6-9-2015

Pilot-1: autonomous fixed-wing aircraft control system

Chris Millsap
Santa Clara University

Nathan Garvey
Santa Clara University

Faisal Hayat
Santa Clara University

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING
DEPARTMENT OF ELECTRICAL ENGINEERING

Date: June 5, 2015

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Chris Millsap
Nathan Garvey
Faisal Hayat

ENTITLED

Pilot-1: Autonomous Fixed-Wing Aircraft Control System

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREES OF

BACHELOR OF SCIENCE IN COMPUTER ENGINEERING
BACHELOR OF SCIENCE IN ELECTRICAL ENGINEERING

[illegible]
Dan Lewis

[illegible]
Sarah Kate Wilson

[illegible]
Department Chair

[illegible]
Department Chair

Pilot-1: Autonomous Fixed-Wing Aircraft Control System

by

Chris Millsap
Nathan Garvey
Faisal Hayat

Submitted in partial fulfillment of the requirements
for the degrees of
Bachelor of Science in Computer Engineering
Bachelor of Science in Electrical Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 9, 2015

Pilot-1: Autonomous Fixed-Wing Aircraft Control System

Chris Millsap
Nathan Garvey
Faisal Hayat

Department of Computer Engineering
Department of Electrical Engineering
Santa Clara University
June 9, 2015

ABSTRACT

In the past decade, the personal ownership of unmanned aerial vehicles has exploded in the US and around the world. The rapidly declining size and cost of integrated circuits, sensors and embedded micro-controllers has lead to a flourishing community of hobbyists designing flight controllers with levels of sophistication approaching those for government and military applications. The typical flight assisted controllers integrate data from the user's control system and an Inertial Measurement Unit (IMU) in order to keep the craft level and on course. Deriving mostly from the radio controlled (RC) hobby industry, the flight control technologies for both rotary and fixed wing systems are generally community-built and open source. While this leads to rapid development and ease of modification, quality usually suffers. Because the community is not a community of professionals, best coding practices are often left behind, leading to unexpected failures. Such flight control systems are unsuitable for integration into US airspace due to their failure-prone nature and inability to mitigate the failure of flight control surfaces. Fixed wing systems can also be controlled without an onboard flight controller or autopilot, with a simple camera downlink and direct control surface control being sufficient for most first-person video (FPV) needs. This has left a hole in the market for such controllers, with all offerings lacking in professional features such as redundancy and failure mitigation. Our project suffered from many setbacks, including one team member becoming ill and another leaving the project halfway into development. We were also hampered by our choice to use the brand new STM32Cube Hardware Abstraction Layer (described in more detail in Chapter 9), as 3rd party support and example code was nonexistent or conflicting with official documentation. As such, we were required to greatly reduce the featureset of our system. To compensate for this, we implemented a basic software plugin system, where future developers can code their own flight modes and recompile the software without having to revisit the basic I/O required to setup and access the onboard sensors and actuators. Looking forward, we believe we have solved many of the issues involved with developing a flight controller on this powerful, next-gen platform. We have implemented basic IMU-based flight stabilization and control, and future developers can easily code in the more premium features, such as GPS and telemetry, on top of the existing foundation.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

With the emerging technology of very complex microcontrollers and small, accurate, low-power sensors becoming available for less and less money, the devices necessary to bring a high-quality autonomous aircraft to the market is now a reality. We believe that providing an integrated solution that is easy to manufacture, cost effective and highly reliable, will lead to the wide popularization of these vehicles. Currently we see no clear leader in this space. Our opportunity to make a contribution to the field parallels the way personal computers emerged in the late 1970s with the introduction of the Apple II. We see commonality with this previous generation of technology with the hobbyist community and the ametuer computer clubs of the late 60s. Back then, nobody really knew what to do with computers. All the software was opensource and unprofessional, and each was custom built to the creators specific vision, bringing with it inherent difficulties that proved distasteful or too complex for ordinary people to understand. Possible ways an automated aircraft technology such as the Pilot-1 system could contribute to present needs are gathering of scientific data from remote locations such as measuring the rate of glacial melt, surveying remote lands for climate change or geological activity, observing wildlife in an unobtrusive way, performing crop and animal population studies and performing search missions for mission people. Other exciting uses include rapid package delivery and transport, security and surveillance and low cost terrain and road mapping for further development of GPS navigation systems. All of the aforementioned applications currently require a piloted aircraft, which can be extremely costly. To bring autonomous fixed-wing vehicle technology into the mainstream, a standardization across the board of technologies with high levels of integration, safety and friendly user experiences is needed.

1.2 Solution

Our solution is a fully featured fixed wing autopilot designed and programmed to aerospace quality specifications. It is plug-and-play ready for any fixed wing airframe, to be chosen by the client. Unlike all other controllers on the market today, it includes failure detection and malfunction mitigation. For example, in the event of loss of RC control, the system will begin a controlled glide, using available techniques (flaps or cross-controlled[1] ailerons and elevators) to reduce the speed of the craft. The systems architecture allows for expansion, such that it may control a hobby grade 2-meter airplane as readily as it controls a 20-meter military surveillance drone. Control is available through standardized expansion busses; for example, a hobbyist may simply use an RC transmitter for real-time control, whereas a corporate user may opt for a satellite uplink modem. With to-be-written software expansion, the user may be able to use an interactive program to determine flight waypoints for autonomous flights, as well as configuring such systems settings as failure modes. It may also interface with ground based networks to perform routing and collision avoidance with other aircraft, both manned and unmanned. This makes our product an attractive offering on the market, as users can integrate it into their airframes without modification to existing control actuators and motors. Our system is a professional grade autopilot available to both corporate, government and private users and, as such, we expect it to be highly profitable.

[1]Cross-controlled: A technique used in aviation where opposite aileron and rudder commands (i.e. left aileron, right rudder) are given to reduce the speed of an aircraft when flaps are not available.

Chapter 2

System Overview

By definition, an autopilot is a system which controls the trajectory of a vehicle without constant "hands-on" adjustment by a human operator[1]. Autopilots are not intended to replace the need for a human operator, but rather assist them in maintaining vehicle velocity, stability and heading, allowing the operator to focus on larger tasks that may be required. Our system is the foundation of a true and robust autopilot system, and begins by implementing basic attitude control and a manual passthrough mode allowing for electronically selecting between modes of operation. The system is composed of 4 major components: The inertial measurement unit (IMU/MPU), Barometric pressure sensor (BARO), a microcontroller, and non-volatile memory for datalogging. Other features such

