 block diagram

2.3.1 PCB Design

Two double-sided complementary PCBs comprise the electromechanical interface between the UAV and payload. These PCBs are custom designed using the freely available KiCAD suite of EDA tools. Once designed, prototype PCBs were produced using a home photo fabrication process that enabled quick turnaround of revisions. In order to avoid unnecessary vias that are difficult to manufacture by hand, the PCB design is optimized so that all traces are routed on a single side, and the only connections between layers are through the through-hole components. Both PCBs can be manufactured using only a single sided PCB if desired making it more accessible to the hobbyist UAV community.

The offboard (payload-side) PCB incorporates an Atmel ATmega328P microprocessor in a surface mount 32TQFP package. This is the same processor used in the popular Arduino development boards and enables the simple IDE and existing software to be used directly on the payload board with no additional hardware. The processor's flash memory can be programmed on the ground via the pin

headers accessible on the backside, or while seated on the UAV through with the Raspberry Pi acting as the programmer enabling over-the-air updates of payload configuration.

Several key characteristics of the PCB designs are optimized for use as a mating pair. Copper pads with thermal reliefs are positioned and aligned on both sides of the interface for easy placement and attachment of the alignment magnets. The spring-loaded pins contain redundant power connections to increase current carrying capability, as well as ensuring constant power availability even during bumps or other physical interruptions to prevent the resetting of payload electronics. An LED attached to the payload side microcontroller provides a visual indication of successful PCB mating and is programmed to indicate the successful receipt and transmission of messages with the UAV.

The primary mechanical attachment mechanism consists of four 8mm diameter neodymium magnets mounted directly to each PCB. They are arranged such that the board will not attach when rotated 180° from the correct orientation. The magnets provide solid physical support to small payloads and facilitate alignment of the spring-loaded pins used for data and power connections. These magnets can be supplemented with surrounding attachment mechanisms to accommodate larger and heavier payloads.

