



UNIVERSITY OF IDAHO

CS CAPSTONE DESIGN

Capstone Portfolio

Drone Mission Planning Software

Team:
Mission Control

Authors:
Joseph HIGLEY
David KLINGENBERG
Emeth THOMPSON
Taylor TRABUN

Advisors:
Dr. Bruce BOLDEN
Dr. Robert RINKER

Customer:
Brandon ORTIZ

May 6, 2015

Contents

1	Team Member Contact Information	1
2	Introduction	1
2.1	Target Priorities	1
3	Initial Client Interview Transcript 9/10/14	2
3.1	Meetings	2
3.2	End Goal	2
3.3	First Steps	2
3.4	Requirements	2
3.5	Other Notes	2
4	Meeting Agendas	3
4.1	Sept. 10, 2014	3
4.1.1	short	3
4.2	Sept. 18, 2014	3
4.2.1	short	4
4.3	Sept. 25, 2014	4
4.3.1	short	5
4.4	Oct. 2, 2014	6
4.4.1	short	6
4.5	Oct. 9, 2014	7
4.5.1	short	7
4.6	Oct. 16, 2014	8
4.6.1	short	9
4.7	Nov. 6, 2014	10
4.7.1	short	10
4.8	Nov. 20, 2014	12
4.8.1	short	12
4.9	Jan. 22, 2015	14
4.9.1	short	14
4.10	Jan. 27, 2015	16
4.10.1	short	16
4.11	Feb. 3, 2015	17
4.11.1	short	17
4.12	Feb. 17, 2015	19
4.12.1	short	19
4.13	March 5, 2015	20
4.13.1	short	20

5	Code	21
5.1	Supplemental quad copter autopilot V1.8	21
5.2	Home Arduino Firmware	27
5.3	xAPI Services	31
5.3.1	Do Move Service	31
5.3.2	Heading Service	34
5.3.3	Heading Hold Service	38
5.3.4	Heartbeat Service	42
5.3.5	Land Service	47
5.3.6	Takeoff Service	50
5.3.7	Serial Service (Modified from Brandon's Version)	53
6	Design Presentation	64
6.1	Nov 13, 2014	64
7	Design Document	74
7.1	Introduction	74
7.2	Drone Design	74
7.3	Communication Design	75
7.3.1	Communication Overview	75
7.3.2	Hardware Components	75
7.3.3	XAPI	76
7.4	Graphical User Interface Design	77
	Appendices	79
A	Miscellaneous UML Charts	79
B	ATMEL[©] Microcontrollers	80
C	Technical Drawings	82

List of Figures

1	3D sketch of partbin	74
2	Overview of communications system	76
3	Graphical user interface mock-up design	78
4	Communication Sequence	79
5	PID Controller	79
6	ATmega644	80
7	ATmega2560	81
8	Battery Bracket	82
9	Electronic Speed Controller Bracket	83

10	Landing Strut	84
11	Wire Brace Upper Clamp	85
12	Motor Mount Wire Brace	86
13	Gps Mount	87
14	Battery Box Spacers	88
15	10 Deg. of Freedom Sensor Platform	89
16	Wire Brace	90
17	Arduino Platform	91
18	LED Bracket	92

List of Tables

1	Team Member Contact Information	1
2	Priorities	1

1 Team Member Contact Information

Name	Phone Number	Email Address
Joseph Higley	(208) 310-9657	higley@vandals.uidaho.edu
David Klingenberg		bigwookiee@Gmail.com
Emeth Thompson		thom5468@vandals.uidaho.edu
Taylor Trabun	(509) 995-0904	trab1744@vandals.uidaho.edu

Table 1: Team Member Contact Information

2 Introduction

Software to create and upload a flight plan to a quad copter drone. The flight plan will be uploaded using xBee radio communication.

This project will use off-the-shelf parts. ATMEL[©] based microcontrollers found on arduino based open source boards is the current preference.

2.1 Target Priorities

Number	Category	Need	Importance
1	Quadcopter	Center of Gravity Refined	5
2	Quadcopter	Reliable Flight	5
3	Quadcopter	Functioning xBee Hardware	4
4	Quadcopter	Hardware (Microcontroller) with xAPI and services to control flight	5
5	Quadcopter	Controlled with XP communications	4
6	Quadcopter	Autoland	5
7	Software	software package for flight planning	2
8	Software	API for sending commands from computer	2

Table 2: Priorities

3 Initial Client Interview Transcript 9/10/14

Mentor/Client: Brandon Ortiz

3.1 Meetings

We will be having weekly meetings in Brandon's office on Thursdays at 3:30 PM. These meeting will include status updates, further work on designs, troubleshooting, and assignment of tasks

3.2 End Goal

To have a stable and flying quadcopter that can be communicated with remotely. In addition, work done on a flight planning software (including GUI) should be underway. The project will be done in small steps, as this project requires research and development throughout.

3.3 First Steps

- Learn how quadcopter works
- Reconstruct quadcopter to be stable
- Learn how to fly quadcopter
- Understand flight computer documentation
- Design communications
- Be sure to use xAPI

3.4 Requirements

- Functional quadcopter (stable)
- Documentation of quadcopter construction
- Use of xAPI on arduino communication system
- Communication system using xBEE to communicate from computer to quadcopter
- Ability to send commands to quadcopter
- Flight planning software, including GUI

3.5 Other Notes

Other notes from the meeting included aviation terminology, how to pair the remote control and quadcopter receiver, quick tour of controller and motor adjustments, and a quick tour of flight computer.

4 Meeting Agendas

4.1 Sept. 10, 2014

Mission Control Team Agenda

Friday September 10, 2014.
1500 — 1600 in JEB Think Tank.

Type of Meeting

Initial client interview.

Attendees

David Klingenberg

Taylor Trabun

Brandon Ortiz

Topics

Topic	Responsible	Time (in minutes)
Product Overview	Brandon	15
System Requirements	Brandon	15
Tasks Breakdown	Open Discussion	15
Question & Answers	Open Discussion	25

Additional Information: This is our initial client interview.

4.1.1 Minutes from Friday September 10 Meeting

Refer to [Section 3](#) initial client transcript.

4.2 Sept. 18, 2014

Mission Control Team Agenda

Thursday September 18, 2014.
1500 — 1600 in JEB Think Tank.

Type of Meeting

Initial Planning

Attendees

David Klingenberg

Taylor Trabun

Brandon Ortiz

Bruce Bolden

Topics

Topic	Responsible	Time (in minutes)
Progress Report	David, Taylor	5
System Overview	Brandon	10
Tasks Breakdown	Open Discussion	20
Additional Words of Wisdom	Bruce	5
Question & Answers	Open Discussion	20

Additional Information:

The rerouting and reconfiguring of the drone is proceeding nicely. It progress will be shown at the meeting time.

4.2.1 Minutes from Thursday September 18 Meeting

- 1505 Meeting Started
- Discussed drone rebuild progress.
- Evaluated ESC bin for the drone.
 - Refer to [figur 9](#) in Appendix [C](#)
- Discussed, evaluated, and illustrated the communication sequence.
 - Refer to [figur 4](#) in Appendix [A](#)
- 1610 Meeting

4.3 Sept. 25, 2014

Mission Control Team Agenda

**Thrusday September 25, 2014.
1530 — 1630 in JEB 37**

Type of Meeting

Status Report and Next Week Planning

Attendees

David Klingenberg

Taylor Trabun

Brandon Ortiz

Topics

Topic	Responsible	Time (in minutes)
Progress Report	David & Taylor	10
Demonstrations	David & Taylor	10
New Tasks	Open Discussion	20
Question & Answers	Open Discussion	20

Additional Information:**4.3.1 Minutes from Thursday September 25 Meeting**

- 1530 Meeting Start
- Discussed LCD use on Arduinos.
- Reviewed TUN packets.
- Status updates
 - Things moving along.
 - Getting closer to flying possibly next Thursday.
- xBee discussion on how to connect.
- Evaluated future problems.
 - Gyros and accelerometers need to be implemented separately from the flight computer.
- 1630 Meeting Ended

4.4 Oct. 2, 2014

Mission Control Team Agenda

**Thursday October 2, 2014.
1530 — 1630 in JEB 37**

Type of Meeting

Status Report and Next Week Planning

Attendees

David Klingenberg

Taylor Trabun

Brandon Ortiz

Topics

Topic	Responsible	Time (in minutes)
Progress Report	David & Taylor	10
Demonstrations	David & Taylor	10
New Tasks	Open Discussion	20
Question & Answers	Open Discussion	20

Additional Information:

4.4.1 Minutes from Thursday October 2 Meeting

- 1530 Meeting Start
- Status updates.
 - Taylor has one-way communications working.
 - David finished a prototype for the ECS bin.
 - * Bin needs its weight reduced.
 - * ECS cables need to be lengthened.
- To
 - Taylor will attempt to get XP comm working.
 - David will finish quadcopter.
 - Get a new adrenal for running a second xBee radio.
 - Solder new LCD board.

- xBee Configuration notes.
 - Use XCTU tool for configuration.
 - Need FID drivers installed for XCTU tool.
- 1630 Meeting Ended

4.5 Oct. 9, 2014

Mission Control Team Agenda

**Thursday October 9, 2014.
1530 — 1630 in JEB 37**

Type of Meeting

Status Report and Next Week Planning

Attendees

David Klingenberg
Taylor Trabun
Brandon Ortiz

Topics

Topic	Responsible	Time (in minutes)
Progress Report	David & Taylor	10
Demonstrations	David & Taylor	10
New Tasks	Open Discussion	20
Question & Answers	Open Discussion	20

Additional Information:

4.5.1 Minutes from Thursday October 9 Meeting

- 1530 Meeting Start
- Update
 - Taylor is preparing for snapshot day.
 - David
 - * Quadcopter rebuilt.
 - * Simple xBee terminals working between two computers.

- New Resources
 - UAV control paper with GUI design example.
 - Survey of UAV papers.
- Action Items
 - David will experiment with PWM and the quadcopter and portfolio.
 - Taylor will work on poster for snapshot day and continue working on communications.
- Test Flight
 - Quadcopter has severe drift forward. David will work on solution.
- 1630 Meeting Ended

4.6 Oct. 16, 2014

Mission Control Team Agenda

Thursday October 16, 2014.
1530 — 1630 in JEB 37

Type of Meeting

Status Report and Next Three Week Planning

Attendees

David Klingenberg
 Taylor Trabun
 Brandon Ortiz

Topics

Topic	Responsible	Time (in minutes)
Progress Report	David & Taylor	10
Demonstrations	David & Taylor	10
New Tasks	Open Discussion	20
Question & Answers	Open Discussion	20

Additional Information:

Our next meeting will be in 3 weeks Nov 6, 2014.

4.6.1 Minutes from Thursday October 16 Meeting

- 1530 Meeting Start
- Update
 - Tatlor reported on snapshot day and his progress with the zigBee radios.
 - David
 - * Begin fine-tuning the drone for stabilization and self level flight. Drifting stability have been greatly improved.
- Action Items
 - David will continue to experiment with PWM and the quadcopter. He will explore control algorithms.
 - Taylor will continue his work on communications.
- Test Flight
 - Quadcopter severe forward drift has been improved. David needs to develop a battery frame to stop the batteries from shifting which is causing some of the uncontrolled drift.
- 1630 Meeting Ended

4.7 Nov. 6, 2014

Mission Control Team Agenda

**Thursday November 6, 2014.
1530 — 1630 in JEB 37**

Type of Meeting

Status Report and additional Short-term Planning.

Attendees

David Klingenberg
Taylor Trabun
Brandon Ortiz

Topics

Topic	Responsible	Time (in minutes)
Progress Report	David & Taylor	10
Demonstrations	David & Taylor	10
New Tasks	Open Discussion	20
Question & Answers	Open Discussion	20

Additional Information:

Our next meeting will be Nov 20, 2014.

4.7.1 Minutes from Thursday October 16 Meeting

- 1530 Meeting Start
- Update
 - Taylor
 - * Gui mock-up finished, class documentation work (design review presentation, wiki).
 - David
 - * Having a great deal of problem with PWM as an input to flight computer. Will have to try different firmware's for the flight computer and explore possible alternatives to PWM.
- New Resources
- Action Items

- David will continue to experiment with PWM and the quadcopter. Will explore control algorithms used by existing quad copters.
 - Taylor will continue his work on communications.
- Test Flight
 - Quadcopter severe forward drift has been improved. David needs to develop a battery frame to stop the batteries from shifting.
- 1630 Meeting Ended

4.8 Nov. 20, 2014

Mission Control Team Agenda

**Thursday November 20, 2014.
1530 — 1630 in JEB 37**

Type of Meeting

Status Report and additional Short-term Planning.

Attendees

David Klingenberg
Taylor Trabun
Brandon Ortiz

Topics

Topic	Responsible	Time (in minutes)
Progress Report	David & Taylor	10
Demonstrations	David & Taylor	10
New Tasks	Open Discussion	20
Question & Answers	Open Discussion	20

Additional Information:

Our next meeting will be Dec 4, 2014.

4.8.1 Minutes from Thursday October 16 Meeting

- 1530 Meeting Start
- Update
 - Taylor
 - * Worked with Brandon to debug XBee comms, still under-way Action Items 15min
 - David
 - * Focusing more on senior design and plans on continuing to work on project next semester, code written for PWM flight control
- Action Items
 - David is working on PWM, PWM flight service, team citizenship form.

- Taylor is working on design document, wiki update, team citizenship form, continue working out XBee comm bugs and send EXTERNAL_LCD TUN packet to another Arduino successfully.
- Test Flight
 - Broken bones and foul weather will place any future flight testing on hold.
- 1630 Meeting Ended

4.9 Jan. 22, 2015

Mission Control Team Agenda

Thursday January 22, 2015.
1100 — 1200 in JEB 30

Type of Meeting

Introductory Meeting

Attendees

David Klingenberg

Taylor Trabun

Emeth Thompson

Joe Higley

Topics Assign responsibilities to new members and bring new members "up to speed" on the state of the project.