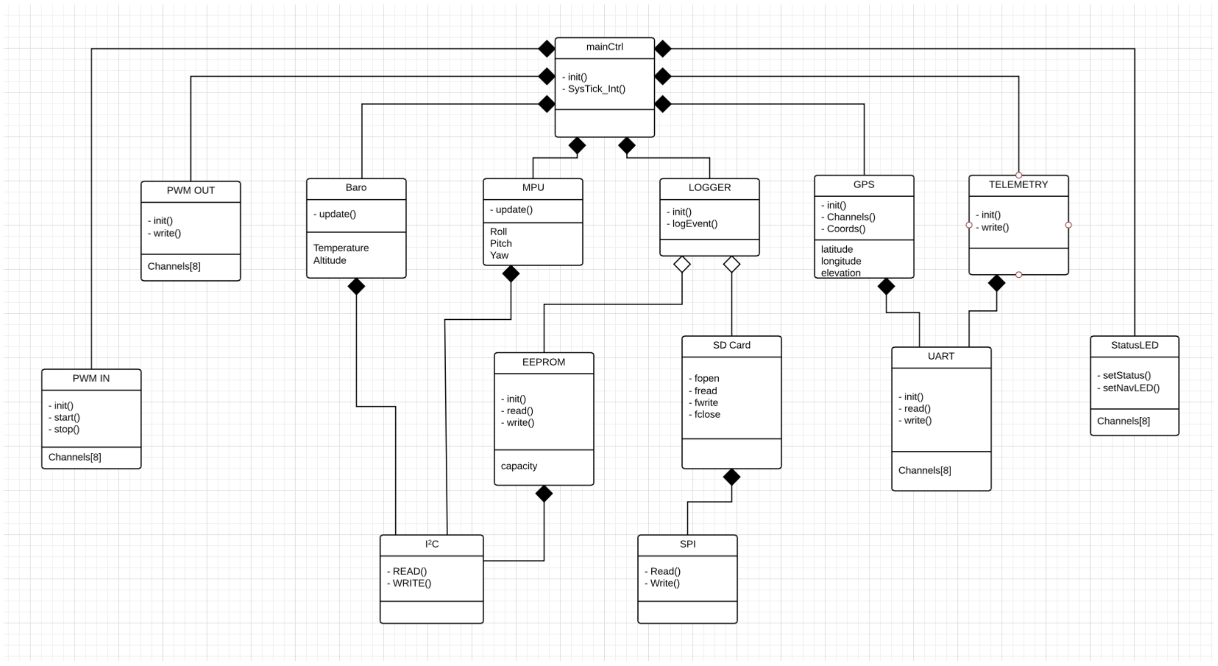

Figure 2.1: System Block Diagram of the Complete Pilot-1 Implimentation

[1]http://en.wikipedia.org/wiki/Autopilot

as GPS have not been included, as attitude control was the main focus for this thesis. Further development is needed to include all the parts included in the block diagram in figure 2.1. This figure includes all components needed in high level to realize the Pilot-1 and continues development.

Chapter 3

Vehicle

The Pilot-1 flight controller is designed to support a wide variety of hobbyist standard, fixed-wing aircraft ranging from a 1m wingspan to 20m wingspan. Some of the standard features provided in these types of aircraft are three-line signal, power and ground wires with 0.1" center pinheads for servo/speed control to interface with the receiver and Pilot-1 system. Additionally, most model aircraft come with a Battery Eliminator Circuit (BEC), (figure 3.2) which allows for a single battery to provide power to the motor while breaking out a separate line of typically 5V for servo and auxiliary system power.

3.1 The Sky Surfer

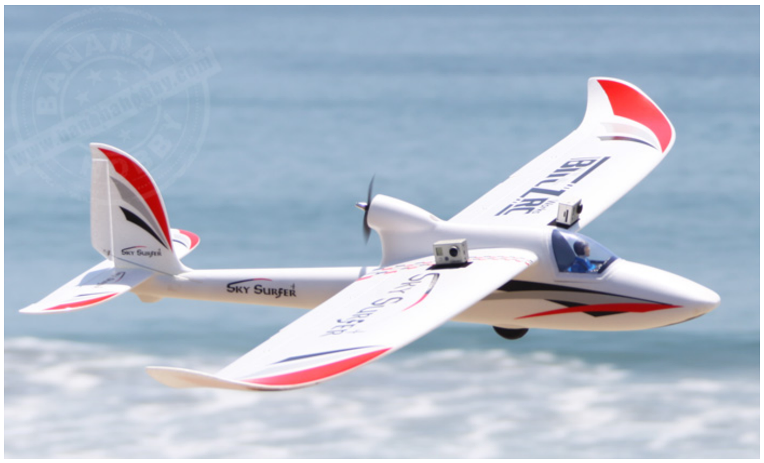

Figure 3.1: The Sky Surfer From Banana Hobby

Our selection of aircraft is the Sky surfer from Banana Hobby. The aircraft is made of a solid lightweight Expanded PolyOlefin (EPO) foam that is strong enough to support the aircraft and its payload, but is also a safe, lightweight material that is less hazardous than wood, fiberglass or carbon fiber in instances of unintentional hard landings. The SkySurfer is perfect for our application because of its large canopy and hull allowing for expansion and addition of our custom components to the interior of the aircraft. It also provides plenty of power in the form of a 3S Li-Po battery rated at 11.1V, 3600mAh, 25C[1] discharge rating. The aircraft is full four channel allowing for Aileron, Elevator, Rudder and Throttle to be controlled independently, giving us the ability to control the aircraft in unorthodox ways in the case of certain failures. The aircraft comes 95% assembled as an RTF, requiring only the installation of our flight controller, the servos and construction of the wing and push-rod assemblies for control surfaces.

3.2 Other Supported Aircraft

In general, all aircraft can be supported by the Pilot-1 flight controller with no or some modification necessary. The current prototype design of the Pilot-1 is best suited for any side model aircraft using 0.1" center, three-pin control wires and a standard 2.4 Ghz receiver/transmitter. As long as the aircraft is large enough to accommodate the 2.5" X 3.5" footprint of the main board, and the aircraft has or can be equipped with a battery eliminator circuit, such as the one in fugure 3.2, the

Figure 3.2: Battery Eliminator Circuit Used for 5V System Power

Pilot-1 will work.

[1]The C rating referred to here is the ratio of discharge rate over charge rate. Meaning this particular battery can safely discharge 25 times faster than it can charge.

Chapter 4

Requirements

Our system requirements, both functional and nonfunctional, were determined by examination of the feature-sets of existing flight controllers, both fixed wing and multirotor. In addition, experienced pilots were interviewed at Baylands park in Sunnyvale. Our requirements were tailored to improve upon what existing offerings are doing well, and patch holes in existing feature-sets, implementing features that pilots would like to have, but do not have access to. Lastly, requirements were added to make the system more appealing to industrial clients seeking a turnkey system. The functional requirements, section 4.1, define the tasks that the system must perform. The non-functional requirements, section 4.2, describe the manner in which the system must perform these tasks.

4.1 Functional Requirements

- The system must offer the flight control modes as described in chapter 6. The user must be able to switch between these flight modes, or a predefined subset of these flight modes, via a switch on the control system or a setting on the ground station.
- The system must stabilize the craft in such a way that it will prevent oscillations in the roll, pitch and yaw axis, which can range in severity from an annoyance to a danger to the craft.
- The system must receive commands from a standard RC receiver using both the Pulse Width Modulation (PWM) and Pulse Position Modulation (PPM) protocols.
- The system must control the control surfaces of the aircraft using standard PWM-controlled RC servos.
- The system must transmit telemetry data via a UART port. The medium over which this data is transmitted after it has left the flight control board is undefined.

- The system must provide stable operation on a range of aircraft sizes from 1 meter in wingspan to 30+ meters in wingspan.
- The systems parameters, GPS waypoints and operational modes must be tunable via a USB port.
- The system must detect failures of the controller, Inertiaql Measurement Unit (IMU), GPS and control surfaces and use equipment such as airbrakes, parachutes and flaps if installed on the aircraft, to provide control to the user and minimize the kinetic energy of the craft when it touches down.
- When operating in any semi or fully autonomous modes (see modes 2-6 in section 7.1), the system must manage the throttle to ensure the airspeed does not fall below the aircraft stall speed, as defined by the user.
- The system should, when operating in modes 2-6, manage the throttle to minimize power expended per mile traveled. After the system has been powered on, it must test all sensors, inputs and outputs. It must refuse to take off and alert the user if any fatal anomalies are detected.
- The system must log all user inputs and sensor values to an onboard, non-user-accessible memory such as a black box.