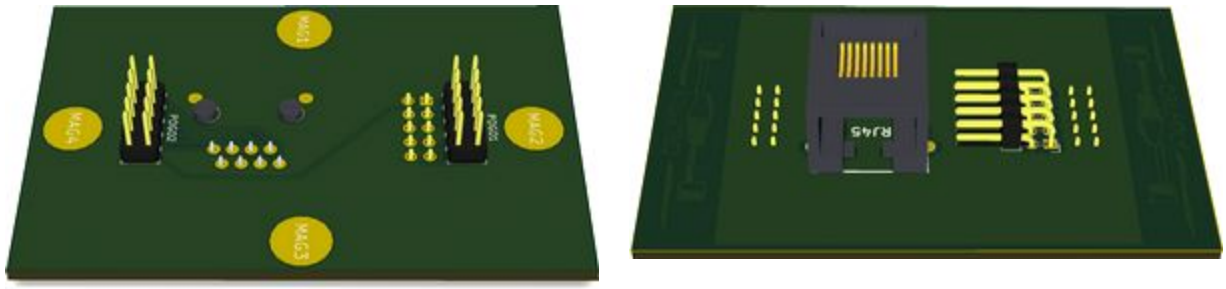


Figure 2.7: Onboard PCB rendering

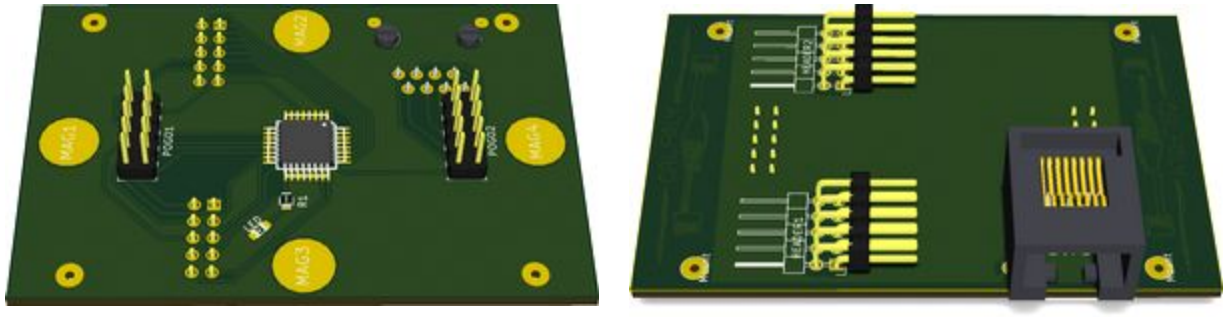


Figure 2.8: Offboard PCB rendering

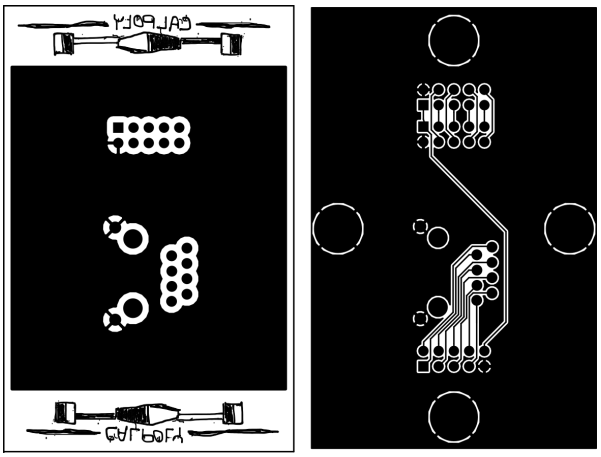


Figure 2.9: Onboard PCB copper mask

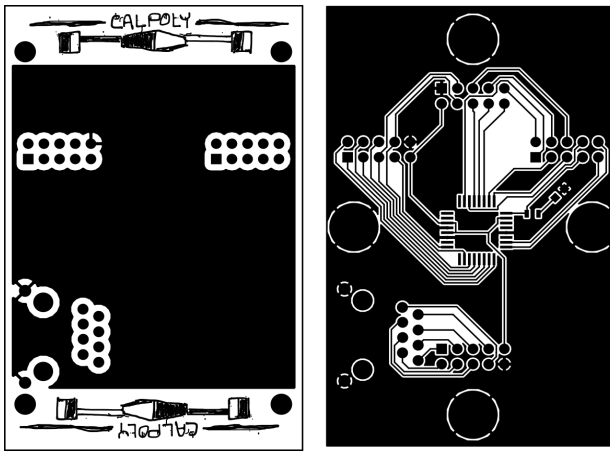


Figure 2.10: Offboard PCB copper mask

2.3.2 PCB Fabrication

Both sides of the payload interface PCBs were manufactured at home using a small scale photofabrication process that is common among hobbyists. This process starts with the copper mask printed on plastic transparency film. This is used as an exposure mask to a photosensitive material affixed to a copper-clad substrate. The photosensitive etch-resist material can be purchased as a film to be laminated onto copper clad board, or already applied to a presensitized copper board. After developing the resist in sodium carbonate, the copper is selectively removed in a heated ferric chloride solution. The whole process takes about 30 minutes for a two sided PCB, and has a minimum feature size of about 0.1mm which was determined through several trials of fabrication. Solder mask film can be optionally applied to the etched PCBs similarly to the resist material to give them a typical green finish, prevent surface tarnish, and control solder flow.