Additional Information:

4.9.1 Minutes from Thursday January 22 Meeting

- 1100 Meeting Start
- All documents and software is on Github.
- goal: build GUI to mission plan for autonomous drones.
- review: hardware, communications, and GUI.
- goal: missions are expected to operate within visible range.
- goal: flight instruments for the GUI
- discussion: ideal design is modular with ability to add tools easily.
- hardware: need gps module
- goal: Basic Functionality
 - auto take-off
 - maintain position
 - auto-land

- move from point of origin to destination point
- 1200 Meeting Ended

4.10 Jan. 27, 2015

Mission Control Team Agenda

**Tuesday January 27, 2015.
1100 — 1200 in JEB 30**

Type of Meeting

Discussion

Attendees

David Klingenberg

Taylor Trabun

Emeth Thompson

Joe Higley

Brandon Ortiz

Topics Goals for GUI
Communications

Additional Information:

4.10.1 Minutes from Tuesday January, 27 Meeting

- 1100 Meeting Start
- Discussion: Communications
 - What information needs to be passed?
 - Packet design
- Brandon layed out goals and expectations
- 1200 Meeting Ended

4.11 Feb. 3, 2015

Mission Control Team Agenda

Tuesday February, 3 2015.
1100 — 1200 in JEB 30

Type of Meeting

Demonstration and Discussion

Attendees

David Klingenberg

Taylor Trabun

Emeth Thompson

Joe Higley

Brandon Ortiz

Topics Joe's GUI prototype

Additional Information:

4.11.1 Minutes from Tuesday February, 3 Meeting

- 1100 Meeting Start
- Demonstration: GUI prototype
- GUI needs topographical data and potentially the ability to map gps
- Real-time Controls
 - emergency land button
 - stop and hover button
- Instrument Panel
 - artificial horizon
 - vertical speed indicator
 - dial compass
 - two-minute turn coordinator
 - speed: number in a box
 - altitude: number in a box

- BRANDON
 - need to make extra propellers
 - flight tests: drone debugging - David is making parts via 3D printing
 - Goal: need to be able to take-off -> hover -> land via communications of a mission plan and real-time controls
 - self take-off
 - altitude control
 - landing
- Organize: Design review
 - Taylor - communications
 - David - hardware
 - Emeth - Documentation and slides
 - Joe - GUI
- Assignment: Emeth - investigate topographical mapping or google maps
- Assignment: Joe - Serial Communication and Xapi
- 1210 Meeting Ended

4.12 Feb. 17, 2015

Mission Control Team Agenda

Tuesday February 17, 2015.
1100 — 1200 in JEB 30

Type of Meeting

Discussion and Review

Attendees

David Klingenberg

Taylor Trabun

Emeth Thompson

Joe Higley

Topics Discuss design review and update goals

Additional Information:

4.12.1 Minutes from Tuesday January, 27 Meeting

- 1100 Meeting Start
- design review went well
- Taylor and Joe combined the mission control GUI and serial port terminal.
- 1200 Meeting Ended

4.13 March 5, 2015

Mission Control Team Agenda

**Thursday March 5, 2015.
1100 — 1200 in JEB 30**

Type of Meeting

Working Meeting

Attendees

David Klingenberg

Taylor Trabun

Emeth Thompson

Joe Higley

Topics Discuss and prepare for upcoming snapshot day

Additional Information:

4.13.1 Minutes from March 5, 2015 Meeting

- 1100 Meeting Start
- Taylor and Joe work on the software side
- David and Emeth discussed the poster
- 1200 Meeting Ended

5 Code

5.1 Supplemental quad copter autopilot V1.8

```
1 //ifndef AutoPilot_H
2 //define AutoPilot_H
3
4 /*****
5 Supplemental quad copter Autopilot.
6 Arthur David Klingenberg
7
8
9 Contains a prototype controller algorithm
10 to implement an autopilot.
11
12 4/25/2015
13 Version 1.8
14 — Final version for expo
15
16 added SD card logging
17 added diagnostic LEDs
18 reworked HOLD PID.
19 added waypoints
20 added drone stats.
21 added rx heartbeat to shut down
22 drone if the signal is lost.
23 added throttle kill if master autopilot
24 is turned off to force drone to land.
25
26
27 Important numbers for the kk2.1
28
29 center satick :1500 micro seconds
30 Full Right/Back: 2100
31 Full Left/Forward: 900
32
33 100% Throttle: 2300
34 0% Throttle: 1000
35
36 X-bee is on com 2 (serial 02).
37 GPS is on com 2 (serial 03).
38
39 version history
```

```

40 4/15/2015 1.7
41 —First attempted at berring hold.
42
43
44 4/3/2015 1.6
45 —Compleatly scraped and replaced RX pass throught code.
46
47 3/24/2015 1.5
48 —added passthrought controll for rx.
49
50 12/10/2014 1.4
51 Contains a prototype controller algorithm
52 to maintain an altitude hold. In this
53 version only an ultrasonic rangefinder
54 is used to measure altitude.
55
56 1.0 throught 1.3 fall 2015
57 sensore expermentations.
58
59 *****/
60 /* Debug and log file settings. */
61 //Change to a one to enable debug LEDs
62 #define DEBUG 1
63 #define LOG_NAME 04262015001
64
65 /* GPS PINS */
66 #define FIX 45
67
68 /* INPUT RANGE */
69 #define ZERO_THROTTLE 1000
70 #define FULL_THROTTLE 2000
71
72 #define FULL_STICK_LEFT_FORWARD 1000
73 #define FULL_STICK_RIGHT_BACK 2000
74 #define ZERO_STICK 1500
75
76 /* Defualt test hold data. */
77 #define HOLD_HEADING 270
78 #define HOLD_ALTITUDE 50
79
80
81 /* PWM pin assignments. */

```

```

82 #define AILERONS_PIN 5
83 #define ELEVATOR_PIN 6
84 #define THROTTLE_PIN 7
85 #define RUDDER_PIN 8
86 #define AUX 9
87
88 /* Hand held RX inputs assignments. */
89 #define RX_AILERONS A15
90 #define RX_ELEVATOR A14
91 #define RX_THROTTLE A13
92 #define RX_RUDDER A12
93 #define RX_AUX A11
94
95 /* RX Max miss pulse length */
96 // Used in rx_on() function.
97 #define SAFE_PULSE 320
98
99 /* RX center stick values */
100 #define HIGH_CENTER_STICK 1515
101 #define LOW_CENTER_STICK 1475
102
103 /* Bitflag used in interups to indecate a
104 new signal on a channel.
105
106 !!!Note this is not a pin number!!!
107 This is a power of 2. Next up would be 32,
108 64, and 128. If armed is moved hear use 128.*//
109
110 #define ALI_FLAG 1
111 #define ELE_FLAG 2
112 #define THR_FLAG 4
113 #define RUD_FLAG 8
114 #define AUX_FLAG 16
115
116 /* Bitflag used to indecate autopilot status.
117 This ius a power of 2. valied numbers are
118 1,2,4,8,16,32,64,128
119 0x01 0x02 0x04 0x08 0x10 0x20 0x40 0x80
120 0b00000001 0b00000010 ... 0b10000000
121 */
122
123 #define HEADING_FLAG 1 //Set autopilot to hold a heading.

```

```

124 #define ALTHOLD.FLAG    2    //
125 #define LATHOLD.FLAG    4    //
126 #define LONGHOLD.FLAG   8    //
127
128 #define ARM.FLAG         16    //
129 #define IS_ARMED.FLAG    32    //
130 #define MASTER_ON.FLAG   64    //
131 #define MASTER_AUTOPILOT 128   //Set by aux switch on remote.
132
133 /* Sonar Setup */
134 #define GROUND_PING_PIN    27
135 #define GROUND_ECHO_PIN    26
136 #define GROUND_MAX_SONAR_DISTANCE 200
137 #define GROUND_SONAR_ITERATION 5
138 #define CALIBRATION_ERROR 9
139
140 /* PID Setup */
141 // #define thr_out_range 1.25
142 #define Kp                40
143 #define Ki                10
144 #define Kd                1
145 #define auxset_add        3
146 #define pidMode_add       4
147 #define SAMPLE_TIME      100
148
149 /* LED Setup */
150 #define LED_UPDATE_DELAY   100
151 #define LED_PIN            23
152 #define NUMPIXELS          16
153 #define HALFPixels         8    // Divide the pixels in have to
    allow one side of deone to be a defrent color.
154 #define L_SIDE            50,0,0    // For max intesity of red set
    255,0,0
155 #define R_SIDE            0,50,0    // For max green set to 0,255,0
156 #define MAUTOPIOT_ON      50,50,50    // White
157 #define AUTOPILOT_HOLD_ON 0,0,50    // Blue
158 struct drone_state
159 {
160     int    current_head; // deg. The current heading.
161     int    hold_head;    // deg. The heading to hold, this is
    independend of
162                                     //    course.

```

```

163 double    current_alt;    // cm Above sea level(ASL).
164 double    hold_alt;      // cm Altitude to hold above sea level(ASL).
165 double    ground_alt;    // cm The ground level when armed(AGL) ie.
        ASL when armed.
166 double    sonar_alt;     // cm The returned from range finder(AGL).
167 double    current_lon;   // Longitude
168 double    target_lon;    // Longitude I want to go to.
169 double    current_lat;   // Latatude
170 double    target_lat;    // Latatide I want to travil to.
171 unsigned int    loiter_time; // Time on station when drone arives.
172 };
173
174 struct waypoint
175 {
176     waypoint    *next_waypoint; // Pointer to next waypoint.
177     double    hold_head;        // deg. The heading to hold, this is
        independend of
178                                     // course.
179     double    hold_alt;        // cm Altitude to hold above sea
        level(ASL).
180     double    target_lat;      // longitude drone needs to go to.
181     double    target_lon;      // latatide drone needs to travil to.
182     unsigned int    loiter_time; // Time on station when drone arives.
183 };
184
185
186
187
188 typedef struct drone_state drone_state;
189 typedef struct waypoint waypoint;
        ./AutoPilot.h

```

```

1 #include <Serial_service.h>
2 #include <LiquidCrystal.h>
3 #include <LCD_service.h>
4 #include <Xapi.h>
5 #include <Subscriptions.h>
6 #include <Universal.h>
7 #include <Util.h>
8 #include <Single_buff.h>
9 #include <Land_service.h>
10 #include <Takeoff_service.h>
11 #include <DoMove_service.h>

```

```

12 #include <Arm_service.h>
13 #include <Heartbeat_service.h>
14
15 //0013a200
16 //40a1446d
17
18 //*****
19 //*****
20 Xapi xapi = Xapi(Serial);
21 //Serial_service serial_service = Serial_service(Serial1, xapi);
22 LCD_service lcd_service (xapi);
23 //*****
24 // The next services pertain to drone instructions
25 //*****
26 Takeoff_service takeoff_service(xapi, lcd_service);
27 Land_service land_service(xapi, lcd_service);
28 DoMove_service doMove_service(xapi, lcd_service);
29 Arm_service arm_service(xapi, lcd_service);
30 //Heartbeat_service heartbeat_service(xapi, lcd_service);
31 //Serial_service serial_service = Serial_service(Serial1, xapi,
    lcd_service);
32 uint8_t msg1[] = "I FEEL GREAT";
33 uint8_t msg2[] = "COMMODORE 64";
34 uint8_t _clear[] = " ";
35
36
37 //*****
38 //*****
39 void setup()
40 {
41   Serial.begin(MISC_PC_SPEED);
42   //Serial1.begin(MISC_PC_SPEED);
43
44
45
46 }
47
48 //*****
49 //*****
50 void loop()
51 {
52   //system_active();

```

```

53  //process_buttons();
54  xapi.xapi_latch();
55  lcd_service.lcd_service_latch();
56  takeoff_service.takeoff_service_latch();
57  land_service.land_service_latch();
58  doMove_service.DoMove_service_latch();
59  arm_service.arm_service_latch();
60  //heartbeat_service.heartbeat_service_latch();
61  //serial_service.serial_service_latch();
62
63  //delay(4000);
64 }
65
66
67 //*****
68 // Old function. Not needed
69 //*****
70 void system_active()
71 {
72     static uint16_t cnt = 0;
73     static uint8_t row = 0;
74
75     cnt++;
76
77     if ( (cnt%2500) == 0)
78     {
79         // turn off both stars
80         lcd_service.lcd_print(15, 0, (const char*)" ");
81         lcd_service.lcd_print(15, 1, (const char*)" ");
82
83         row++;
84         // turn on new row
85         lcd_service.lcd_print(15, row%2, (const char*)"*");
86
87     }
88
89 }

```

./XAPI_DRONE_DEBUG_COMMS.c

5.2 Home Arduino Firmware

```

1 #include <Serial_service.h>
2 #include <LiquidCrystal.h>

```



```

3 #include <LCD_service.h>
4 #include <Xapi.h>
5 #include <Subscriptions.h>
6 #include <Universal.h>
7 #include <Util.h>
8 #include <Single_buff.h>
9
10 // #include <Arm_service.h>
11 // #include <Heartbeat_service.h>
12 // #include <AltHold_service.h>
13 // #include <Heading_service.h>
14
15 //0013a200
16 //40a1446d
17
18 //*****
19 //*****
20 Xapi xapi = Xapi(Serial);
21 //Serial_service serial_service = Serial_service(Serial1, xapi);
22 LCD_service lcd_service(xapi);
23 Serial_service serial_service = Serial_service(Serial1, xapi,
        lcd_service);
24 //Arm_service arm_service(xapi, lcd_service);
25 //Heartbeat_service heartbeat_service(xapi, lcd_service);
26 uint8_t msg1[] = "I FEEL GREAT";
27 uint8_t msg2[] = "COMMODORE 64";
28 uint8_t _clear[] = " ";
29
30
31 //*****
32 //*****
33 void setup()
34 {
35   Serial.begin(MISC_PC_SPEED);
36   Serial1.begin(MISC_PC_SPEED);
37
38
39
40 }
41
42 //*****
43 //*****

```

```

44 void loop()
45 {
46     system_active();
47     process_buttons();
48     xapi.xapi_latch();
49     lcd_service.lcd_service_latch();
50     serial_service.serial_service_latch();
51     //arm_service.arm_service_latch();
52     //heartbeat_service.heartbeat_service_latch();
53     //delay(4000);
54 }
55
56 //*****
57 //*****
58
59 void process_buttons()
60 {
61     // storage for the button
62     int button;
63
64     button = lcd_service.get_lcd_key();
65
66     // process packet
67     if(button == LCD_btnSELECT)
68     {
69
70         lcd_service.lcd_snd_local_serial_debug((const uint8_t*)"SELECT");
71         //lcd_service.lcd_snd_LOCAL_message(0,0,(const
72             uint8_t*)"VICTORY(C)");
73         lcd_service.lcd_snd_EXTERNAL_message(ADDR_MSB, 0x40a1446d,
74             ADDR16_BROADCAST,
75             0,0,(const
76                 uint8_t*)"VICTORY(EN)");
77
78         /*
79         lcd_service.lcd_snd_EXTERNAL_message(  DEBUG_MSB_ADDR,
80             DEBUG_LSB_ADDR,
81             DEBUG_ADDR16,
82             0,
83             0,
84             _clear);

```

```

83
84
85
86     lcd_service.lcd_snd_EXTERNAL_message(  DEBUG_MSB_ADDR,
87         DEBUG_LSB_ADDR,
88         DEBUG_ADDR16,
89         0,
90         0,
91         msg1);
92
93     lcd_service.lcd_print(0, 0, (const char*)"first message");
94     lcd_service.lcd_print(0,1, (const char*)"");
95
96 */
97 }
98
99 if(button == LCD_btnLEFT )
100 {
101
102     lcd_service.lcd_snd_LOCAL_message(0,0,(const uint8_t*)"GOOD
        JERB!!!!!!");
103
104     /*
105     lcd_service.lcd_snd_EXTERNAL_message(  DEBUG_MSB_ADDR,
106         DEBUG_LSB_ADDR,
107         DEBUG_ADDR16,
108         0,
109         1,
110         _clear);
111
112
113     lcd_service.lcd_snd_EXTERNAL_message(  DEBUG_MSB_ADDR,
114         DEBUG_LSB_ADDR,
115         DEBUG_ADDR16,
116         0,
117         1,
118         msg2);
119
120     lcd_service.lcd_print(0, 1, (const char*)"second message");
121     lcd_service.lcd_print(0,0, (const char*)"");
122
123     */

```

```

124     }
125 }
126
127 //*****
128 //*****
129 void system_active()
130 {
131     static uint16_t cnt = 0;
132     static uint8_t row = 0;
133
134     cnt++;
135
136     if ( (cnt%2500) == 0)
137     {
138         // turn off both stars
139         lcd_service.lcd_print(15, 0, (const char*)" ");
140         lcd_service.lcd_print(15, 1, (const char*)" ");
141
142         row++;
143         // turn on new row
144         lcd_service.lcd_print(15, row%2, (const char*)"*");
145
146     }
147
148 }

```

./commCode/XAPI_HOME_ARDUINO.ino

5.3 xAPI Services

5.3.1 Do Move Service

```

1 #ifndef DOMOVE_SERVICE.h
2 #define DOMOVE_SERVICE.h
3 #include <arduino.h>
4 #include <Xapi.h>
5 #include <Util.h>
6 #include <LCD_service.h>
7
8
9 //*****
10 //*****
11
12 class DoMove_service

```

```

13 {
14     // objects used
15     private:
16         Xapi& m_xapi;
17         Util m_util;
18         LCD_service& m_lcd;
19     private:
20
21
22     // functions for do_move service
23     private:
24         void reset_TUN_storage();
25         void process_local_TUN_packet();
26         void process_external_TUN_packet();
27
28     // general functions for do_move
29     public:
30         void DoMove_service_latch();
31
32     // Constructor
33     public:
34         DoMove_service(Xapi& _xapi, LCD_service& _lcd);
35 };
36
37 #endif