4.2 Non-Functional Requiremets

- The system should be provided in a durable, user friendly package.
- The system should exhibit a plug-and-play functionality for aircrafts of a 2 meter, aileron-elevator-throttle-rudder-flap configuration. Such aircraft include the Penguin from FinWing and the Super SkySurfer from Banana Hobby.
- The computer interface and configuration program should be user friendly and easily accessible to personnel without knowledge of industrial control tuning.
- Often-accessed ports, such as the USB port for configuration, should be easily accessible. The retail product should be bundled with an extension cable to allow the user to embed this port in the side of the airframe, allowing access without opening an access hatch.
- All user-accessible ports should have unique, keyed plugs to prevent the accidental insertion of an unsuitable accessory into a port.

- Where possible, the system should communicate on standard protocols, to allow interoperability with existing systems and accessories.
- When on the ground, the system should express its state using a system that can be easily read from a short distance, such as audio tones.
- The system shall offer several expansion ports. These will be used in conjunction with a future software update to enable industrial-grade functionality such as CAN-BUS enabled actuators and aircraft position reporting transponders.
- To ease the ability of the system to conform with future FAA regulations regarding autonomous flight, the hardware and software must be implemented to aerospace industry standards. In the event that full conformance with these standards is too costly and time consuming, implementation will follow all standards and recommendations possible within the allotted time frame.

Chapter 5

Hardware

5.1 Hardware Component Overview

Pilot-1 is built around a high-speed ARM Cortex M4 processor, specifically the STM32F407 processor from STMicroelectronics. This processor offers 32 bit performance at over 180 MHz. In addition, hardware implementation of I^{2}C (Inter-Integrated Circuit), SPI (Serial Peripheral Interface), Timer and UART (Universal Asynchronous Receiver Transmitter) peripherals equates to high-speed, asynchronous communication with system sensors and user inputs and outputs. Our system is implemented on specialized printed circuit boards, which allows for flexibility depending on the users needs and physical space constraints. The hardware goals are for a one size fits all application where the user can easily adapt the system to their particular application without extensive modification. the hardware selected allows for this ease of adaptability by using a small footprint, industry standard communication protocols and standard hardware interfaces for peripheral expandability.

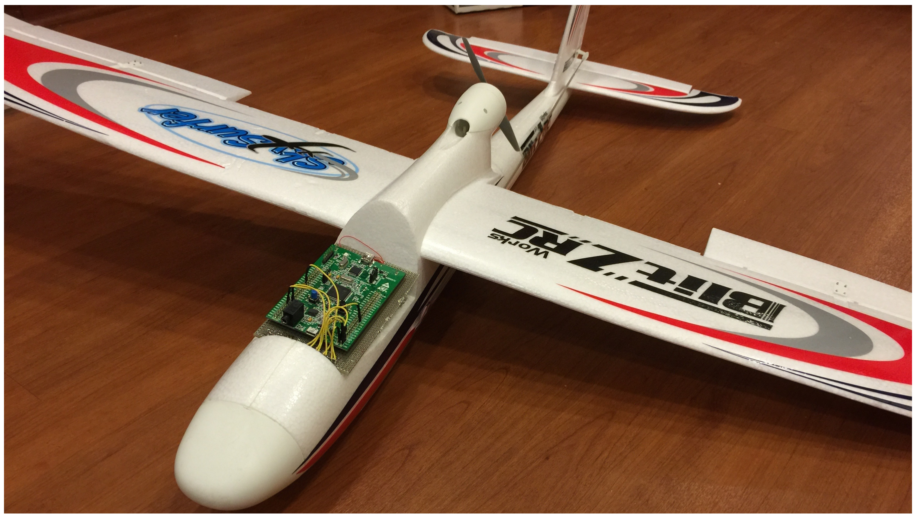

Figure 5.1: Pilot-1 Prototype Installed

5.2 Microcontroller Protocols

The STM32F407 provides enough horsepower for supporting up to 16 peripheral devices on a single I^{2}C bus, as well as additional USART I/O for serial to parallel applications such as sending and receiving GPS data, controlling payload devices and handling live and stored video. Some of the technology utilized in the development of this platform is explained in detail below.

Pulse-Width Modulation

Pulse-Width Modulation (PWM) is a technique for getting analog results from a digital means. Digital control is used to create a square wave whose duty cycle or length of the pulse corresponds to a position on an analog motor controller. In our particular case, PWM is used to control the position of the analog servos, where the length of the pulse corresponds to the position of the servo, giving the control surfaces more or less influence on the airstream they encounter.

I^{2}C Communication

The Inter-integrated Circuit (I^{2}C) Protocol is a protocol intended to allow multiple slave digital integrated circuits (chips) to communicate with one or more master chips. Like the Serial Peripheral Interface (SPI), it is only intended for short distance communications within a single device. Like Asynchronous Serial Interfaces (such as RS-232 or UARTs), it only requires two signal wires to exchange information. This dual wire bus can support up to 16 devices with a 4 bit unique identifier on each device. The master sends the identifier byte out which signals the device to start receiving and transmitting.

5.3 Prototypes

Due to financial constraints and time limitations, the project is only a prototype at this time. The board constructions consists of several breakout parts listed below and a breadboard with all devices blue-wired to signals, power and ground. Onboard power is provided from the battery eliminator circuit (BEC) which provides a 5V output and is found on most hobby aircraft. The BEC is typically a feature of the electronic speed control and takes power from the motor battery, typically 6V and higher and steps it down to 5V and delivers this to the receiver and servos. This eliminates the need for a second battery of the right voltage just for the receiver/servos and in our case, the control hardware.

Breadboard and Bluewire

The breadboard and bluewire prototype is a fully functioning prototype demo of the main flight controller that is to be printed in the future. This bluewire breadboard contains all the sensors, servo and motor control inputs and outputs, and serial interfaces necessary for full scale operation.

Printed Routing Board

The printed routing board is a concept board that was initially planned to enhance testing before full scale production of the PCB begins. This board will contain all electrical routes needed between the STM32F407 and the sensor components, as well as power and ground, and servo and motor control signals.

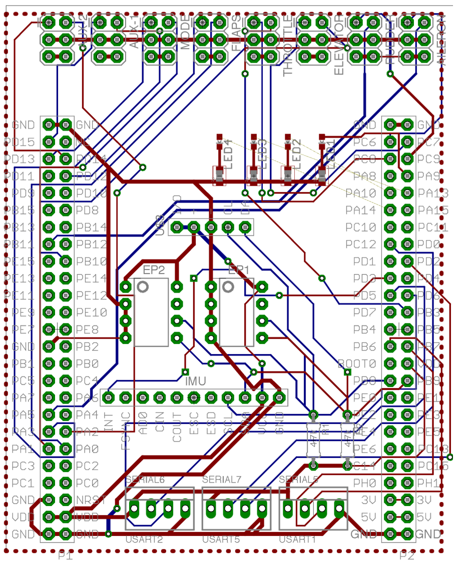

Figure 5.2: Custom Printed Routing Board Layout for Rapid Prototyping

5.4 Main Flight Board

The main flight control board will provide all functionality required for basic flight control and stabilization. It will house the STM32 processor, all sensors required for flight stabilization, and user accessible interfaces. It will connect directly to the control surface servos and actuators on board the aircraft.