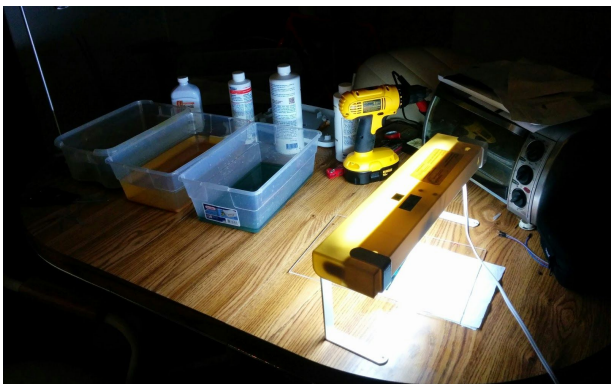


Figure 2.11: Home PCB etching station

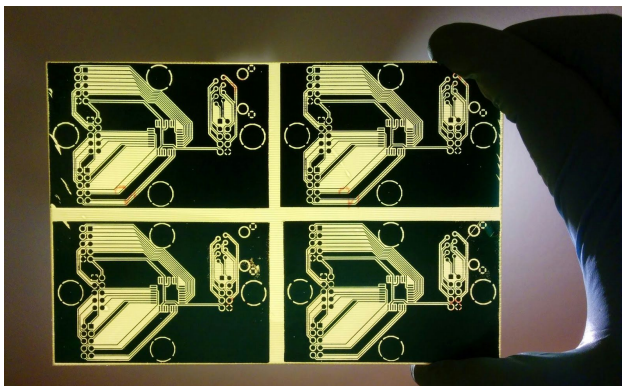


Figure 2.12: First PCB batch ready for drilling

3 Testing Methodology

3.1 Test Vehicle

A flight-test UAV was constructed for this project to verify UPI system efficacy. The flight controller hardware for this vehicle was required to be ubiquitous in use while supporting a host of autonomous features. For this reason, the Ardupilot Mega (APM 2.6) was chosen. Its award winning Ardupilot control software is open source. As such, the headaches of communication with external electronics are reduced. Furthermore, the APM 2.6 is widely used by hobbyists and researchers alike. Its support of the standardized MAVLink communications protocol ensured UPI system compatibility with a multitude of other flight controllers.

Testing of the UPI's instruction passing ability included carrying out a GPS-based flight plan. An onboard GPS receiver closes this loop and allows the APM to autonomously navigate to payload-specified waypoints. To avoid designing the complex mechanical controls elements of a single-rotor UAV, a quadrotor was chosen instead. 3 kW brushless DC motors and 50 A motor controllers were chosen to accommodate payloads in excess of 10 lbs.

All in all, the assembled test vehicle was constructed solely from commercial off-the-shelf (COTS) parts that are widely available to and commonly used by unmanned flight hobbyists.

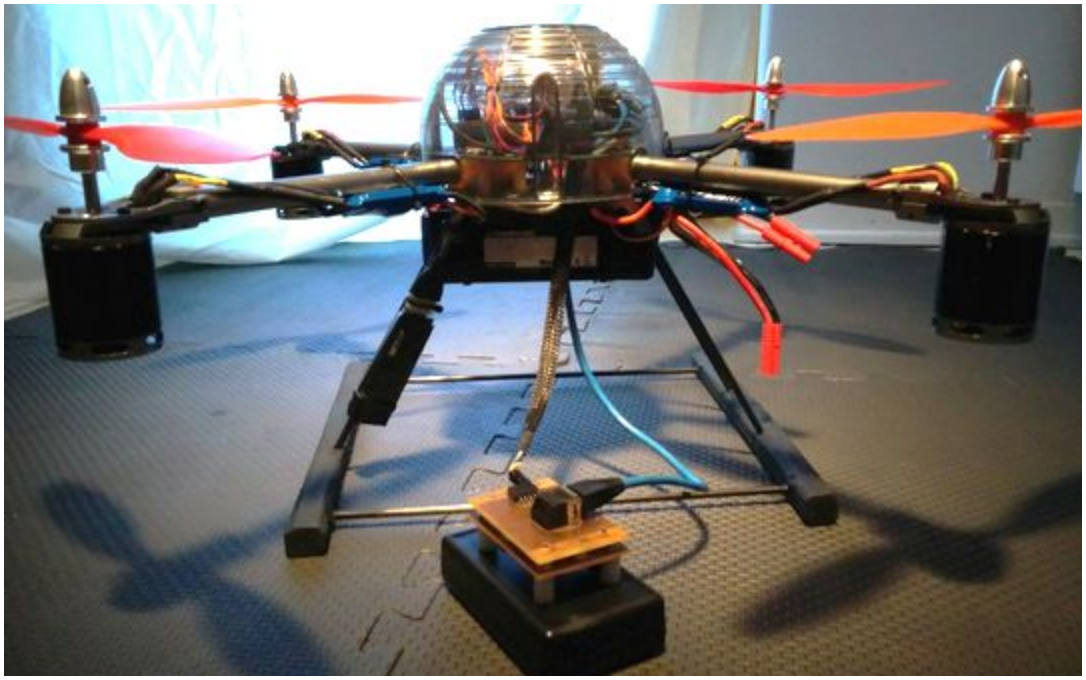


Figure 3.1: Test quadrotor with UPI tethered for communication verification

3.2 Early Flight Tests

Initial testing with the quadcopter focused on becoming familiar with the limitations and functionality of the APM 2.6 flight controller, Raspberry Pi, and WiFi link to the ground station. Both manual and autonomous flight modes were tested to ensure reliability and prove that MAVlink in conjunction with the Raspberry Pi and WiFi link would be adequate to achieve the desired command, control, and data acquisition capabilities. Multiple flights proved the flight controller and data link are very capable with the only failures occurring due to faulty motor controllers and shutdown due to a low battery. This testing stage consumed two full sets (8) propellers.