```

./commCode/DoMove_service.h

```

1
2 #ifndef DOMOVE_SERVICE_cpp
3 #define DOMOVE_SERVICE_cpp
4 #include <DoMove_service.h>
5
6 //*****
7 // This latch is what is called in the microcontroller's
8 // main loop. Put any required processing here
9 //*****
10 void DoMove_service::DoMove_service_latch()
11 {
12
13     // process any local LCD message packets
14     process_local_TUN_packet();
15
16     // process any external LCD message packets

```

```

17  process_external_TUN_packet();
18 }
19
20
21
22 //*****
23 //*****
24 // This routine will query the XAPI to see
25 // if there is a local message waiting for
26 // the Land service. If so, we need to grab it and react.
27 void DoMove_service::process_external_TUN_packet()
28 {
29     // see if there is a packet waiting
30     if(m_xapi.CONNECT_external_TUN_get_type() ==
        TUN_TYPE_EXTERNAL_DO_MOVE)
31     {
32         // allocate the space
33         uint8_t TUN_packet[MED_BUFF_SZ];
34
35         // extract the packet
36         m_xapi.CONNECT_external_TUN_get_packet(TUN_packet, MED_BUFF_SZ);
37
38         // do something
39         //lcd prints are for debugging, should be removed
40         m_lcd.lcd_print(0,0,"do move");
41
42
43
44     }
45 }
46
47 //*****
48 //*****
49 // This routine will query the XAPI to see
50 // if there is a local message waiting for
51 // the Land service. If so, we need to grab it and do something.
52 void DoMove_service::process_local_TUN_packet()
53 {
54     // see if there is a packet waiting
55     if(m_xapi.CONNECT_local_TUN_get_type() == TUN_TYPE_LOCAL_DO_MOVE)
56     {
57         // allocate the space

```

```

58     uint8_t TUN_packet[MED_BUFF_SZ];
59
60     // extract the packet
61     m_xapi.CONNECT_local_TUN_get_packet(TUN_packet, MED_BUFF_SZ);
62
63     // do something
64
65 }
66 }
67
68 *****
69 *****
70 // Resets the TUN packet storage
71 void DoMove_service::reset_TUN_storage()
72 {
73     // obsolete
74 }
75
76 *****
77 *****
78 DoMove_service::DoMove_service(Xapi& _xapi, LCD_service& _lcd):
79 m_xapi(_xapi), m_lcd(_lcd)
80 {
81     reset_TUN_storage();
82
83 }
84
85
86 #endif

```

./commCode/DoMove_service.cpp

5.3.2 Heading Service

```

1 #ifndef HEADING_SERVICE_h
2 #define HEADING_SERVICE_h
3 #include <arduino.h>
4 #include <Xapi.h>
5 #include <Util.h>
6 #include <LCD_service.h>
7
8
9 *****
10 *****

```

```

11
12 class Heading_service
13 {
14     // objects used
15     private:
16         Xapi& m_xapi;
17         Util m_util;
18         LCD_service& m_lcd;
19     private:
20
21
22     // functions for heading service
23     private:
24         void reset_TUN_storage();
25         void process_local_TUN_packet();
26         void process_external_TUN_packet();
27
28     // general functions for heading
29     public:
30         void heading_service_latch();
31
32     // Constructor
33     public:
34         Heading_service(Xapi& _xapi, LCD_service& _lcd);
35 };
36
37 #endif

```

./commCode/Heading_service.h

```

1 #ifndef HEADING_SERVICE_cpp
2 #define HEADING_SERVICE_cpp
3 #include <AutoPilot.h>
4 #include <Heading_service.h>
5
6 extern uint8_t bit_autopilot_flags;
7 extern drone_state *P_state;
8 //*****
9 // This latch is what is called in the microcontroller's
10 // main loop. Put any required processing here
11 //*****
12 void Heading_service::heading_service_latch()
13 {
14

```



```

15 // process any local LCD message packets
16 process_local_TUN_packet();
17
18 // process any external LCD message packets
19 process_external_TUN_packet();
20 }
21
22
23
24 //*****
25 //*****
26 // This routine will query the XAPI to see
27 // if there is a local message waiting for
28 // the Land service. If so, we need to grab it and react.
29 void Heading_service::process_external_TUN_packet()
30 {
31 // see if there is a packet waiting
32 if(m_xapi.CONNECT_external_TUN_get_type() ==
    TUN_TYPE_EXTERNAL_SET_HEADING)
33 {
34 // allocate the space
35 uint8_t TUN_packet[MED_BUFF_SZ];
36 uint8_t payload_buff[SMALL_BUFF_SZ];
37 uint8_t payload_buff_sz = 0;
38 int heading = 0;
39 // extract the packet
40 m_xapi.CONNECT_external_TUN_get_packet(TUN_packet, MED_BUFF_SZ);
41
42 //extract payload
43 payload_buff_sz = m_util.get_TUN_payload(TUN_packet, payload_buff,
    SMALL_BUFF_SZ);
44
45 //grab height from payload (2s bytes)
46 heading = m_util.hex_to_int(0, 2, payload_buff_sz, payload_buff);
47
48 P_state->hold_head = heading;
49 // do something
50 //lcd prints are for debugging, should be removed
51 //m_lcd.lcd_print(0,0,"*****");
52 //m_lcd.lcd_print(0,0,"Got Takeoff");
53 //m_lcd.lcd_print(0,0,"ttest1");
54 //m_lcd.lcd_print(0,0,"takeoff");

```

```

55     //m_lcd.lcd_print(0,0,"ttest2");
56     //if(height = 10){
57     //    m_lcd.lcd_print(0,0,"Height 10");
58     //}
59     //P_state->hold_alt = P_state->ground_alt + height;
60     //bit_autopilot_flags |= ALTHOLD_FLAG;
61 }
62 }
63
64 //*****
65 //*****
66 // This routine will query the XAPI to see
67 // if there is a local message waiting for
68 // the Land service. If so, we need to grab it and do something.
69 void Heading_service::process_local_TUN_packet()
70 {
71     // see if there is a packet waiting
72     if(m_xapi.CONNECT_local_TUN_get_type() ==
        TUN_TYPE_LOCALSET_HEADINGS)
73     {
74         // allocate the space
75         uint8_t TUN_packet[MED_BUFF_SZ];
76
77         // extract the packet
78         m_xapi.CONNECT_local_TUN_get_packet(TUN_packet, MED_BUFF_SZ);
79
80         // do something
81
82     }
83 }
84
85 //*****
86 //*****
87 // Resets the TUN packet storage
88 void Heading_service::reset_TUN_storage()
89 {
90     // obsolete
91 }
92
93 //*****
94 //*****
95 Heading_service::Heading_service(Xapi& _xapi, LCD_service& _lcd):

```

```

96 m_xapi(_xapi), m_lcd(_lcd)
97 {
98     reset_TUN_storage();
99
100 }
101
102
103 #endif

```

./commCode/Heading_service.cpp

5.3.3 Heading Hold Service

```

1 #ifndef HEADINGHOLD_SERVICE_h
2 #define HEADINGHOLD_SERVICE_h
3 #include <arduino.h>
4 #include <Xapi.h>
5 #include <Util.h>
6 #include <LCD_service.h>
7
8
9 //*****
10 //*****
11
12 class HeadingHold_service
13 {
14     // objects used
15     private:
16         Xapi& m_xapi;
17         Util m_util;
18         LCD_service& m_lcd;
19     private:
20
21
22     // functions for headinghold service
23     private:
24         void reset_TUN_storage();
25         void process_local_TUN_packet();
26         void process_external_TUN_packet();
27
28     // general functions for althold
29     public:
30         void headingHold_service_latch();
31

```

```

32  // Constructor
33  public:
34      HeadingHold_service(Xapi& _xapi, LCD_service& _lcd);
35  };
36
37 #endif

```

./commCode/HeadingHold_service.h

```

1 #ifndef HEADINGHOLD_SERVICE.cpp
2 #define HEADINGHOLD_SERVICE.cpp
3 #include <AutoPilot.h>
4 #include <HeadingHold_service.h>
5
6 extern uint8_t bit_autopilot_flags;
7 extern drone_state *P_state;
8 //*****
9 // This latch is what is called in the microcontroller's
10 // main loop. Put any required processing here
11 //*****
12 void HeadingHold_service::headingHold_service_latch()
13 {
14
15     // process any local LCD message packets
16     process_local_TUN_packet();
17
18     // process any external LCD message packets
19     process_external_TUN_packet();
20 }
21
22
23
24 //*****
25 //*****
26 // This routine will query the XAPI to see
27 // if there is a local message waiting for
28 // the Land service. If so, we need to grab it and react.
29 void HeadingHold_service::process_external_TUN_packet()
30 {
31     // see if there is a packet waiting
32     if(m_xapi.CONNECT_external_TUN_get_type() ==
        TUN_TYPE_EXTERNAL_HEADING_HOLD)
33     {
34         // allocate the space

```

```

35     uint8_t TUN_packet[MED_BUFF_SZ];
36     uint8_t payload_buff[SMALL_BUFF_SZ];
37     uint8_t payload_buff_sz = 0;
38     int hold = 0;
39     // extract the packet
40     m_xapi.CONNECT_external_TUN_get_packet(TUN_packet, MED_BUFF_SZ);
41
42     //extract payload
43     payload_buff_sz = m_util.get_TUN_payload(TUN_packet, payload_buff,
        SMALL_BUFF_SZ);
44
45     //grab height from payload (4 bytes)
46     hold = m_util.hex_to_int(0, 1, payload_buff_sz, payload_buff);
47     switch(hold){
48         case 0:
49             //activate longitude hold
50             bit_autopilot_flags |= LONGHOLD_FLAG;
51             break;
52         case 1:
53             //disable longitude hold
54             bit_autopilot_flags &= ~LONGHOLD_FLAG;
55             break;
56
57         case 2:
58             //activate lat hold
59             bit_autopilot_flags |= LATHOLD_FLAG;
60             break;
61
62         case 3:
63             //disable lat hold
64             bit_autopilot_flags &= ~LATHOLD_FLAG;
65             break;
66     }
67
68     // do something
69     //lcd prints are for debugging, should be removed
70     //m_lcd.lcd_print(0,0,"*****");
71     //m_lcd.lcd_print(0,0,"Got Takeoff");
72     //m_lcd.lcd_print(0,0,"ttest1");
73     //m_lcd.lcd_print(0,0,"takeoff");
74     //m_lcd.lcd_print(0,0,"ttest2");
75     //if(height = 10){

```

```

76     // m_lcd.lcd_print(0,0,"Height 10");
77     //}
78     //P_state->hold_alt = P_state->ground_alt + height;
79     //bit_autopilot_flags |= ALTHOLD_FLAG;
80 }
81 }
82
83 //*****
84 //*****
85 // This routine will query the XAPI to see
86 // if there is a local message waiting for
87 // the Land service. If so, we need to grab it and do something.
88 void HeadingHold_service::process_local_TUN_packet()
89 {
90     // see if there is a packet waiting
91     if(m_xapi.CONNECT_local_TUN_get_type() ==
        TUN_TYPE_LOCAL_HEADING_HOLD)
92     {
93         // allocate the space
94         uint8_t TUN_packet[MED_BUFF_SZ];
95
96         // extract the packet
97         m_xapi.CONNECT_local_TUN_get_packet(TUN_packet, MED_BUFF_SZ);
98
99         // do something
100
101     }
102 }
103
104 //*****
105 //*****
106 // Resets the TUN packet storage
107 void HeadingHold_service::reset_TUN_storage()
108 {
109     // obsolete
110 }
111
112 //*****
113 //*****
114 HeadingHold_service::HeadingHold_service(Xapi& _xapi, LCD_service&
    _lcd):
115 m_xapi(_xapi), m_lcd(_lcd)

```

```

116 {
117     reset_TUN_storage();
118
119 }
120
121
122 #endif

```

./commCode/HeadingHold_service.cpp

5.3.4 Heartbeat Service

```

1 #ifndef HEARTBEAT_SERVICE_h
2 #define HEARTBEAT_SERVICE_h
3 #include <arduino.h>
4 #include <Xapi.h>
5 #include <Util.h>
6 #include <LCD_service.h>
7
8
9 //*****
10 //*****
11
12 class Heartbeat_service
13 {
14     // objects used
15     private:
16         Xapi& m_xapi;
17         Util m_util;
18         LCD_service& m_lcd;
19         unsigned long lastSent;
20
21     private:
22
23
24     // functions for land service
25     private:
26         void reset_TUN_storage();
27         void process_local_TUN_packet();
28         void process_external_TUN_packet();
29         void send_heartbeat();
30
31     // general functions for landing
32     public:

```

```

33     void heartbeat_service_latch();
34
35     // Constructor
36     public:
37         Heartbeat_service(Xapi& _xapi, LCD_service& _lcd);
38 };
39
40 #endif

```

./commCode/Heartbeat_service.h

```

1 #ifndef HEARTBEAT_SERVICE_cpp
2 #define HEARTBEAT_SERVICE_cpp
3 #include <Heartbeat_service.h>
4
5 //*****
6 // This latch is what is called in the microcontroller's
7 // main loop. Put any required processing here
8 //*****
9 void Heartbeat_service::heartbeat_service_latch()
10 {
11     unsigned long currTime = millis();
12     // process any local LCD message packets
13     process_local_TUN_packet();
14
15     // process any external LCD message packets
16     process_external_TUN_packet();
17     //millis() will rollover to 0 after 50 days, extra provisioning
18     if(currTime < lastSent){
19         lastSent = currTime;
20     }
21     //if 1 second has passed send heartbeat
22     if((currTime-lastSent) > 1000){
23         send_heartbeat();
24         lastSent = millis();
25     }
26 }
27
28
29
30 //*****
31 //*****
32 // This routine will query the XAPI to see
33 // if there is a local message waiting for

```



```

34 // the Land service. If so, we need to grab it and react.
35 void Heartbeat_service::process_external_TUN_packet()
36 {
37     // see if there is a packet waiting
38     if(m_xapi.CONNECT_external_TUN_get_type() ==
        TUN_TYPE_EXTERNAL_HEARTBEAT)
39     {
40         // allocate the space
41         uint8_t TUN_packet[MED_BUFF_SZ];
42
43         // extract the packet
44         m_xapi.CONNECT_external_TUN_get_packet(TUN_packet, MED_BUFF_SZ);
45
46         // do something
47         //lcd prints are for debugging, should be removed
48         //m_lcd lcd_print("*****");
49         //m_lcd lcd_print(0,0,"ltest1");
50         m_lcd lcd_print(0,0,"heartbeat packet");
51         //m_lcd lcd_print(0,0,"ltest2");
52     }
53 }
54
55 *****
56 *****
57 // This routine will query the XAPI to see
58 // if there is a local message waiting for
59 // the Land service. If so, we need to grab it and do something.
60 void Heartbeat_service::process_local_TUN_packet()
61 {
62     // see if there is a packet waiting
63     if(m_xapi.CONNECT_local_TUN_get_type() == TUN_TYPE_LOCAL_HEARTBEAT)
64     {
65         // allocate the space
66         uint8_t TUN_packet[MED_BUFF_SZ];
67
68         // extract the packet
69         m_xapi.CONNECT_local_TUN_get_packet(TUN_packet, MED_BUFF_SZ);
70
71         // do something
72     }
73 }
74 *****