LT3805 PWM Buck Converter

For powering the main board and associated hardware, we are using Linear Technologies LT3805 in 17 pin FE package shown in figure 5.3. This buck converter is a switch-mode power supply where the average Vout = Vin*Duty cycle. It can take a wide range of input voltages

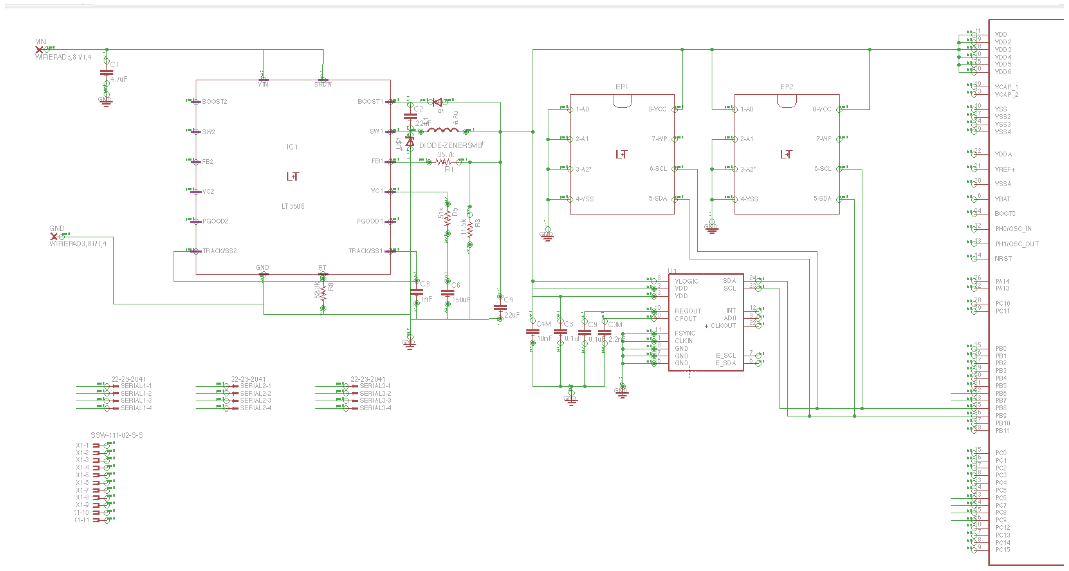

Figure 5.3: Power Supply and I^{2}C Devices in Eagle Schematic

between 3.7V and 36V and provide a range of DC outputs. Our STM32F4, EEPROM and MPU all run on 3.3V so the output voltage selected is 3.3V to power the main board. Output voltage is dictated by a resistor value to control switching frequency, and a feedback loop which controls the pulse-width-modulated duty cycle. This high frequency (current config. is 750MHz) produces a small output ripple allowing the use of small, low cost inductors and ceramic capacitors.

MPU-9150

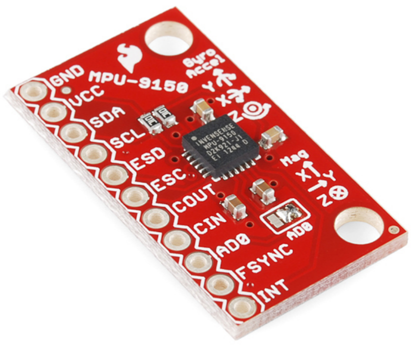

Figure 5.4: Sparkfun Breakout of the MPU-9150 9-axis Gyro and Accelerometer

This MPU, from Invensense, is an integrated 9-axis motion processor solution, providing a 3-axis accelerometer, 3-axis gyroscope and 3-axis magnetometer (compass). Taken together in a process called motion fusion, these three sets of data provide a complete picture of the aircraft's roll, pitch and yaw with respect to the ground. In addition, the magnetometer will be used for heading information. The MPU-9150 also includes a motion fusion processor which can be used to offload this processing from the main processor.

BMP 180

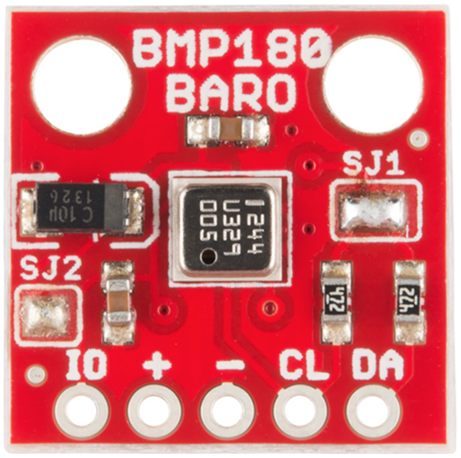

Figure 5.5: Sparkfun Breakout of the BMP 180 Pressure Sensor

The BMP 180 figure 5.5 barometer will provide short-term altitude information, to complement the GPS altitude reading when a GPS is installed, or replace it when a GPS is not installed. The barometer only measures relative altitude, however, so it will only provide altitude relative to the takeoff location unless periodically referenced against a GPS. It is most useful in an altitude-hold situation, where the flight controller attempts to minimize the change in altitude while cruising.

1 Mbit EEPROM x2

Two 1 Mbit (128 KB) EEPROM chips will provide non-volatile storage for the system. The lower approx. 1KB will be used for configuration, calibration and tuning data. This includes PID control loop settings, failsafe actions, channel mapping and MPU calibration offsets. A further 2KB will be allocated for user-defined GPS waypoint coordinates, which the aircraft will fly to in sequence when in GPS waypoint mode. The rest of the memory will be dedicated to black-box logging of all system parameters and debug variables. This will provide a tamperresistant form of logging, which can be used in the face of frivolous warranty claims or lawsuits.

Figure 5.6: 8-pin DIP Package of EEPROM Used for Datalogging

Micro SD Card Slot

A Micro SD card slot will provide user-accessible logging of GPS coordinates and system performance. This will allow the user to view and monitor system performance and efficiency, as well as debug any aircraft related issues affecting flight performance. In addition, the results of the power on self-test (POST) will be written to this card.

5.5 Current And Voltage Sensor Board

This board, a standard among higher-end flight controllers, connects in series with the battery and motor. It monitors the current voltage of the battery as well as the current drawn by the motor and all other onboard systems. This aids in improving the efficiency of flight, as well as providing an estimate of the flight time remaining to the pilot. In addition, a system can be configured to turn home before its mission is complete if the battery will not permit completion. This board will also house the 3.3 and 5 volt regulators which provide a power source to the flight control board. However, as 5 volts is also provided by the motor speed controller (ESC), this device is optional. Our implementation will be sized to measure the voltage of a 3S (11.1 volt nominal) Lithium Polymer battery pack with a maximum current draw of 80 amps, which is enough to drive the two meter gliders we are prototyping on. However, we plan to publish the pinout of this device in our documentation, which will allow this functionality to be easily implemented on much larger aircraft.

5.6 High-Power Accessory Board

This optional board, which will attach to the main flight control board over a UART, will house up to 10 MOSFETS to switch current at the voltage of the main power source, between 11-25 volts.