3.3 Interface Functionality Testing

Throughout the software and hardware development process tests were conducted for mating alignment and electrical connection integrity. The magnets used for alignment have proven to be sufficiently strong to force the pogo pins into alignment even if coupling occurs from an off-center starting point.

Debug output from MAVProxy and sniffing the serial bus between the payload and the Raspberry Pi allowed for confirmation that messages were getting from the flight controller to the payload and back, and that they were being properly understood. This is also verified visually through the use of a status LED on the payload itself.

For further tests a functional example payload was created consisting of just the on-PCB ATmega328P programmed to send waypoints to the UAV for flight. This payload has a small plastic project box attached to it allowing future interface stress testing with varying weights in flight, in addition to providing a mounting platform for test sensors. This payload was used to demonstrate the functionality of the onboard ethernet connection between the mated PCBs, which negotiated up to 100 Mbit/sec successfully upon connection and worked to full capacity under speed tests (roughly 95 Mbit/sec real throughput). While both test endpoints were capable of 1Gbit/sec, the capacitance of the PCBs and pogo pins prevented negotiation up to this speed. This is not a very big issue as the WiFi link shares the same bus on the Raspberry Pi and will thus be limited to slightly less than 100 Mbit/sec from payload to ground in realistic scenarios.

4 Conclusion & Future Work

UAVs are becoming more and more pervasive in everyday life, and this trend is only accelerating. There is no shortage of useful and innovative applications for these flying robots, and by enabling them to be much more versatile, accessible, and universal we're paving the way for rapid deployment of new and life changing technologies.

4.1 Encountered Challenges

PCB photofabrication proved to be the most difficult part of the development process. Small mistakes at any point in the process can make the resulting PCB unusable and this meant some degree of trial and error was necessary to produce several working prototype revisions of the interface. Drilling sub-millimeter holes in the PCBs also proved challenging and was prone to the breaking of drill bits.

While this project would not have been possible without the hobbyist community and widespread availability of UAV components, the vast majority of these parts are very poorly documented and have either misleading or no technical specifications at all. Dimensions, power ratings, interfaces, and mechanical drawings are few and far between with a lot of what we encountered proving inaccurate or simply incomplete. This is one area where it becomes clear the majority of hobbyist UAV component consumers are not engineers, which is exemplified by the next development hurdle: poor quality parts.

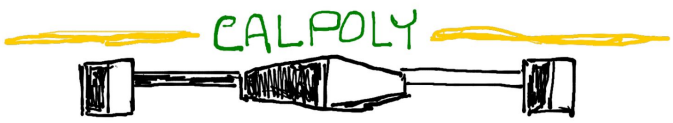
Most of the hobbyist UAV components on the market come from China and are poorly designed or quality controlled. Finding "deals" on components online led to four failed motor controllers, four complete sets of unstable propellers, and a broken frame. We ended up buying most critical components from a local hobby shop that has so far proved to stock much more reliable components.

On the software side of the project, contradictory and fragmented documentation made finding and adapting working versions of all of the three major software components proved difficult. This is an area where some community support could be given back by contributing well documented working code.

4.2 Improvements

The software, mechanical, and electrical elements of the interface comprise a working interface with great prospects for future extensibility and improvements. First, the coupling and decoupling processes could be made automatic, allowing payloads to be attached and dropped on command or according to a flight plan. The next logical progression from there is to implement an autonomous attachment system so remote payloads could be picked up and dropped off without human intervention. This would enable very rapid deployment of large area sensor networks with applications reaching from agriculture, to search and rescue, to surveying. The payload interface thus far has been designed with this eventual goal in mind. The onboard Raspberry Pi has adequate computational power to attempt computer-vision alignment and attachment to the payload, as well as the ability to add radio based location systems.

While this revision of the payload interface only provides 5V DC power to payloads, high current zero-insertion-force (ZIF) connectors are commercially available that would enable flight batteries to be added to payloads. This could extend the range of the UAV potentially indefinitely, and allow frequent autonomous survey operations to be conducted with the UAV swapping its own batteries. More electrical protection and battery management hardware would need to be added to ensure high current shorts do not occur while the interface is being connected.