```

```

75 //*****
76 // This function is used to send a external heartbeat
77 // packet, which contains drone information
78 void Heartbeat_service::send_heartbeat()
79 {
80     uint32_t msb = 0x0;
81     uint16_t lsb = 0xffff;
82
83
84     uint8_t x = 0;
85     uint8_t y = 0;
86     uint8_t LCD_payload_sz = 0;
87     uint8_t packet_size = 0;
88     // storage for payload
89     //uint8_t payload[] = {0x00};
90     //uint8_t message[MED_BUFF_SZ];
91     // storage for new packet
92     //uint8_t new_TUN_packet[LARGE_BUFF_SZ];
93     //uint8_t new_TUN_packet_sz = 0;
94
95     uint8_t payload_buff_sz = 0;
96
97     uint8_t payload_buff[MED_BUFF_SZ];
98     uint8_t packet_buff[LARGE_BUFF_SZ];
99
100     uint8_t TUN_buff[LARGE_BUFF_SZ];
101     uint8_t TUN_buff_sz = 0;
102
103
104     //m_lcd.lcd_print(0,0,"first");
105     //payload_buff_sz = m_util.get_TUN_payload(buff, payload_buff,
106         MED_BUFF_SZ);
107     payload_buff[0] = 0x00; //temp code
108     payload_buff_sz = 2;
109     //m_lcd.lcd_print(0,0,"second");
110
111     TUN_buff_sz = m_util.create_TUN_packet( TUN_TYPEEXTERNALHEARTBEAT,
112         payload_buff,
113         payload_buff_sz,
114         TUN_buff,
115         LARGE_BUFF_SZ);
116     //construct and send packet

```

```

116         //m_lcd.lcd_print(0,0,"third");
117
118
119     //TUN_buff[0] = 'X';
120     //TUN_buff[1] = 'X';
121     //TUN_buff[2] = 'X';
122     //TUN_buff_sz = 3;
123
124     packet_size = m_xapi.construct_transmit_req(msb,
125                                                lsb,
126                                                ADDR16_BROADCAST,
127                                                TUN_buff,
128                                                TUN_buff_sz,
129                                                packet_buff,
130                                                LARGE_BUFF_SZ);
131     //m_lcd.lcd_print(0,0,"fourth");
132
133
134     m_xapi.snd_packet(packet_buff, packet_size);
135
136 }
137
138 //*****
139 //*****
140 // Resets the TUN packet storage
141 void Heartbeat_service::reset_TUN_storage()
142 {
143     // obsolete
144 }
145
146 //*****
147 //*****
148 Heartbeat_service::Heartbeat_service(Xapi& _xapi, LCD_service& _lcd):
149 m_xapi(_xapi), m_lcd(_lcd)
150 {
151     reset_TUN_storage();
152     //send first heartbeat
153     send_heartbeat();
154     //initialize lastSend
155     lastSent = millis();
156
157 }

```

```

158
159
160 #endif

```

./commCode/Heartbeat_service.cpp

5.3.5 Land Service

```

1 #ifndef LAND_SERVICE_h
2 #define LAND_SERVICE_h
3 #include <arduino.h>
4 #include <Xapi.h>
5 #include <Util.h>
6 #include <LCD_service.h>
7
8
9 //*****
10 //*****
11
12 class Land_service
13 {
14     // objects used
15     private:
16         Xapi& m_xapi;
17         Util m_util;
18         LCD_service& m_lcd;
19
20     private:
21
22
23     // functions for land service
24     private:
25         void reset_TUN_storage();
26         void process_local_TUN_packet();
27         void process_external_TUN_packet();
28
29     // general functions for landing
30     public:
31         void land_service_latch();
32
33     // Constructor
34     public:
35         Land_service(Xapi& _xapi, LCD_service& _lcd);
36 };

```

```

37
38 #endif

                                   ./commCode/Land_service.h

1 #ifndef LAND_SERVICE.cpp
2 #define LAND_SERVICE.cpp
3 #include <Land_service.h>
4
5 //*****
6 // This latch is what is called in the microcontroller's
7 // main loop. Put any required processing here
8 //*****
9 void Land_service::land_service_latch()
10 {
11
12     // process any local LCD message packets
13     process_local_TUN_packet();
14
15     // process any external LCD message packets
16     process_external_TUN_packet();
17 }
18
19
20
21 //*****
22 //*****
23 // This routine will query the XAPI to see
24 // if there is a local message waiting for
25 // the Land service. If so, we need to grab it and react.
26 void Land_service::process_external_TUN_packet()
27 {
28     // see if there is a packet waiting
29     if(m_xapi.CONNECT_external_TUN_get_type() == TUN.TYPE.EXTERNALLAND)
30     {
31         // allocate the space
32         uint8_t TUN_packet[MED_BUFF_SZ];
33
34         // extract the packet
35         m_xapi.CONNECT_external_TUN_get_packet(TUN_packet, MED_BUFF_SZ);
36
37         // do something
38         //lcd prints are for debugging, should be removed
39         //m_lcd.lcd_print("*****");

```

```

40     m_lcd.lcd_print(0,0,"ltest1");
41     m_lcd.lcd_print(0,0,"land");
42     m_lcd.lcd_print(0,0,"ltest2");
43 }
44 }
45
46 //*****
47 //*****
48 // This routine will query the XAPI to see
49 // if there is a local message waiting for
50 // the Land service. If so, we need to grab it and do something.
51 void Land_service::process_local_TUN_packet()
52 {
53     // see if there is a packet waiting
54     if(m_xapi.CONNECT_local_TUN_get_type() == TUN_TYPE_LOCALLAND)
55     {
56         // allocate the space
57         uint8_t TUN_packet[MED_BUFF_SZ];
58
59         // extract the packet
60         m_xapi.CONNECT_local_TUN_get_packet(TUN_packet, MED_BUFF_SZ);
61
62         // do something
63     }
64 }
65
66 //*****
67 //*****
68 // Resets the TUN packet storage
69 void Land_service::reset_TUN_storage()
70 {
71     // obsolete
72 }
73
74 //*****
75 //*****
76 Land_service::Land_service(Xapi& _xapi, LCD_service& _lcd):
77 m_xapi(_xapi), m_lcd(_lcd)
78 {
79     reset_TUN_storage();
80
81 }

```

82
83
84 **#endif**

./commCode/Land_service.cpp

5.3.6 Takeoff Service

```
1 #ifndef TAKEOFF_SERVICE_h
2 #define TAKEOFF_SERVICE_h
3 #include <arduino.h>
4 #include <Xapi.h>
5 #include <Util.h>
6 #include <LCD_service.h>
7
8
9 //*****
10 //*****
11
12 class Takeoff_service
13 {
14     // objects used
15     private:
16         Xapi& m_xapi;
17         Util m_util;
18         LCD_service& m_lcd;
19     private:
20
21
22     // functions for takeoff service
23     private:
24         void reset_TUN_storage();
25         void process_local_TUN_packet();
26         void process_external_TUN_packet();
27
28     // general functions for takeoff
29     public:
30         void takeoff_service_latch();
31
32     // Constructor
33     public:
34         Takeoff_service(Xapi& _xapi, LCD_service& _lcd);
35 };
36
```

37 **#endif**

./commCode/Takeoff_service.h

```
1
2 #ifndef TAKEOFF_SERVICE_cpp
3 #define TAKEOFF_SERVICE_cpp
4 #include <AutoPilot.h>
5 #include <Takeoff_service.h>
6
7 extern uint8_t bit_autopilot_flags;
8 extern drone_state *P_state;
9 //*****
10 // This latch is what is called in the microcontroller's
11 // main loop. Put any required processing here
12 //*****
13 void Takeoff_service::takeoff_service_latch()
14 {
15
16     // process any local LCD message packets
17     process_local_TUN_packet();
18
19     // process any external LCD message packets
20     process_external_TUN_packet();
21 }
22
23
24
25 //*****
26 //*****
27 // This routine will query the XAPI to see
28 // if there is a local message waiting for
29 // the Land service. If so, we need to grab it and react.
30 void Takeoff_service::process_external_TUN_packet()
31 {
32     // see if there is a packet waiting
33     if(m_xapi.CONNECT_external_TUN_get_type() ==
        TUN_TYPE_EXTERNALTAKEOFF)
34     {
35         // allocate the space
36         uint8_t TUN_packet[MED_BUFF_SZ];
37         uint8_t payload_buff[SMALL_BUFF_SZ];
38         uint8_t payload_buff_sz = 0;
39         int height = 0;
```



```

40 // extract the packet
41 m_xapi.CONNECT_external_TUN_get_packet(TUN_packet, MED_BUFF_SZ);
42
43 //extract payload
44 payload_buff_sz = m_util.get_TUN_payload(TUN_packet, payload_buff,
      SMALL_BUFF_SZ);
45
46 //grab height from payload (4 bytes)
47 height = m_util.hex_to_int(0, 4, payload_buff_sz, payload_buff);
48
49 // do something
50 //lcd prints are for debugging, should be removed
51 //m_lcd.lcd_print(0,0,"*****");
52 //m_lcd.lcd_print(0,0,"Got Takeoff");
53 //m_lcd.lcd_print(0,0,"ttest1");
54 //m_lcd.lcd_print(0,0,"takeoff");
55 //m_lcd.lcd_print(0,0,"ttest2");
56 //if(height == 10){
57 //    m_lcd.lcd_print(0,0,"Height 10");
58 //}
59 P_state->hold_alt = P_state->ground_alt + height;
60 bit_autopilot_flags |= ALTHOLD_FLAG;
61 }
62 }
63
64 //*****
65 //*****
66 // This routine will query the XAPI to see
67 // if there is a local message waiting for
68 // the Land service. If so, we need to grab it and do something.
69 void Takeoff_service::process_local_TUN_packet()
70 {
71 // see if there is a packet waiting
72 if(m_xapi.CONNECT_local_TUN_get_type() == TUN_TYPE_LOCALTAKEOFF)
73 {
74 // allocate the space
75 uint8_t TUN_packet[MED_BUFF_SZ];
76
77 // extract the packet
78 m_xapi.CONNECT_local_TUN_get_packet(TUN_packet, MED_BUFF_SZ);
79
80 // do something

```

```

81
82     }
83 }
84
85 //*****
86 //*****
87 // Resets the TUN packet storage
88 void Takeoff_service::reset_TUN_storage()
89 {
90     // obsolete
91 }
92
93 //*****
94 //*****
95 Takeoff_service::Takeoff_service(Xapi& _xapi, LCD_service& _lcd):
96 m_xapi(_xapi), m_lcd(_lcd)
97 {
98     reset_TUN_storage();
99 }
100 }
101
102
103 #endif

```

./commCode/Takeoff_service.cpp

5.3.7 Serial Service (Modified from Brandon's Version)

```

1
2 #ifndef SERIAL_SERVICE_h
3 #define SERIAL_SERVICE_h
4
5 #include <arduino.h>
6 #include <single_buff.h>
7 #include <util.h>
8 #include <xapi.h>
9 #include <LCD_service.h>
10
11 //*****
12 //*****
13 // The Serial Service allows for the XAPI to
14 // communicate over a standard serial connection
15 // to the outside world. This service only
16 // communicates via the standard TUN packet.

```

```

17 // TUN packet format:
18 // Format of the TUNNELED (TUN) packet in ASCII-HEX:
19 // $[TYPE:2][PAYLOAD_SZ:2][CHECKSUM:4][PAYLOAD:~]%
20
21
22 class Serial_service
23 {
24     // The objects this service requires
25     private:
26         Util m_util;
27         HardwareSerial& m_serial;
28         Xapi& m_xapi;
29         LCD_service& m_lcd;
30
31     // This buffer and variables keep track of
32     // incoming RX bytes.
33     private:
34         uint8_t m_rx_buff[LARGE_BUFF_SZ];
35         boolean m_rx_start_found;
36         boolean m_rx_end_found;
37         boolean m_rx_buff_ready;
38         uint8_t m_rx_buff_index;
39
40     // routines to maintain state
41     private:
42         uint8_t assemble_TUN_packet(uint8_t c);
43         void reset_rx_state();
44         boolean process_TUN_packet( uint8_t* buff, uint8_t buff_sz);
45         void simple_local_LCD_msg( uint8_t* buff, uint8_t buff_sz);
46         void snd_serial_add_frame(const uint8_t* buff, uint8_t buff_sz);
47         void snd_local_TUN_packet_via_serial();
48         void create_and_pass_external(uint8_t packet_type, uint8_t* buff,
49                                     uint8_t buff_sz);
50
51     // constructor and latch
52     public:
53         Serial_service(HardwareSerial& _serial, Xapi& _xapi, LCD_service&
54                        _lcd);
55         void serial_service_latch();
56 };

```

```

57 #endif

                                   ./commCode/Serial_service.h

1
2 #ifndef SERIAL_SERVICE.cpp
3 #define SERIAL_SERVICE.cpp
4
5 #include <serial_service.h>
6
7 //*****
8 //*****
9 // Simply sends a buffer over serial.
10 // Will add a frame to the buffer
11 void Serial_service::snd_serial_add_frame(const uint8_t* buff, uint8_t
    buff_sz)
12 {
13     m_serial.print(SENT_START_BYTE);
14
15     m_serial.write(buff, buff_sz);
16
17     m_serial.print(SENT_END_BYTE);
18
19     m_serial.flush();
20 }
21
22 //*****
23 //*****
24 // Process any local serial messages
25 void Serial_service::snd_local_TUN_packet_via_serial()
26 {
27     uint8_t packet_sz = 0;
28
29     // storage for the debug packet
30     uint8_t serial_packet[LARGE_BUFF_SZ];
31
32     // clean packet
33     m_util.clean_packet(serial_packet, LARGE_BUFF_SZ);
34
35     // extract the packet
36     m_xapi.CONNECT_local_TUN_get_packet(serial_packet, LARGE_BUFF_SZ);
37
38     // get the size of the packet
39     packet_sz = m_util.get_TUN_packet_sz(serial_packet);

```

```

40
41 // ship out the packet through serial
42 snd_serial_add_frame(serial_packet , packet_sz);
43 }
44
45 //*****
46 //*****
47 // Latch to sample the serial hardware
48 void Serial_service::serial_service_latch()
49 {
50     uint8_t packet_type = 0;
51
52     // see if there is a new byte
53     if (m_serial.available() > 0)
54     {
55         assemble_TUN_packet(m_serial.read());
56     }
57
58     // process any local serial packets that need to
59     // be shipped out over serial.
60     // NOTE: the point of this code is to allow other
61     // services to ship out packets via serial instead
62     // of radio.
63     packet_type = m_xapi.CONNECT_local_TUN_get_type();
64     switch(packet_type)
65     {
66         case TUN_TYPE_LOCAL_SERIAL_DEBUG_MSG:
67         case TUN_TYPE_LOCAL_CHAT:
68             snd_local_TUN_packet_via_serial();
69             break;
70     }
71 }
72
73 //*****
74 //*****
75 // This routine allows for the serial service to display
76 // a simple local LCD message.
77 // The payload is used to create an entirely new LCD packet.
78 // The new derived packet is of type TUN_TYPE_LOCAL_LCD_MSG
79 // packet format:
80 // [TYPE:2][PAYLOAD_SZ:2][CHECKSUM:4][PAYLOAD:?]
81 void Serial_service::simple_local_LCD_msg( uint8_t* buff,

```

```

82             uint8_t buff_sz)
83 {
84     // storage for payload
85     uint8_t payload[MED_BUFF_SZ];
86
87     // storage for new packet
88     uint8_t new_TUN_packet[LARGE_BUFF_SZ];
89     uint8_t new_TUN_packet_sz = 0;
90
91     // extract payload
92     m_util.get_TUN_payload(buff, payload, MED_BUFF_SZ);
93
94     // create the TUN packet
95     new_TUN_packet_sz = m_util.create_TUN_lcd_packet( true, 0, 0,
96             payload,
97             strlen((char*)payload),
98             new_TUN_packet,
99             LARGE_BUFF_SZ);
100
101     // send the new TUN packet out locally
102     m_xapi.CONNECT_local_TUN_set_packet(new_TUN_packet,
103             new_TUN_packet_sz);
104 }
105
106 //*****
107 //*****
108 // This routine processes a completely assembled RX TUN packet.
109 // Returns:
110 // true: packet was processed
111 // false: packet was not processed due to CHECKSUM error
112 boolean Serial_service::process_TUN_packet( uint8_t* buff,
113             uint8_t buff_sz)
114 {
115     boolean success = false;
116     uint8_t TUN_type = ILLEGAL_TUN_TYPE;
117
118     // only process the buffer if passes CHECKSUM
119     if(m_util.verify_checksum(buff))
120     {
121         TUN_type = m_util.get_TUN_type(buff);
122
123         switch(TUN_type)