These accessories will be user defined and configurable, and will include high-luminance navigation lighting, a loud buzzer to locate downed aircraft in remote areas, landing gear retraction actuators and more. Each switching circuit will also be fused to prevent these non-mission-critical accessories from blowing the entire system, which is a real danger in current systems which lack fuses.

5.7 Device Compatibility

Our system will be integrated with standard RC components such as servo actuators, batteries, receivers and telemetric on-screen displays. Section 9.2.1 describes the components and protocols that already exist on the market that our system will be compatible with.

5.8 Current Protocol Compatibility

RC Receiver - 8 Channel PWM or PPM Input

PWM input consists of up to 8 individual wires leaving an RC receiver, one for each channel of control. On each wire, a high pulse of between 1 and 2 ms indicates the analog value of the input, with 1ms being -100 percent, 1.5ms being 0 percent and 2ms being 100 percent. Traditionally, this signal is sent directly to servos to indicate the angle they should rotate to, and to speed controllers to indicate the throttle value. A PPM input is the same signal on one wire, with a synchronization pulse followed by one PWM pulse per channel. While all RC receivers support PWM and only some support PPM, PPM has recently become highly popular due to its two wire interface (1 line + ground) versus the 9 or more lines required for PWM input.

RC Servo - PWM Output

These actuators rotate to a specific angle depending on the length of the PWM pulse inputted, and are used to move control surfaces such as the elevator and rudder. In addition, the same protocol is used to control the motor speed.

MAVLink Telemetry

An industry standard protocol, MAVLink, will be used to transmit telemetry from a USART port. This will ensure compatibility with a wide range of ground station accessories, such as telemetric antenna trackers, glass cockpits and voice annunciators.

Chapter 6

Flight Modes

This section describes the various flight modes that the system can be in at any time. These flight modes can be changed in-flight using a switch on the RC transmitter.

6.1 Mode 0 - Unarmed

Entered after power-on self-test (POST) Throttle pinned to zero (motor will not spin). Other outputs follow user-selected flight mode.

6.2 Mode 1 - Manual Control

In this mode, RC control is passed through the Pilot-1 system with no change. To the user, it appears that the Pilot-1 system has disconnected, and the RC receiver is connected directly to the servos and propeller. This is the preferred control mode whenever abrupt movements and hands on control are required, such as takeoffs and landings.

6.3 Mode 2 - Angle

This mode uses the IMU to keep the aircraft level in the pitch and roll axis. A control input on the RC control sticks equates to a specific angle, and the aircraft will hold that angle until the stick returns to zero. This is the preferred mode once the craft is in the air, allowing the pilot to perform other tasks while the aircraft levels itself.

6.4 Mode 3 - Failsafe

Sensors required: accelerometer, gyro, GPS or pitot airspeed, current sensor roll, pitch same as mode 2 throttle input = climb/fall rate Throttle managed to keep airspeed at most efficient level above stall speed.

Chapter 7

Use Cases

7.1 Cases

A use case defines the steps a user must take to accomplish a specific task. The following use case describes how a pilot and observer may fly to a specific location, with the pilot controlling the aircraft while the observer uses the observer camera to scan the area. This is a common use case in recreational, search-and-rescue, surveying, firefighting and other fields.

7.2 Use Case - Flying to a Specific Location

Goal:

Take off, fly to GPS coordinate. Manually explore surrounding area, and return home.

Actors:

Pilot, Observer

Preconditions:

Pilot-1 is installed in aircraft. Pilot-1 is properly configured and tuned. A GPS is installed on the system. The specified GPS coordinate has been transmitted to the system via The configuration program. Pilot has basic RC piloting skills.

Postconditions:

System is circling above its launch location.

Pilot steps:

Power on aircraft, control transmitter, video receiver. Wait for GPS lock. Switch to preferred takeoff mode - manual mode or auto-level mode Increase throttle command, pull back on elevator to lift off. Adjust aileron, elevator and throttle as required to keep aircraft on desired departure path. Switch flight mode to GPS waypoint mode. Wait for aircraft to attain GPS waypoint. Switch flight mode to auto-level mode. Use aileron and elevator input to steer aircraft. Wait until observer is satisfied or battery level necessitates a return to launch. Switch flight mode to GPS return-to-home mode. Wait until aircraft is loitering above launch area. Switch flight mode to preferred landing mode - manual or auto-level. Land aircraft as a conventional RC aircraft.

Observer steps:

Wait until pilot has completed step 7. Use auxiliary input stick, if available, to pan and tilt observer camera.

Exceptions:

Insufficient battery power to complete flight. GPS target is beyond range of control or video links. Intended target is behind physical features such as buildings or a mountain.

Chapter 8

Testing

8.1 Benchtop Tests

Once the hardware breadboarding had been complete with the Discovery development kit and the MPU and Barometer breakout boards, certain input and output modules were coded. Once these input/output modules were assumed to be functioning properly, development on the processing code, such as auto-stabilization and attitude control began. As this code was developed, the hardware was connected to generate inputs and hardware and software interfaces were created to view the outputs (servos or debug code on a laptop). The hardware was manually moved to elicit a change in the MPU, and the outputs were be observed to ensure appropriate device response. A rudimentary test rig was built using breadboarding of components, which lay out servos according to their physical position on the plane (ailerons, rudder and elevator, along with a servo to indicate the value of the throttle output). A pointer was then attached to the output shaft of each servo to visually indicate the deflection from neutral.

8.2 Hardware Test

To ensure code stability, the hardware was first be tested to ensure it could will successfully run the flight code. To do this, unit software tests were be run to ensure that all onboard inputs and outputs, such as the MPU, EEPROM and UART accessory links, could be read to and written from. Long term tests several times the length of an expected flight have been run overnight to ensure that the hardware remains stable after prolonged runs.

8.3 Manual Mode Verification

Before prototyped hardware was installed on an airframe, the operation of the manual, or passthrough, mode had to be verified. This mode pipes the RC inputs directly to their respective outputs, essentially removing the system from the control of the aircraft. This mode is entered or exited using a switch on the RC receiver. Bench tests were used to verify that in the case of malfunction of more advanced flight modes, the manual mode can always be entered into by the pilot or a failsafe routine could be entered into in case of a total communication loss or hardware disconnect.

8.4 Future Testing

Airframe Operation Verification

After development has progressed to the point where all team members believe that flight testing may proceed, the system will be installed in a test airframe. This small, 1 meter wingspan powered glider is made out of EPO foam, which is easily repaired following a crash. This aircraft will be equipped with the least expensive RC batteries and components possible, to minimize the financial loss in case of a catastrophic failure. The aircraft will be launched in manual mode, and taken to a “minimum recovery altitude”, being the minimum altitude required to recover from a roll-over or stall. At this point, the flight mode undergoing test will be activated by the pilot, who will subjectively assess the “feel” of the flight mode. A second team member will also objectively assess system performance, either in real-time via telemetry or post-flight via data logged onboard. If at any time the aircraft becomes unresponsive or enters an unacceptable orientation, the pilot will switch back to manual mode and land the plane. A landing will only be attempted in the flight mode under test if the pilot deems it safe. Optionally and at the test pilots discretion, the aircraft will be equipped with a parachute system entirely separate from the Pilot-1 system which can, in the case of system failure, halt the forward motion of the aircraft and bring it to the ground with minimal damage.