References

Datasheets

Atmel, “ATMEL 8-BIT MICROCONTROLLER WITH 4/8/16/32KBYTES IN-SYSTEM PROGRAMMABLE FLASH,” ATmega48A/PA/88A/PA/168A/PA/328/P datasheet, 2014. Available: http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf. [Accessed: 12- June- 2015].

Open source documentation

Qgroundcontrol.org, 'MAVLink Micro Air Vehicle Communication Protocol - QGroundControl GCS', 2015. [Online]. Available: http://qgroundcontrol.org/mavlink/start. [Accessed: 12- June- 2015].

A. Tridgell and S. Dade, 'MAVProxy', Tridge.github.io, 2015. [Online]. Available: http://tridge.github.io/MAVProxy. [Accessed: 12- Jun- 2015].

Copter.ardupilot.com, 'Copter | Multirotor UAV', 2015. [Online]. Available: http://copter.ardupilot.com. [Accessed: 12- Jun- 2015].

Software

L. Meier, MAVLink Micro Air Vehicle Protocol. MAVLink, 2009. Available: https://github.com/mavlink/mavlink. [Accessed: 12- Jun- 2015].

A. Tridgell, MAVProxy A UAV ground station software package for MAVLink based systems. MAVProxy. Available: https://github.com/Dronecode/MAVProxy. [Accessed: 12- Jun- 2015].

B. Bonney, APM Planner. DIYDrones, 2015. Available: https://github.com/diydrones/apm_planner. [Accessed: 12- Jun- 2015].

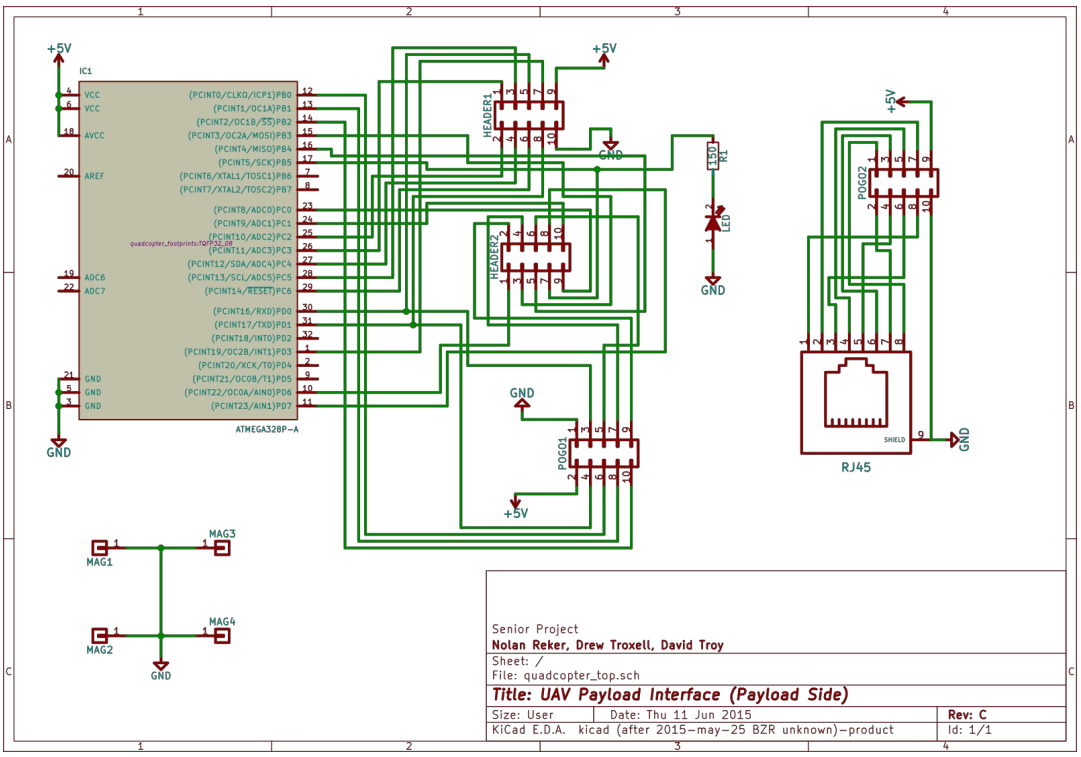
Appendix A: Software & Hardware Sources

For easier access and future development, the KiCad project files and payload ATmega328P source files are hosted at: https://github.com/troxellophilus/universalpayloadinterface