```

```

122 {
123     // an incoming request to use the local
124     // LCD screen to display a message.
125     case TUN_TYPE_LOCAL_LCD_MSG:
126         m_xapi.CONNECT_local_TUN_set_packet(buff, buff_sz);
127     break;
128
129     // for doing a simple local LCD debug message
130     case TUN_TYPE_LOCAL_SIMPLE_LCD_MSG:
131         simple_local_LCD_msg(buff, buff_sz);
132     break;
133
134     case TUN_TYPE_EXTERNAL_LCD_MSG:
135         create_and_pass_external(TUN_TYPE_EXTERNAL_LCD_MSG, buff,
136                                 buff_sz);
137         //m_lcd lcd_print("*****");
138         m_lcd lcd_print(0,0,"Got LCD");
139     break;
140
141     case TUN_TYPE_EXTERNAL_LAND:
142         create_and_pass_external(TUN_TYPE_EXTERNAL_LAND, buff, buff_sz);
143         //m_lcd lcd_print("*****");
144         m_lcd lcd_print(0,0,"Got Land");
145     break;
146
147     case TUN_TYPE_EXTERNAL_TAKEOFF:
148         create_and_pass_external(TUN_TYPE_EXTERNAL_TAKEOFF, buff,
149                                 buff_sz);
150         //m_lcd lcd_print("*****");
151         m_lcd lcd_print(0,0,"Got Takeoff");
152     break;
153
154     case TUN_TYPE_EXTERNAL_DO_MOVE:
155         create_and_pass_external(TUN_TYPE_EXTERNAL_DO_MOVE, buff,
156                                 buff_sz);
157         //m_lcd lcd_print("*****");
158         m_lcd lcd_print(0,0,"Got Do Move");
159     break;
160
161     case TUN_TYPE_EXTERNAL_SET_HEADING:
162         create_and_pass_external(TUN_TYPE_EXTERNAL_SET_HEADING, buff,
163                                 buff_sz);
164         //m_lcd lcd_print("*****");
165         m_lcd lcd_print(0,0,"Got Set HEading");
166     break;
167
168     case TUN_TYPE_EXTERNAL_ALT_HOLD:

```

```

160         create_and_pass_external(TUN_TYPE_EXTERNAL_ALT_HOLD, buff,
            buff_sz);
161         //m_lcd lcd_print("*****");
162         m_lcd lcd_print(0,0,"Got Alt Hold");
163         break;
164         case TUN_TYPE_EXTERNAL_HEADING_HOLD:
165             create_and_pass_external(TUN_TYPE_EXTERNAL_HEADING_HOLD, buff,
                buff_sz);
166             //m_lcd lcd_print("*****");
167             m_lcd lcd_print(0,0,"Got head Hold");
168             break;
169         case TUN_TYPE_EXTERNAL_ARM:
170             create_and_pass_external(TUN_TYPE_EXTERNAL_ARM, buff, buff_sz);
171             //m_lcd lcd_print("*****");
172             m_lcd lcd_print(0,0,"Got Arm");
173             break;
174     }
175
176     success = true;
177 }
178 return success;
179 }
180
181 void Serial_service::create_and_pass_external(uint8_t packet_type,
182                                             uint8_t* buff,
183                                             uint8_t buff_sz)
184 {
185     /* Router info
186         MSB: 13A200
187         LSB: 40A8BC2C
188     */
189     //uint32_t msb = 0x13A200;
190     //uint16_t lsb = 0x40A8BC2C;
191     uint32_t msb = 0x0;
192     uint16_t lsb = 0xffff;
193
194
195     uint8_t x = 0;
196     uint8_t y = 0;
197     uint8_t LCD_payload_sz = 0;
198     uint8_t packet_size = 0;
199     // storage for payload

```



```

200     uint8_t payload[MED_BUFF_SZ];
201     uint8_t message[MED_BUFF_SZ];
202     // storage for new packet
203     //uint8_t new_TUN_packet[LARGE_BUFF_SZ];
204     //uint8_t new_TUN_packet_sz = 0;
205
206     uint8_t payload_buff_sz = 0;
207
208     uint8_t payload_buff[LARGE_BUFF_SZ];
209     uint8_t packet_buff[LARGE_BUFF_SZ];
210
211     uint8_t TUN_buff[LARGE_BUFF_SZ];
212     uint8_t TUN_buff_sz = 0;
213     /*
214     // extract payload, should contain x,y,message
215     LCD_payload_sz = m_util.get_TUN_payload(buff, payload,
        MED_BUFF_SZ);
216
217     // get the X and Y coordinates
218     x = m_util.hex_to_int(LCD_X_START, LCD_X_END, LCD_X_SZ,
        LCD_payload);
219     y = m_util.hex_to_int(LCD_Y_START, LCD_Y_END, LCD_Y_SZ,
        LCD_payload);
220
221     // figure out the msg size
222     LCD_payload_sz -= (LCD_X_SZ + LCD_Y_SZ);
223
224     // display the string
225     for(uint8_t i = 0; i < LCD_payload_sz; i++)
226         message[i] = LCD_payload[i+LCD_MSG_START];
227     */
228     //m_lcd lcd_print(0,0,"first");
229     payload_buff_sz = m_util.get_TUN_payload(buff, payload_buff,
        MED_BUFF_SZ);
230     //m_lcd lcd_print(0,0,"second");
231
232     TUN_buff_sz = m_util.create_TUN_packet( packet_type,
233         payload_buff,
234         payload_buff_sz,
235         TUN_buff,
236         LARGE_BUFF_SZ);
237     //construct and send packet

```

```

238         //m_lcd.lcd_print(0,0,"third");
239
240
241         //TUN_buff[0] = 'X';
242         //TUN_buff[1] = 'X';
243         //TUN_buff[2] = 'X';
244         //TUN_buff_sz = 3;
245
246         packet_size = m_xapi.construct_transmit_req(msb,
247                                                     lsb,
248                                                     ADDR16_BROADCAST,
249                                                     TUN_buff,
250                                                     TUN_buff_sz,
251                                                     packet_buff,
252                                                     LARGE_BUFF_SZ);
253         //m_lcd.lcd_print(0,0,"fourth");
254
255
256         m_xapi.snd_packet(packet_buff, packet_size);
257
258         //uint8_t* TUN_buff[3];
259         //TUN_buff[0] = 'X';
260         //TUN_buff[1] = 'X';
261         //TUN_buff[2] = 'X';
262         //lcd.snd_EXTERNAL_message( msb,
263                                   //      lsb,
264                                   //      ADDR16_BROADCAST,
265                                   //      0,
266                                   //      0,
267                                   //      (uint8_t*)"XXXX");
268
269         //m_lcd.lcd_print(0,0,"final");
270         //m_xapi.display_TUN_packet(TUN_buff, TUN_buff_sz);
271
272     }
273
274     //*****
275     //*****
276     // This routine resets the entire RX state and
277     // makes the service ready to RX a new TUN packet.
278     void Serial_service::reset_rx_state()
279     {

```

```

280  m_util.clean_packet(m_rx_buff, LARGE_BUFF_SZ);
281  m_rx_start_found = false;
282  m_rx_end_found = false;
283  m_rx_buff_ready = false;
284  m_rx_buff_index = 0;
285 }
286
287 //*****
288 //*****
289 // This routine takes in RX bytes and assembles a TUN packet.
290 // Format of the TUNNELED (TUN) packet in ASCII-HEX:
291 // $[TYPE:2][PAYLOAD_SZ:2][CHECKSUM:4][PAYLOAD:?]%
292 uint8_t Serial_service::assemble_TUN_packet(uint8_t c)
293 {
294     uint8_t rx_byte = c;
295
296     // first see if a byte is waiting
297
298     // only add to buffer is byte is read
299     if(c != '\0')
300     {
301         // see if byte is start byte
302         if(rx_byte == SENT_START_BYTE)
303         {
304             // record that it's the start of a new packet
305             reset_rx_state();
306             m_rx_start_found = true;
307         }
308         else if( rx_byte == SENT_END_BYTE)
309         {
310             // ensure we have read start byte
311             // and there is more than 0 characters in buffer
312             if( (m_rx_start_found == true) && (m_rx_buff_index > 0))
313             {
314                 // we have a complete packet
315                 m_rx_end_found = true;
316                 m_rx_buff_ready = true;
317
318                 // process the packet
319                 process_TUN_packet(m_rx_buff, m_rx_buff_index);
320
321                 // reset the buffer

```

```

322         reset_rx_state();
323     }
324 }
325 else if( (m_rx_start_found == true) &&
326         (m_rx_end_found == false) &&
327         (m_rx_buff_index < (LARGE_BUFF_SZ - 1)) )
328     {
329         // we have a single character that is
330         // not a stop or a start
331         // So just store it
332         m_rx_buff[m_rx_buff_index++] = rx_byte;
333     }
334 }
335
336 // return the byte we read
337 return rx_byte;
338 }
339
340 //*****
341 //*****
342 Serial_service::Serial_service(HardwareSerial& _serial, Xapi& _xapi,
343                               LCD_service& _lcd):
344     m_serial(_serial), m_xapi(_xapi), m_lcd(_lcd)
345 {
346     m_util = Util();
347     reset_rx_state();
348 }
349 #endif

```

./commCode/Serial_service.cpp

6 Design Presentation

6.1 Nov 13, 2014

See Next Page

Drone Mission Planning Software

David Klingenberg
Taylor Trabun

- Problem Definition
- Specific Component Design
 - Communications Design
 - Drone Design
 - Graphical User Interface Design
- Timeline
- Questions/Concerns

Overview

The goal of this project is to design and develop a graphical user interface (GUI) for drone mission planning.

Requirements:

- A user-friendly interface
- Allow 3-dimensional mission planning
- Upload the flight plan using XAPI and XBee
- Allow manual override
- Implement drone hardware for flight control

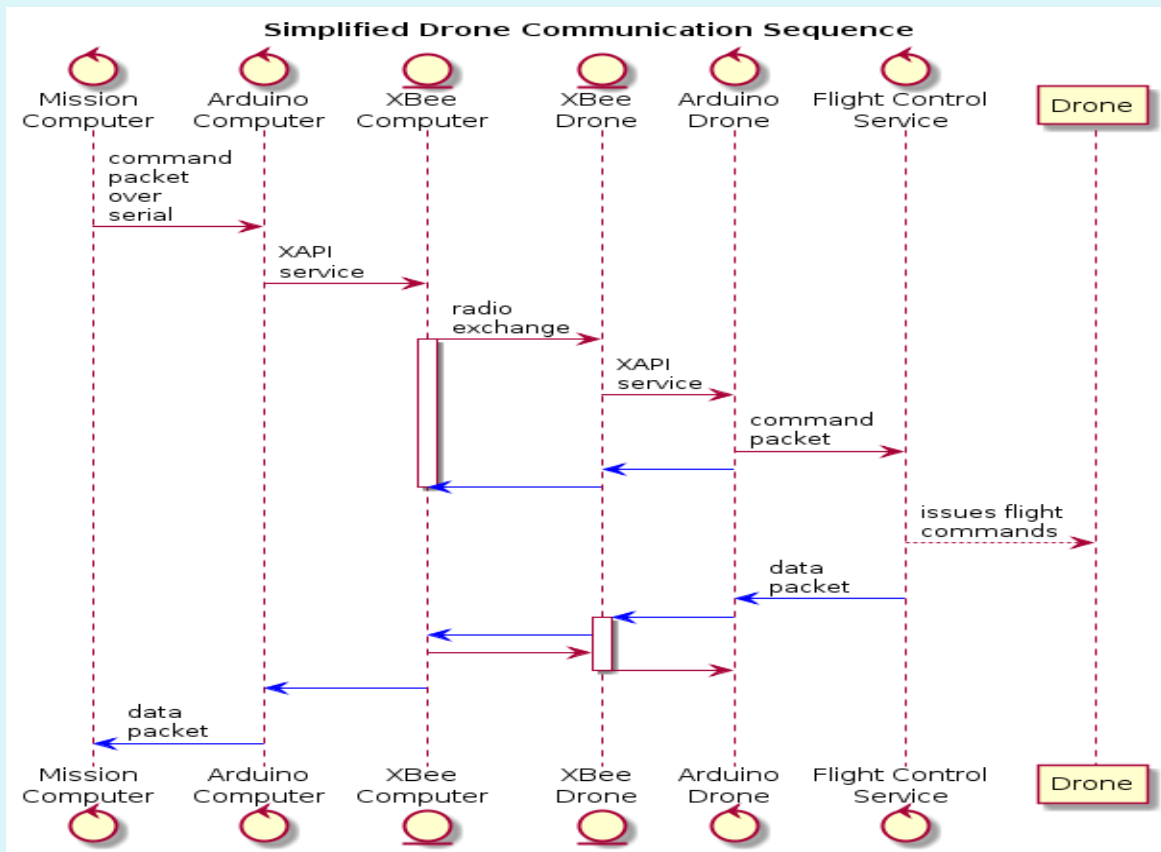
Problem Definition

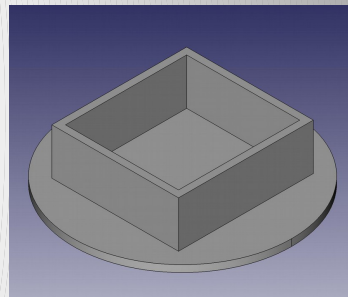
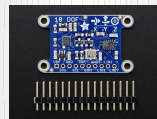
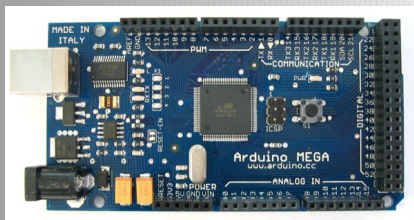
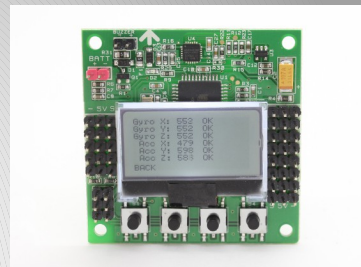
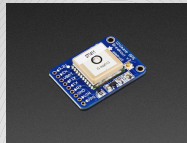
The communication system for this project must allow for commands to be sent from a computer to a drone.