Mid-Range Video Flight Tests

After several successful test flights, the aircraft will be equipped with video flight and long-range radio control equipment. A flight will be executed along the standard Baylands testing flight path. This flight path has been flown by the test pilot for several years, and thusly provides familiar landmarks. This test flight will be a maximum of 1 KM over land, during which time the aircraft does not cross over any populated areas. The aircraft will be flown via video downlink, rather than line of sight. Multiple laps to the 1 KM marker and back to Baylands will be performed, until battery exhaustion, to simulate a much longer flight. The vehicle used in these tests will be equipped with a parachute abort system controlled by a separate radio system.

Long Range Flight Tests

These tests will only be executed once the system is complete and GPS failsafe return-to-home has been proven to work reliably. This test will be flown in the larger Super SkySurfer aircraft, equipped with high capacity batteries which provide two hours of flight time. The test will be flown along the Baylands southern long-range flight path. This flight path heads downwind to the east, until the aircraft crosses over the town of Alviso. At this point, the pilot will visually acquire the Amtrak train tracks running through Alviso, and follow them north to the ghost town of Drawbridge. As the aircraft crosses over Drawbridge, it will be 7 KM away from Baylands. Depending on control signal conditions, the pilot may turn back over the bay to return to Baylands, or continue north along the train tracks to increase the flight distance. When the pilot turns towards Baylands, he may optionally engage GPS return to home to evaluate its performance. As this segment is over-water, it must be closely monitored to prevent against a costly systems loss. This test will be primarily conducted to assess the ability of the system to maximize power efficiency and thusly, the test may be flown simultaneously with an identical aircraft with known efficiency characteristics as a control sample.

Distributed Beta Tests

We would like to thank the community at Sunnyvale Baylands for their continued support and assistance during the development of this product. To repay them as well as receive feedback from experienced flyers, retail kits of Pilot1, which include the Pilot-1 main board, accessory control board, GPS and power module will be offered to the community below the cost of production. Their feedback will be used to improve the usability of the system, as well as integrate oft-requested features. In addition, this test will ensure a main performance goal of the system by ensuring that it is compatible with a wide variety of aircraft and control setups.

Exhibition Test

Upon completion of the development and test plan, the system will be flown for the maximum possible flight time at a scenic location, such as along the California coast. The test airframe will be a new, never-crashed Super SkySurfer, which will be outfitted with pleasing visuals. A second camera ship will tail the test aircraft. Video from onboard cameras, ground-based cameras and the tail aircraft will be integrated to produce a demonstration video of all systems capabilities and features, for the consideration of the Senior Design judges and potential customers.

Chapter 9

Societal Issues

Our project is on the forefront of technological advancement, but it is not without its issues. The very mention of drones in our society brings up controversy and questions of legality. For this reason, it is important to outline the issues concerning the ethical implications of our project. This analysis would delve into the social, team/organizational, and design issues that are relevant to this project. The drone project has a variety of stakeholders that can be affected by our project. These are, but are not limited to:

Ethical

Our Pilot-1 project is designed with safety in mind, and in designing our project, we made sure to address the ethical sides of such an endeavor. In order to maintain safe, autonomous flight, we had to thoroughly test our flight control functionality until it was at a reliable state, with no system-breaking bugs present at launch. To do this, we maintain the highest standard of coding practices to ensure our system is air-tight. We also used the highest quality of electrical components to ensure the reliability of the final product.

Social

Our system is meant to be used by the general public, the consumer, so a major component of our project is the ability to safely handle failure states. To do that, we've implemented a failure handling system that requires the user to take back control, or in case of no user control, an optional parachute attachment may be deployed, landing the aircraft safely. Another concern was the misuse of our project, which we have addressed by limiting the systems capabilities to those only of flight and control of flight. Our project cannot be re-purposed for any other use other than recreational RC plane flying.

Political

Due to the hot-button issue of drones in today's media, there is a certain amount of speculation as to whether drones being in public hands would be acceptable. We understand this concern, and thus have designed our project to work well within the confines of the RC industry. What our group offers is not full-on drone technology, but simply a method of controlling their planes with more precision. We note that drone technology in public and private settings may be alarming, and have thus allowed the project to only exists within a hobbyist scope.

Economic

Our goal with this project was to simply give the RC industry a competitive flight controller, aimed to please the RC hobbyist of all budgets. Due to the costs of custom printed circuit boards, and the time to make them, our project was done by hand using readily available material at local electronics shops. However, this can be produced en masse in a manufacturing facility as any other electric appliance or device. Of course, the pricing of the final product will be based upon the demand, the manufacturing process of the final product, and other things. Our main goal is to make it affordable for the average consumer.

Health and Safety

As far as health and safety concerns for this product goes, our design is aimed to be a small, sturdy, and compact device that is easily fitted into the cavity of an RC plane. The device would be safe to the touch and can be handled without much difficulty. Since the product is an electronic device, we do ask to handle the device with care and keep it away from contaminants and other destructive materials, such as liquids.

Manufacturability

Our project is made with the idea of mass production. The issues we faced in attempting to produce the prototype were the cost of the materials needed for the printed circuit boards and the time-intensive schedule it takes to manufacture them. If this product were to go into production, a manufacturing facility would be needed as well as software/hardware QA testers, the costs of which would need to be covered by investors in the company.

Sustainability

Our software would be updated via simple online releases of new code to be flashed onto the on-board memory. As for the hardware, the sturdy design and the available components make it suitable for flight aboard an RC plane.

Environmental Impact

We do not view this project as having a significant environmental impact, if at all.

Usability

Our product is manufactured with usability in mind. A simple package with instruction would get the user started in installing and using the product.

Lifelong Learning

This project forced us to get acquainted with tools and equipment the group members may not have used before. In an effort to get the project working, we had to find materials to tools to weld together the few components, and use tools that are different than those we have been using. The project truly inspired us to come out of our comfort zones and tackle new challenges head on.

Compassion

Our project does not, in and of itself, alleviate suffering or help those in need. It is a recreational product that results from the use of RC-related products that were deemed lacking or unsatisfactory, which led to the creation of our own fixed-wing RC plane controlling device. While our project isn't an altruistic one, it certainly is made with the knowledge of satisfying other RC hobbyists' wants and needs when it comes to a capable product for their beloved RC planes.

Chapter 10

Outcome and Future Expansion

Our project was hindered by conflicting documentation on the chosen STM32 processor, as well as staffing issues. A team member, representing one quarter of our effective workforce, left the team one third of the way through development, and a second team member was struck with a long-term illness. As such, we dropped the requirements for GPS-based operation and instead focused only on flight modes that would operate using the IMU. To compensate for this reduced functionality, we implemented a plugin system, where a third party may add GPS operation, or indeed any other functionality, to our code at a later date. In addition, development was hindered by issues with the STM32Cube system. As this system was just released a few months before development began, third party support was nonexistent. To compensate for this, we chose to rely much less heavily on the STM32Cube layer, and do the majority of our GPIO through the STM32F4-Discovery third party libraries.

Testing was hindered and delayed due to problems integrating the various software and hardware components together. Per our design plan, each software module was coded and tested independently before integration into the final project. Problems arose, however, due to unforeseen interactions and competition for device resources. For example, the IMU and RC control input modules were tested separately as fully operational. However, when attempting to receive IMU data and RC input at the same time, it was discovered that the task of reading the IMU would block the processor for long enough that it missed one or two RC pulses, which would then be flagged as a failure. The hardware presented a similar challenge – a breadboarded version of the system was fully operational, however due to its construction it was impossible to mount in a flying aircraft. The flight worthy board was built, however hardware failures arose in its construction and the decision was made to limit testing to benchtop testing with the breadboard unit.