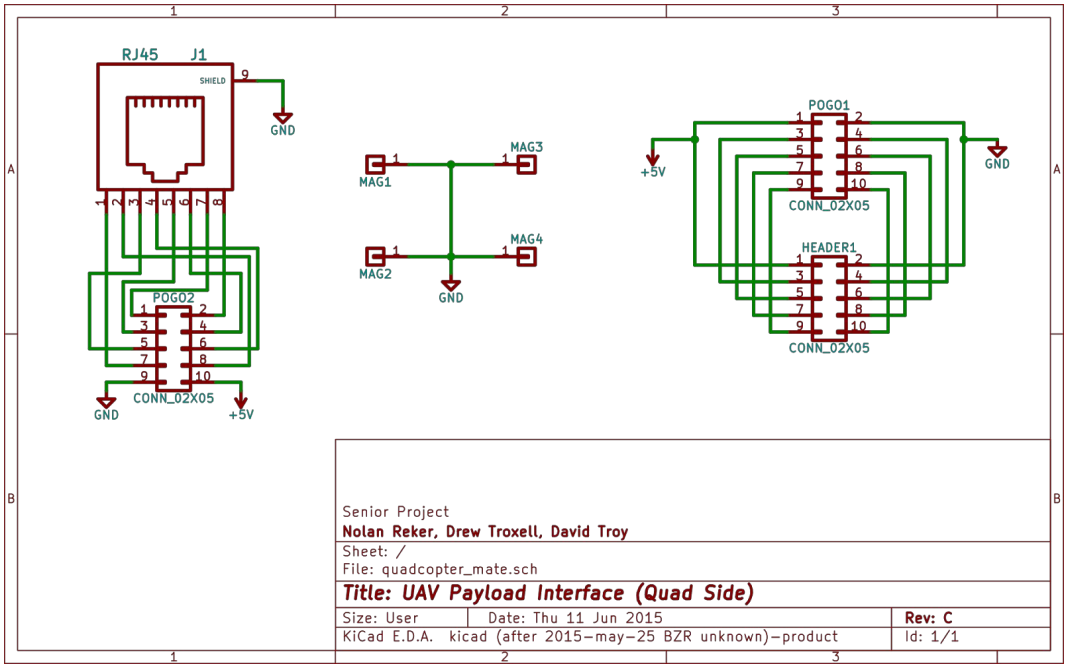
Appendix B: Bill of Materials

Vendor P/N	Item	Vendor	Qty	Unit Cost
68X0155	Raspberry-Pi B+ 512 MB	Newark	1	$35.00
ED8137-10-ND	10 POS Spring Connector	Digi-Key	4	$8.40
ATMEGA328P-AU-ND	32TQFP ATmega328P	Digi-Key	2	$3.61
S5519-ND	10 POS Female Header	Digi-Key	6	$1.10
A31438-ND	Shielded RJ-45 Jack	Digi-Key	4	$1.82
P150CCT-ND	150 Ω 0805 Resistor	Digi-Key	2	$0.10
B008UH45EW	Presensitized Copper-Clad Board	Amazon	2	$18.75
B005SAKW9G	Asus 802.11-A/B/G/N WiFi Dongle	Amazon	1	$40.80
2701801	Project Enclosure	Radioshack	2	$4.49
2760322	1.8 MM Green LED	Radioshack	2	$1.99
ES-02-8mm	8 MM Wire Mesh Cable Guard	HobbyKing	1	$0.51
11260	1 Ft CAT5e Ethernet Cable	Monoprice	2	$0.60
095421070459	10-Pk Neodymium Magnets 0.3”x0.11”	Home Depot	2	$3.98
887480149586	1”x3/8” Nylon Standoff	Home Depot	8	$0.72
887480037388	4-Pk M4 Nut	Home Depot	4	$0.37
887480035988	4-Pk M4 Lock Washer	Home Depot	4	$0.43
887480035285	4-Pk M4 Flat Washer	Home Depot	4	$0.45
887480012781	4-Pk M4 30 MM Bolt	Home Depot	2	$0.60
887480012385	4-Pk M4 12 MM Bolt	Home Depot	2	$0.57
		Total Cost:		**$203.93**

Appendix C: Electrical Schematics



Offboard PCB schematic



Onboard PCB schematic