Requirements:

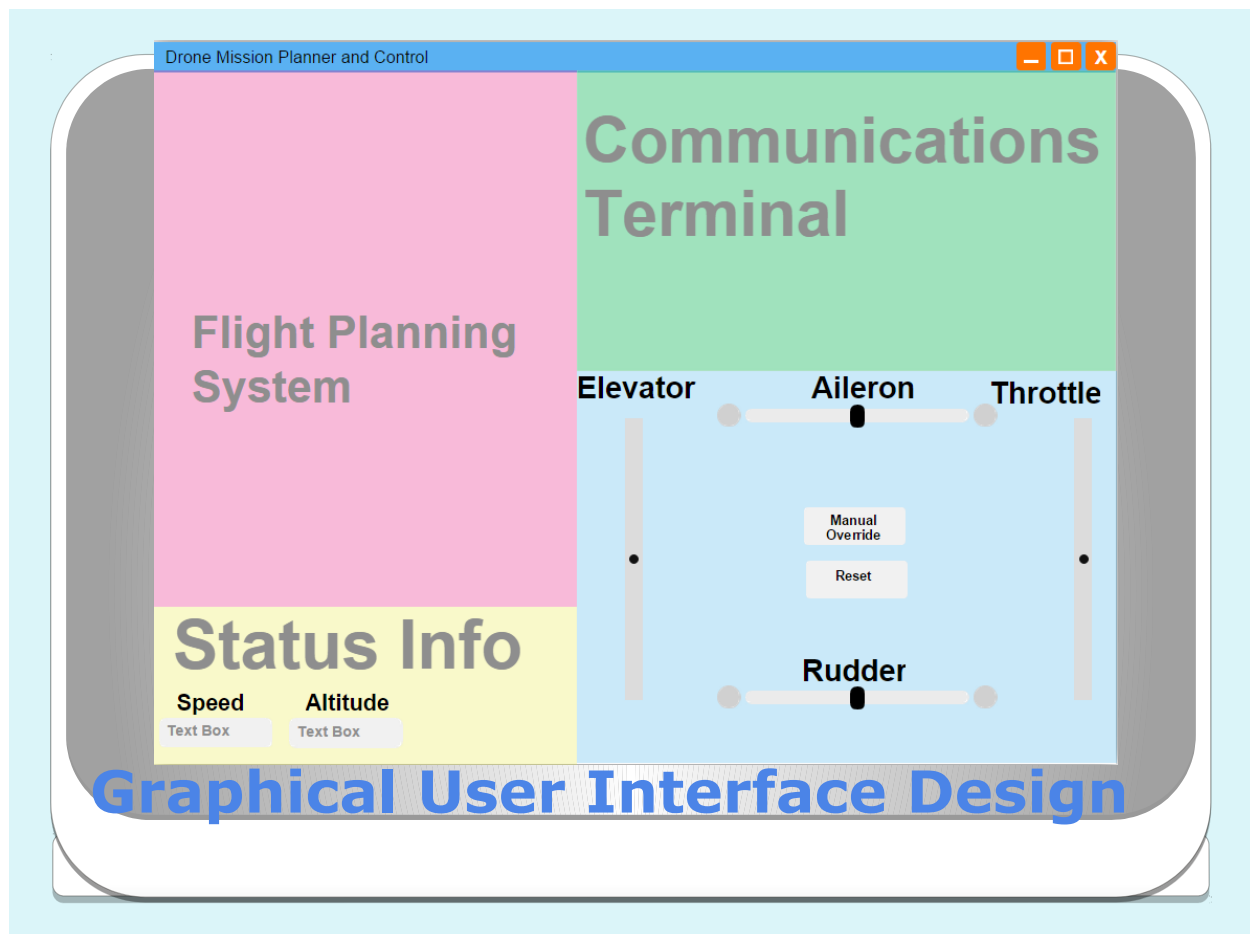
- XAPI and XBee hardware
- Specific TUN packets:
 - Manual drone instructions (altitude, direction, takeoff, etc..)
 - Settings
 - Acknowledgement of packet received
 - Heartbeat/status updates
 - Override (manual, land)
 - Flight plan protocol
 - ▮ Initialize for upload
 - ▮ Get instructions
 - ▮ Echo instructions
 - ▮ Terminate upload

Communication Design





Drone Design



Re-design drone for reliable flight

Finish implementing communications between two Arduino boards over XBee using XAPI

Finish designing all needed packet types for drone control, status, ACK, and response.

Implement service on Arduino to send commands to flight computer

Finalize UI design, implement communications system, design mission planning sub-system

Timeline

Questions or concerns?

Thank You

Drone Mission Planning Software: Design Document

Taylor Trabun

May 6, 2015

7 Design Document

7.1 Introduction

This document provides the general design of the Drone Mission Planning Software by breaking the entire project down into several components. The current components are the physical drone, the communication system, and the graphical user interface for mission planning.

7.2 Drone Design

The drone design's major requirement that needed to be met was to achieve stable and reliable flight.

By analyzing the drone, it was determined that its center of gravity was not directly centered on the drone, which resulted in drifting during flight. To remedy this issue, a "part holder" was designed to be mounted on the underside of the drone to hold all the motor controls, which moved the center of gravity to the center of the drone.

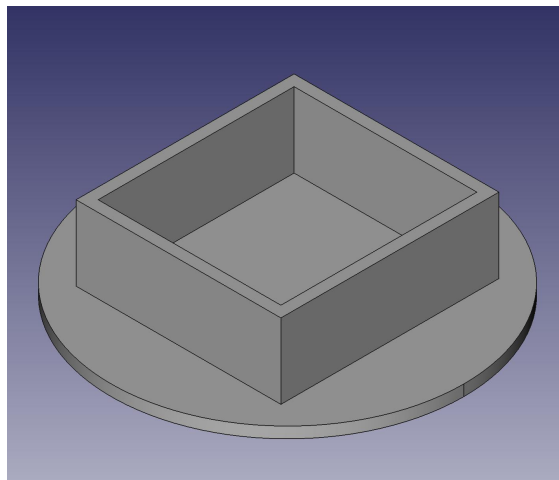


Figure 1: 3D sketch of partbin

7.3 Communication Design

The communications system needs to follow the following requirements:

- Use of XAPI and XBee hardware
- Define required TUN packets for communication system
 - Manual drone instructions
 - Settings and status
 - ACK
 - Heartbeat
 - Override
 - Flight plan protocol types

The following sections will break the communication system down into its several components and detail their design.

7.3.1 Communication Overview

Our communications system, as depicted in the graphic below, requires two-way communication between the computer (including the attached Arduino) and the Arduino located on the drone. This system will take an instruction created on the computer, send it over serial to the connected Arduino, pass it to XAPI, XAPI will ship it over XBee to the Arduino on the drone, and a flight control service will execute the instruction.

7.3.2 Hardware Components

Our current communications requires the following hardware components:

- Arduino Mega 2560
- XBee modules
- LCD Shields (for development and debugging)
- Serial add-on for Arduino
- A computer running Windows to communicate with base Arduino (Workstation needs to be able to run C# programs)

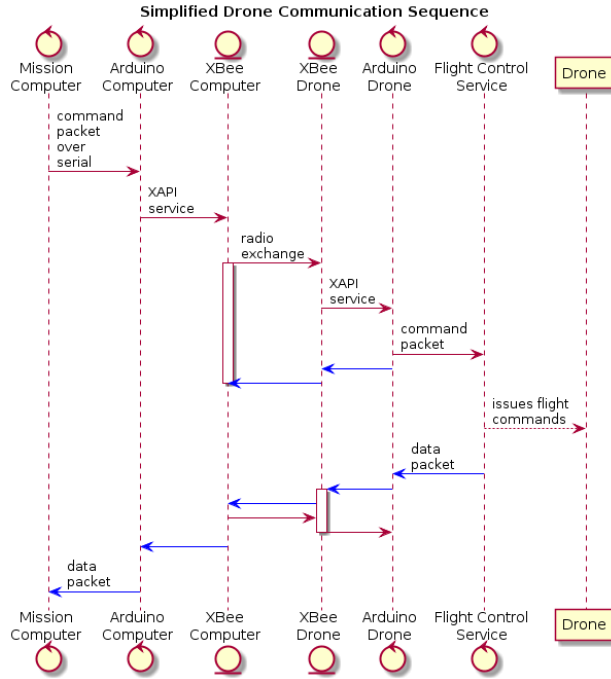


Figure 2: Overview of communications system

7.3.3 XAPI

To satisfy one of our major requirements, we run the XAPI on each Arduino in the communications system.

XAPI is, put simply, a micro-controller service manager that communicates, both internally and externally, using TUN packets. When communicating externally, the XAPI ships the TUN packet using the XBee hardware by embedding the TUN packet in a XBee packet.

Each service available with the API use XAPI to communicate, using XAPI as the core that each service "latches" to. For instance, if a chat service wanted to display a message on a attached LCD screen, the following steps would be carried out:

1. Chat service creates a LOCAL_ LCD TUN packet
2. Chat service passes the newly created packet to the XAPI core
3. XAPI places the packet in its internal packet buffer
4. The LCD service latch queries XAPI for any packets designated for the LCD service
5. LCD service grabs LOCAL_ LCD TUN packet
6. LCD service processes and displays the packet

To satisfy our project's requirements, we need to design a Flight Control service that will be able to handle any instruction packets and translate them to instructions that can be given to the drone's flight computer. In addition to this, there is a requirement for a Mission Plan service that will store and execute flight plan's designed by the user and sent to the drone.

7.4 Graphical User Interface Design

The graphical user interface must satisfy several requirements:

1. Must be user-friendly
2. Allow 3-dimensional mission planning
3. Allow upload of flight plan to drone
4. Allow manual override

Given these requirements we were able to design a general design for the graphical user interface (GUI), as shown below. This GUI is split into several sections that convey different information, that in some cases can be adjusted.

We have a flight control section (light-blue) that shows the status of the drone components and allows the user to "zero out" each component or take manual control. The status information section (yellow) shows different readings from the drone's on-board instruments. The flight planning system (light-red) will be where the user can develop a flight plan to be uploaded to the drone (note that this functionality is still under design and may end up being a separate window that needs to be opened up). The final section is the communications terminal (light-green) that displays all packets sent and received on the Arduino attached to the source computer. This communications terminal will allow the user to see that the drone is still connected and will allow for easy communications debugging.

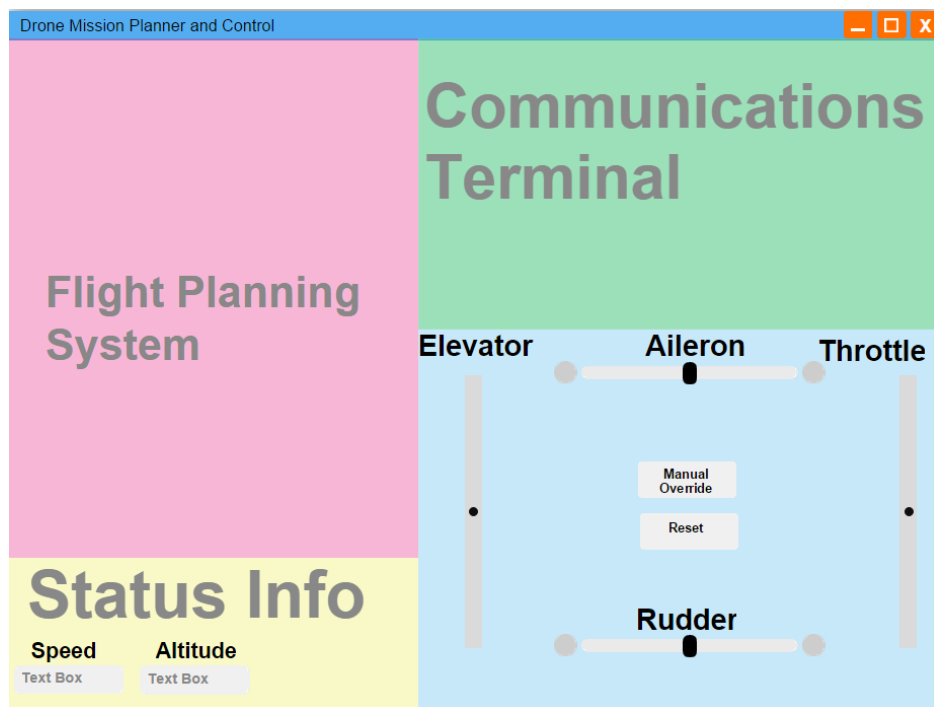


Figure 3: Graphical user interface mock-up design

Appendices

A Miscellaneous UML Charts

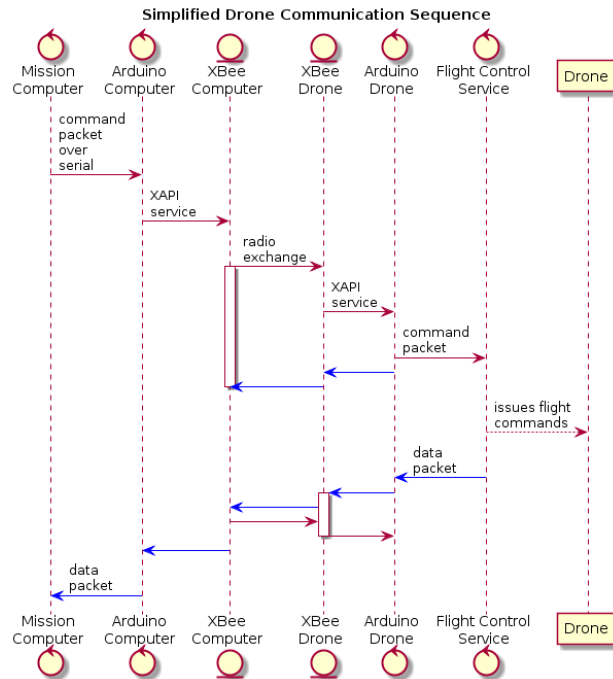


Figure 4: Communication Sequence

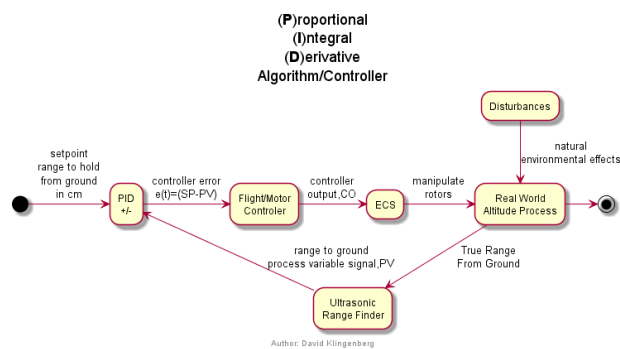


Figure 5: PID Controller

B ATMEL[®] Microcontrollers

Features

- High-performance, Low-power Atmel[®] AVR[®] 8-bit Microcontroller
- Advanced RISC Architecture
 - 131 Powerful Instructions – Most Single-clock Cycle Execution
 - 32 x 8 General Purpose Working Registers
 - Fully Static Operation
 - Up to 20 MIPS Throughput at 20MHz
- High Endurance Non-volatile Memory segments
 - 64 Kbytes of In-System Self-programmable Flash program memory
 - 2 Kbytes EEPROM
 - 4 Kbytes Internal SRAM
 - Write/Erase cycles: 10,000 Flash/100,000 EEPROM⁽¹⁾⁽³⁾
 - Data retention: 20 years at 85°C/100 years at 25°C⁽²⁾⁽³⁾
 - Optional Boot Code Section with Independent Lock Bits
 - In-System Programming by On-chip Boot Program
 - True Read-While-Write Operation
 - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
 - Boundary-scan Capabilities According to the JTAG Standard
 - Extensive On-chip Debug Support
 - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
 - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
 - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
 - Real Time Counter with Separate Oscillator
 - Six PWM Channels
 - 8-channel, 10-bit ADC
 - Differential mode with selectable gain at 1x, 10x or 200x
 - Byte-oriented Two-wire Serial Interface
 - One Programmable Serial USART
 - Master/Slave SPI Serial Interface
 - Programmable Watchdog Timer with Separate On-chip Oscillator
 - On-chip Analog Comparator
 - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
 - Power-on Reset and Programmable Brown-out Detection
 - Internal Calibrated RC Oscillator
 - External and Internal Interrupt Sources
 - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
 - 32 Programmable I/O Lines
 - 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF
- Speed Grades
 - ATmega644V: 0 - 4MHz @ 1.8V - 5.5V, 0 - 10MHz @ 2.7V - 5.5V
 - ATmega644: 0 - 10MHz @ 2.7V - 5.5V, 0 - 20MHz @ 4.5V - 5.5V
- Power Consumption at 1MHz, 3V, 25°C
 - Active: 240µA @ 1.8V, 1MHz
 - Power-down Mode: 0.1µA @ 1.8V

Notes: 1. Worst case temperature. Guaranteed after last write cycle.
2. Failure rate less than 1 ppm.
3. Characterized through accelerated tests.