We believe we have provided a solid base for future development to continue on this platform. As we have abstracted away the underlying tasks present in running such a system, further development may continue without concern for low-level tasks such as reading the IMU or detecting RC failures.

Appendix A

Source Code

A.1 Main Source

```
/*the tm_stm32f4_xxx libraries and files are from http://stm32f4-discovery.com, used
 * See any tm_stm32f4_xxx.h file for further details.
 */
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
#include "tm_stm32f4_servo.h"
#include "tm_stm32f4_delay.h"
#include "tm_stm32f4_usart.h"
#include "tm_stm32f4_pwmin.h"
#include "tm_stm32f4_i2c.h"
#include "tm_stm32f4_disco.h"
#include "settings.h"
#include "structs.h"
#include "flight_modes.h"

#define USE_USART1 1
#define IMU_ADDRESS 0xd0

#define IMU_ROLL_MIN -16000//the IMU readings at -90 and 90 degrees from level...
#define IMU_ROLL_MAX 16000 //for purposes of translating to angles in degrees
#define IMU_PITCH_MIN -16000
#define IMU_PITCH_MAX 16000

void startup();
void debug(char*);


float mapFloat(long x, long in_min, long in_max, long out_min, long out_max);


int main(void)
{

        /* Initialize system */
        SystemInit();
        TM_USART_Init(USART1, TM_USART_PinsPack_2, 9600); //PB6, RX: PB7
        TM_DELAY_Init();
        TM_DISCO_LedInit();
        debug("Pilot-1 V0.1 starting up\n\r");
        startup();


        TM_I2C_Init(I2C1, TM_I2C_PinsPack_2, 100000);

        //Write "5" at location 0x00 to slave with address ADDRESS
        TM_I2C_Write(I2C1, IMU_ADDRESS, 0x6B, 0x01);


        char buffer[255];
        while(1){
                if (TM_DELAY_Time()>50) {
                        TM_DELAY_SetTime(0);
                        readIMU();
                        processIMU();
                        readRCIn();
                        if(rc_raw.mode==0){
                                MODE0;
                        }else if(rc_raw.mode==1){
                                MODE1;
                        }else if(rc_raw.mode==2){
                                MODE2;
                        }else if(rc_raw.mode==3){
                                MODE3;
                        }else{
                                mode_failsafe();
                        }
                        if(rc_raw.failsafe>RC_FAILSAFE_THRESHOLD || imu_proc.failsafe
                                mode_failsafe();
                        }else{
                                clear_failsafe();
                        }
                }
        }
}


void startup(){ //called after the system has been initialized, to begin Servo outpu
        debug("Init servos...");
        TM_SERVO_Init(&ServoRoll, TIM2, TM_PWM_Channel_1, TM_PWM_PinsPack_2); // = P
        TM_SERVO_Init(&ServoPitch, TIM2, TM_PWM_Channel_2, TM_PWM_PinsPack_2); // = 
        TM_SERVO_Init(&ServoThrottle, TIM2, TM_PWM_Channel_3, TM_PWM_PinsPack_2); //
        TM_SERVO_Init(&ServoYaw, TIM2, TM_PWM_Channel_4, TM_PWM_PinsPack_2); //= PB1
        debug("[ OK ]\r\n");
        TM_SERVO_SetMicros(&ServoThrottle, 1000); //pin the throttle to zero

        TM_PWMIN_InitTimer(TIM3, &PWMIN_Roll, TM_PWMIN_Channel_1, TM_PWMIN_PinsPack_
        TM_PWMIN_InitTimer(TIM3, &PWMIN_Pitch, TM_PWMIN_Channel_2, TM_PWMIN_PinsPack
        TM_PWMIN_InitTimer(TIM5, &PWMIN_Yaw, TM_PWMIN_Channel_1, TM_PWMIN_PinsPack_1
        TM_PWMIN_InitTimer(TIM5, &PWMIN_Mode, TM_PWMIN_Channel_2, TM_PWMIN_PinsPack_

}

void readIMU(){
        int8_t data[6];
        TM_I2C_ReadMulti(I2C1, IMU_ADDRESS, 0x3B, data, 6);

        imu_raw.acc[0]=(data[0]<<8) | (data[1]&0xFF);
        imu_raw.acc[1]=(data[2]<<8) | (data[3]&0xFF);
        imu_raw.acc[2]=(data[4]<<8) | (data[5]&0xFF);

        TM_I2C_ReadMulti(I2C1,IMU_ADDRESS, 0x43, data, 6);
        imu_raw.gyro[0]=(data[0]<<8) | (data[1]&0xFF);
        imu_raw.gyro[1]=(data[2]<<8) | (data[3]&0xFF);
        imu_raw.gyro[2]=(data[4]<<8) | (data[5]&0xFF);

        TM_I2C_ReadMulti(I2C1, IMU_ADDRESS, 0x41, data, 2);
        imu_raw.temp = ((data[0]<<8) | (data[1]&0xFF));

}

void processIMU(){
        imu_proc.temp = imu_raw.temp/340 + 35;

        if(imu_proc.temp<10 || imu_proc.temp>200){
                //indicates either an I2C bus failure or a fire in the cockpit. Eith
                imu_proc.failsafe++;
        }

        float r = (double)(imu_raw.gyro[0] * IMU_GYRO_WEIGHT) + (double)((1-IMU_GYRO_
//x gyro, y acc
        imu_proc.roll = mapFloat(r, IMU_ROLL_MIN, IMU_ROLL_MAX, -90, 90);

        float p = (double)(imu_raw.gyro[1] * IMU_GYRO_WEIGHT) + (double)((1-IMU_GYRO_
//y gyro, x acc
        imu_proc.pitch = mapFloat(p, IMU_PITCH_MIN, IMU_PITCH_MAX, -90, 90);


}

void readRCIn(){

        /* Get new data for both input signals */
        TM_PWMIN_Get(&PWMIN_Roll);
        TM_PWMIN_Get(&PWMIN_Pitch);
        TM_PWMIN_Get(&PWMIN_Yaw);
        TM_PWMIN_Get(&PWMIN_Throttle);
        TM_PWMIN_Get(&PWMIN_Mode);

        /* Valid PWM input signal */
        uint8_t fail=0;
        if (PWMIN_Roll.Frequency > 0) {
                rc_raw.roll = (PWMIN_Roll.DutyCycle/100.0)*(1000000.0/PWMIN_Roll.Freq
        }else{
                //fail=1;
        }
        if(PWMIN_Pitch.Frequency > 0){
                rc_raw.pitch = (PWMIN_Pitch.DutyCycle/100.0)*(1000000.0/PWMIN_Pitch.
        }else{
                fail=1;
        }
        if(PWMIN_Yaw.Frequency > 0){
                        rc_raw.yaw = (PWMIN_Yaw.DutyCycle/100.0)*(1000000.0/PWMIN_Ya
                }else{
                        fail=1;
                }
        if(PWMIN_Throttle.Frequency > 0){
                rc_raw.throttle = (PWMIN_Throttle.DutyCycle/100.0)*(1000000.0/PWMIN
        }else{
                fail=1;
        }
        if(PWMIN_Mode.Frequency > 0){
                        int mode = (PWMIN_Mode.DutyCycle/100.0)*(1000000.0/PWMIN_Mod
                        if(mode<1000){
                                rc_raw.mode = 0;
                        }else if(mode<1250){
                                rc_raw.mode = 1;
                        }else if(mode<1750){
                                rc_raw.mode = 2;
                        }else{
                                rc_raw.mode = 3;
                        }
                }else{
                        fail=1;
                }

        fail = (fail || rc_raw.roll>2500 || rc_raw.pitch>2500);
        rc_raw.failsafe=(fail ? rc_raw.failsafe+fail : 0);


}

/* TIM3 IRQ handler */
void TIM3_IRQHandler(void) {
        /* Interrupt request, don't forget! */
        TM_PWMIN_InterruptHandler(&PWMIN_Roll);
        TM_PWMIN_InterruptHandler(&PWMIN_Pitch);

}
void TIM5_IRQHandler(void){
        TM_PWMIN_InterruptHandler(&PWMIN_Mode);
        TM_PWMIN_InterruptHandler(&PWMIN_Throttle);
}

void debug(char* str){
        //#ifdef DEBUG
        TM_USART_Puts(USART1, str);
        //#endif
}

float mapFloat(long x, long in_min, long in_max, long out_min, long out_max)
{
        return (float)(x - in_min) * (out_max - out_min) / (float)(in_max - in_min) +
}
int map(int x, int in_min, int in_max, int out_min, int out_max)
{
        return (int)(x - in_min) * (out_max - out_min) / (int)(in_max - in_min) + ou
}

// ----------------------------------------------------------------------------
```