**8-bit Atmel
Microcontroller
with 64K Bytes
In-System
Programmable
Flash**

ATmega644/V

2593O-AVR-02/12



Figure 6: ATmega644



Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V

8-bit Atmel Microcontroller with 16/32/64KB In-System Programmable Flash

DATASHEET

Features

- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
 - 135 Powerful Instructions – Most Single Clock Cycle Execution
 - 32 x 8 General Purpose Working Registers
 - Fully Static Operation
 - Up to 16 MIPS Throughput at 16MHz
 - On-Chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
 - 64K/128K/256KBytes of In-System Self-Programmable Flash
 - 4Kbytes EEPROM
 - 8Kbytes Internal SRAM
 - Write/Erase Cycles:10,000 Flash/100,000 EEPROM
 - Data retention: 20 years at 85°C/ 100 years at 25°C
 - Optional Boot Code Section with Independent Lock Bits
 - In-System Programming by On-chip Boot Program
 - True Read-While-Write Operation
 - Programming Lock for Software Security
- Atmel® QTouch® library support
 - Capacitive touch buttons, sliders and wheels
 - QTouch and QMatrix acquisition
 - Up to 64 sense channels
- JTAG (IEEE® std. 1149.1 compliant) Interface
 - Boundary-scan Capabilities According to the JTAG Standard
 - Extensive On-chip Debug Support
 - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
 - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
 - Four 16-bit Timer/Counter with Separate Prescaler, Compare- and Capture Mode
 - Real Time Counter with Separate Oscillator
 - Four 8-bit PWM Channels
 - Six/Twelve PWM Channels with Programmable Resolution from 2 to 16 Bits (ATmega1281/2561, ATmega640/1280/2560)
 - Output Compare Modulator
 - 8/16-channel, 10-bit ADC (ATmega1281/2561, ATmega640/1280/2560)
 - Two/Four Programmable Serial USART (ATmega1281/2561, ATmega640/1280/2560)
 - Master/Slave SPI Serial Interface
 - Byte Oriented 2-wire Serial Interface
 - Programmable Watchdog Timer with Separate On-chip Oscillator
 - On-chip Analog Comparator
 - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
 - Power-on Reset and Programmable Brown-out Detection
 - Internal Calibrated Oscillator
 - External and Internal Interrupt Sources
 - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
 - 54/86 Programmable I/O Lines (ATmega1281/2561, ATmega640/1280/2560)
 - 64-pad QFN/MLF, 64-lead TQFP (ATmega1281/2561)
 - 100-lead TQFP, 100-ball CBGA (ATmega640/1280/2560)
 - RoHS/Fully Green
- Temperature Range:
 - -40°C to 85°C Industrial
- Ultra-Low Power Consumption
 - Active Mode: 1MHz, 1.8V: 500µA
 - Power-down Mode: 0.1µA at 1.8V
- Speed Grade:
 - ATmega640V/ATmega1280V/ATmega1281V:
 - 0 - 4MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
 - ATmega2560V/ATmega2561V:
 - 0 - 2MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
 - ATmega640/ATmega1280/ATmega1281:
 - 0 - 8MHz @ 2.7V - 5.5V, 0 - 16MHz @ 4.5V - 5.5V
 - ATmega2560/ATmega2561:
 - 0 - 16MHz @ 4.5V - 5.5V