A.2 Flight Modes Source

```
/*
 * flight_modes.c
 *
 *  Created on: Jun 1, 2015
 *      Author: millsapski
 */

#include "tm_stm32f4_disco.h"

#include "structs.h"
#include "settings.h"

#include "flight_modes.h"


/*
 * Additional flight modes may be added here. To add a flight mode, use data in rc_ra
 * Use TM_SERVO_SetMicroseconds and TM_SERVO_SetDegrees as output to flight surfaces
 * Lastly, register the flight mode as your desired mode number in settings.h
 */

void mode_unarmed(){
        TM_SERVO_SetMicros(&ServoThrottle, 1000); //pin the throttle to zero

}

void mode_manual(){

        TM_SERVO_SetMicros(&ServoRoll, rc_raw.roll);

        TM_SERVO_SetMicros(&ServoPitch, rc_raw.pitch);

        TM_SERVO_SetMicros(&ServoThrottle, rc_raw.throttle);

        TM_SERVO_SetMicros(&ServoYaw, rc_raw.yaw);
}
```

```
void mode_level(){
        //uses a simple proportional filter, which is all thats really necessary for
        if(isValidMicro(rc_raw.roll))
                TM_SERVO_SetDegrees(&ServoRoll, 90-((int)(imu_proc.roll*P_ROLL))+map
        if(isValidMicro(rc_raw.pitch))
                TM_SERVO_SetDegrees(&ServoPitch, 90-((int)(imu_proc.pitch*P_PITCH))+
        if(isValidMicro(rc_raw.throttle))
                TM_SERVO_SetMicros(&ServoThrottle, rc_raw.throttle); //passthrough t
        if(isValidMicro(rc_raw.yaw))
                TM_SERVO_SetMicros(&ServoYaw, rc_raw.yaw);
}
void mode_failsafe(){
        TM_SERVO_SetMicros(&ServoThrottle, 1000); //shut down throttle
        if(PARACHUTE_FAILSAFE_ENABLED){
                TM_DISCO_LedOn(LED_RED); //parachute connected to D13 (red LED for v
        }else{
                rc_raw.roll = CROSSCONTROL_GLIDE_VAL;
                rc_raw.yaw = -CROSSCONTROL_GLIDE_VAL;
                mode_level();
        }
        TM_DISCO_LedOn(LED_GREEN | LED_ORANGE | LED_BLUE);
}

void clear_failsafe(){
        TM_DISCO_LedOff(LED_RED | LED_GREEN | LED_ORANGE | LED_BLUE);
}

int isValidMicro(long time){
        return (time>800 && time<2200);
}
```

A.3 Header Files

```
/*
 * settings.h
 *
 *  Created on: Jun 1, 2015
 *      Author: millsapski
 */

#ifndef SETTINGS_H_
#define SETTINGS_H_

//section: RC controller configuration
//EPA : end-point configuration. The minimum and maximum values that the RC controlle
#define RC_EPA_ROLL_LOW 1000
#define RC_EPA_ROLL_HIGH 2000
#define RC_EPA_PITCH_LOW 1000
#define RC_EPA_PITCH_HIGH 2000

//section: flight tuning. These are aircraft-specific values, and must be tuned to a
#define IMU_GYRO_WEIGHT 0.3
```

```
#define P_ROLL 0.9//proportional response for changes to attitude. Higher = stronger
#define P_PITCH 0.9


//section: failsafe configuration
#define PARACHUTE_FAILSAFE_ENABLED 1 //set to 1 to enable parachute failsafe, 0 to d
#define RC_FAILSAFE_THRESHOLD 10 //the number of failures on the RC input before fail
#define IMU_FAILSAFE_THRESHOLD 3 //the number of I2C bus errors before failsafe. Due
#define CROSSCONTROL_GLIDE_VAL 10 //when failsafe is deployed and parachute is not a

//section: flight mode configuration
// use MODEn, where n is 0-3, and
#define MODE0 mode_unarmed()
#define MODE1 mode_manual()
#define MODE2 mode_level()
#define MODE3 mode_failsafe() //always set mode3 to failsafe()


#endif /* SETTINGS_H_ */



/*
 * structs.h
 *
 *  Created on: Jun 1, 2015
 *      Author: millsapski
 */
/*
 * Public structures available for creators of new flight modes. Updated every 50ms
 */
#ifndef STRUCTS_H_
#define STRUCTS_H_

#include "tm_stm32f4_pwmin.h"
#include "tm_stm32f4_servo.h"

TM_SERVO_t ServoRoll, ServoPitch, ServoYaw, ServoThrottle;
TM_PWMIN_t PWMIN_Roll, PWMIN_Pitch, PWMIN_Yaw, PWMIN_Throttle, PWMIN_Mode;

struct imu_raw_data{
        int16_t acc[3]; //x, y, z
        int16_t gyro[3]; //x,y,z
        int16_t compass[3]; //x,y,z
        int16_t temp;
};
struct imu_processed_data{
        float roll, pitch, yaw;
        int16_t temp;
        uint8_t failsafe;
};
struct rc_raw_data{
        float roll, pitch, yaw, throttle, mode;
        uint8_t failsafe;
};

struct imu_raw_data imu_raw;
struct imu_processed_data imu_proc;
struct rc_raw_data rc_raw;


#endif /* STRUCTS_H_ */



/*
 * flight_modes.h
 *
 *  Created on: Jun 1, 2015
 *      Author: millsapski
 */

#ifndef FLIGHT_MODES_H_
#define FLIGHT_MODES_H_

void mode_unarmed();
void mode_manual();
void mode_level();
void mode_failsafe();
void clear_failsafe();

#endif /* FLIGHT_MODES_H_ */
```