2549Q-AVR-02/2014

Figure 7: ATmega2560

C Technical Drawings

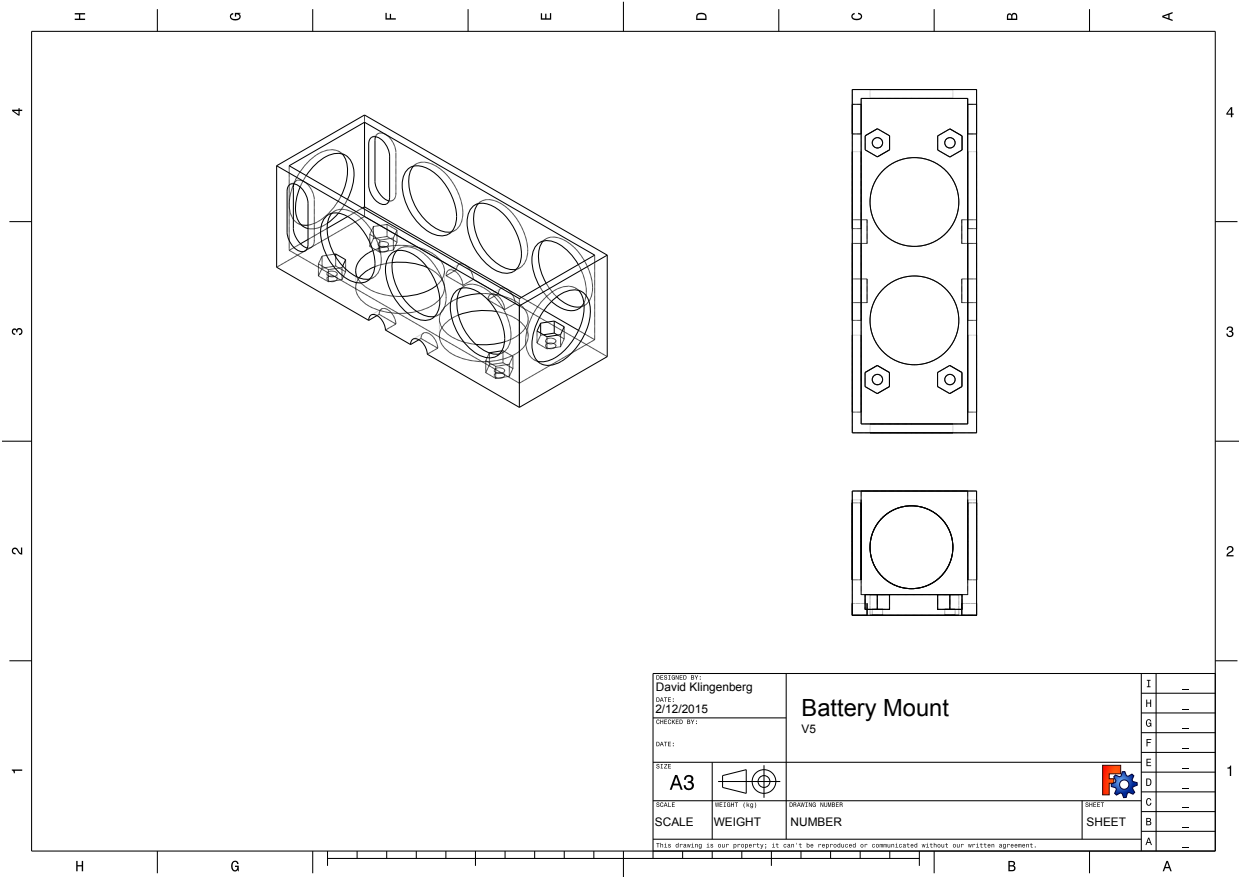


Figure 8: Battery Bracket

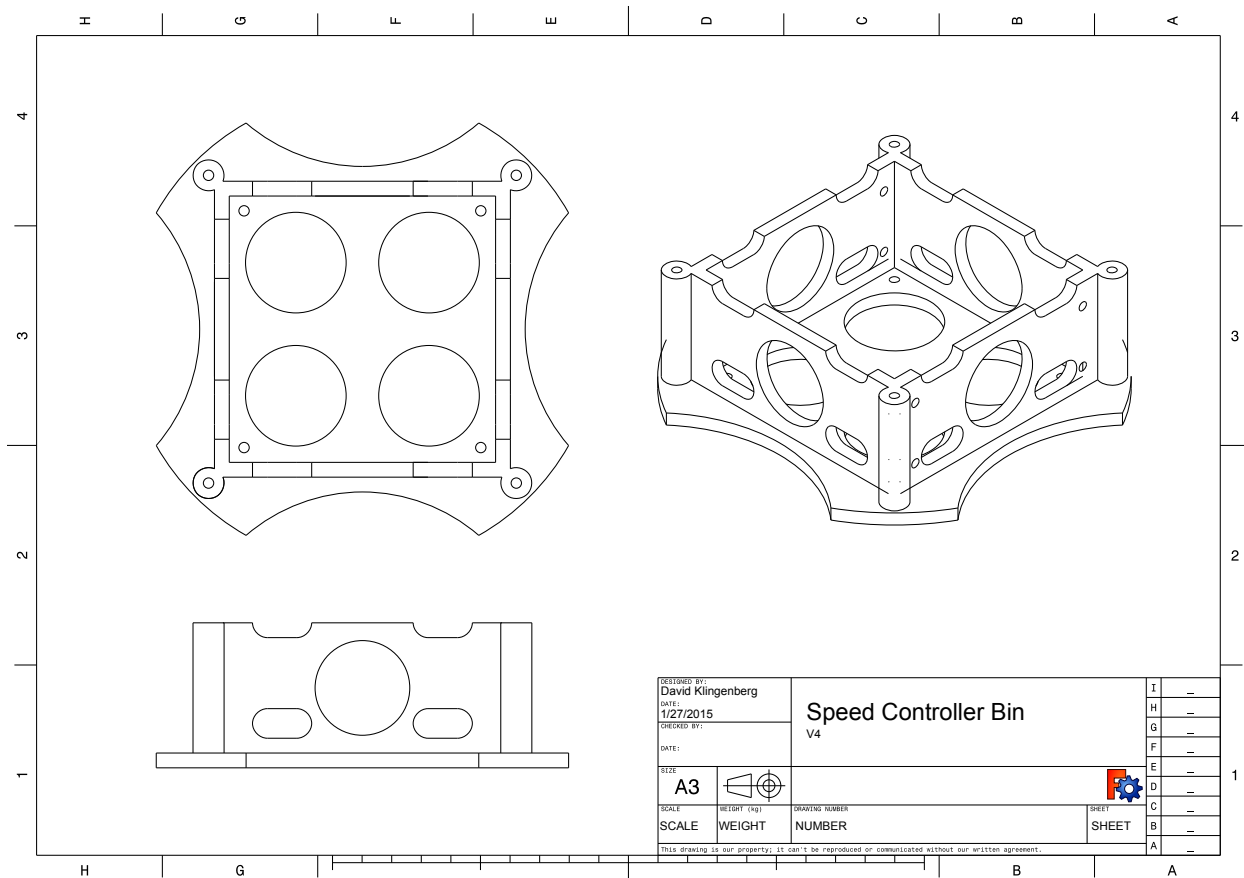


Figure 9: Electronic Speed Controller Bracket

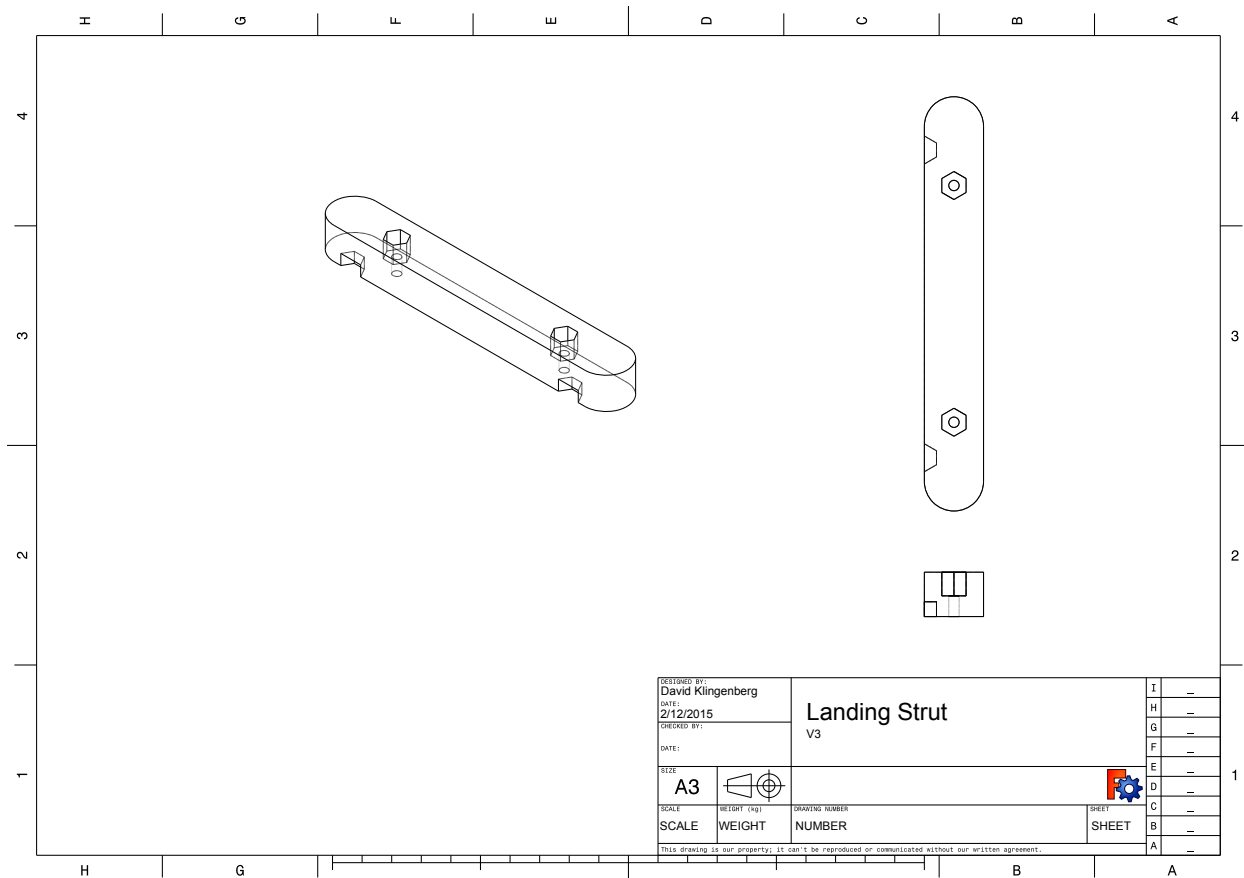


Figure 10: Landing Strut

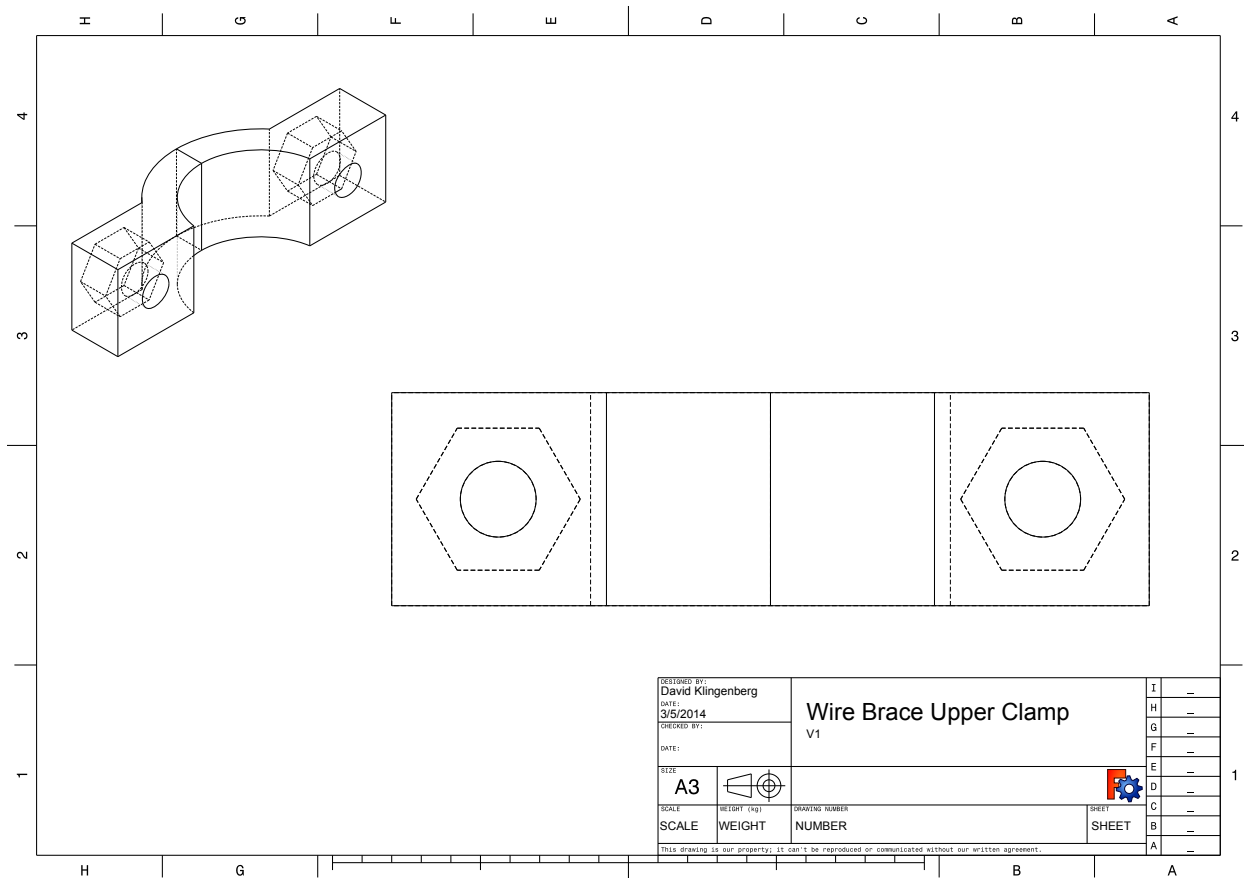


Figure 11: Wire Brace Upper Clamp

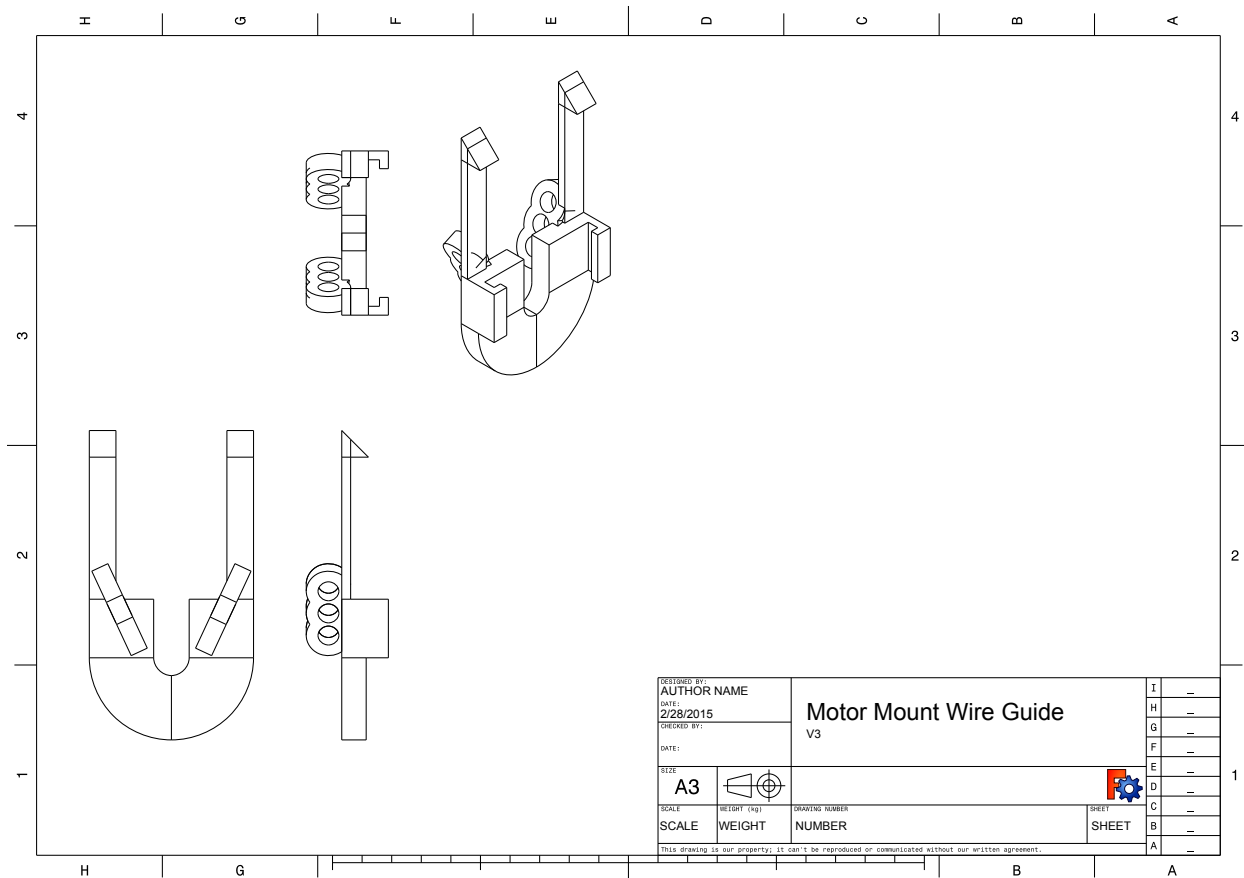


Figure 12: Motor Mount Wire Brace

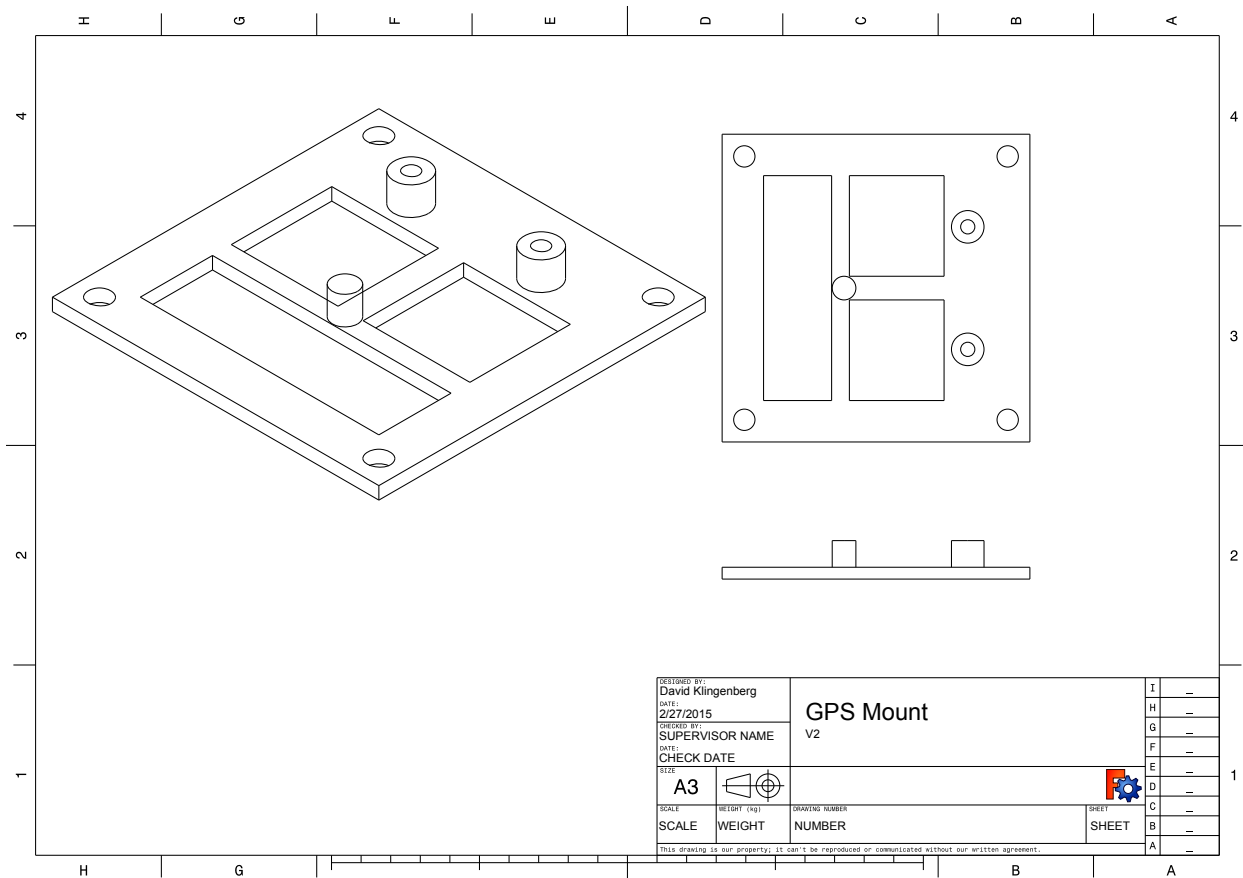


Figure 13: Gps Mount

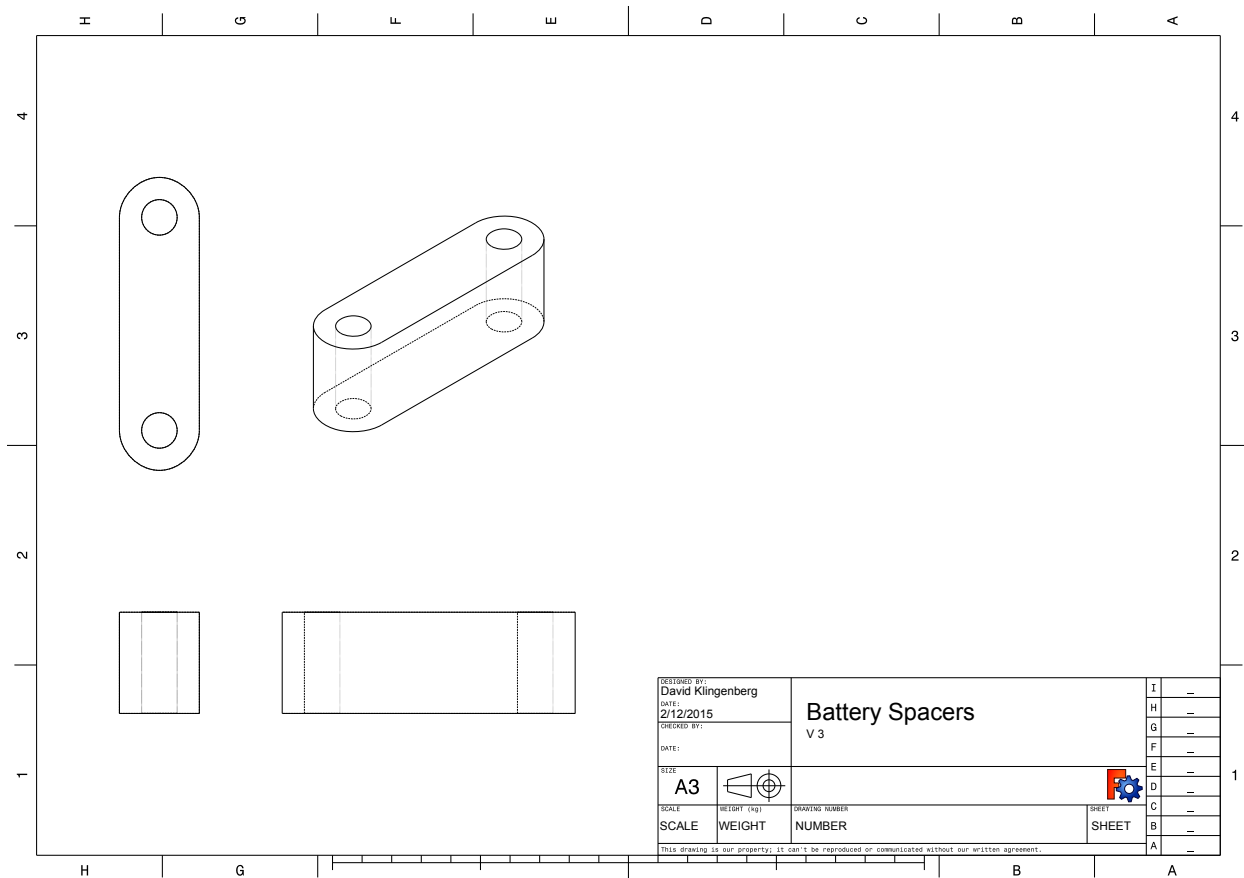


Figure 14: Battery Box Spacers

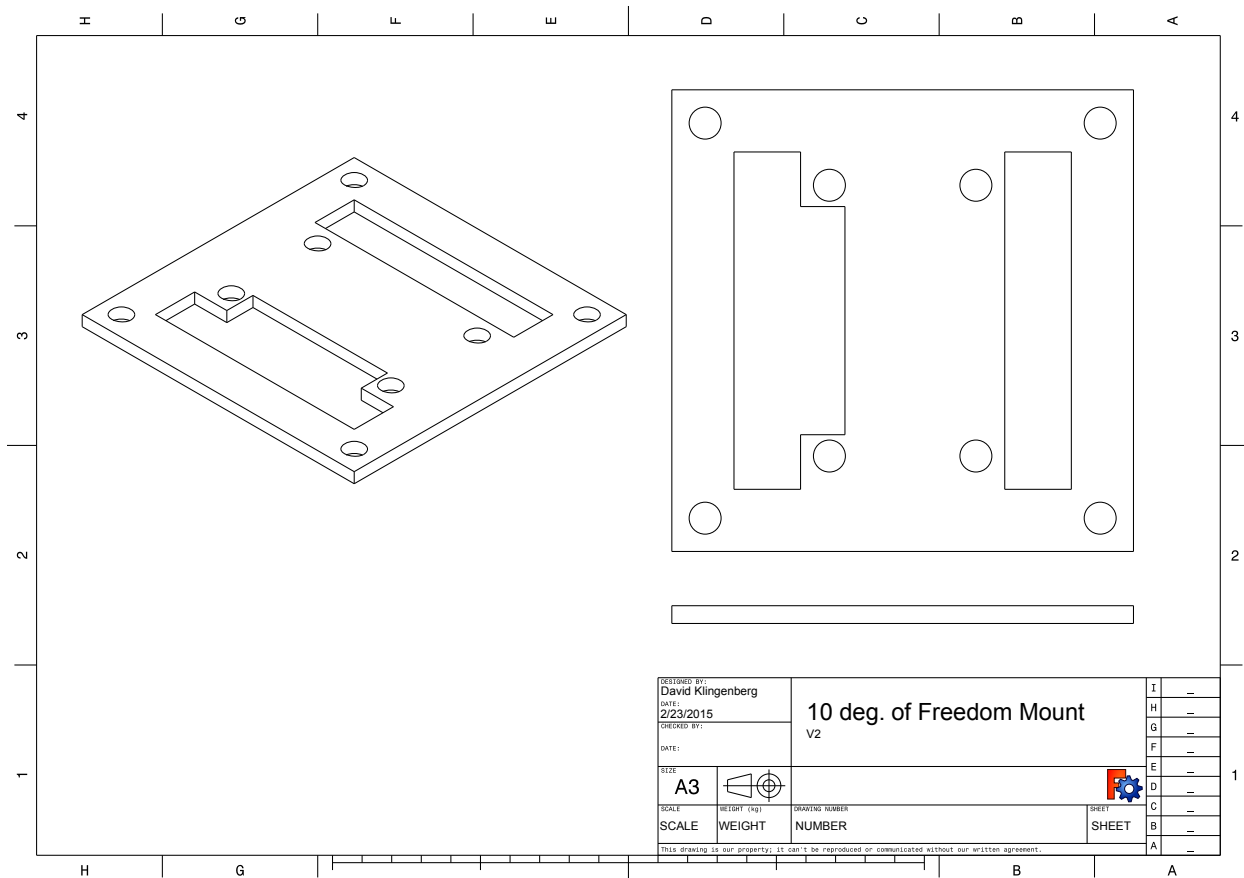


Figure 15: 10 Deg. of Freedom Sensor Platform

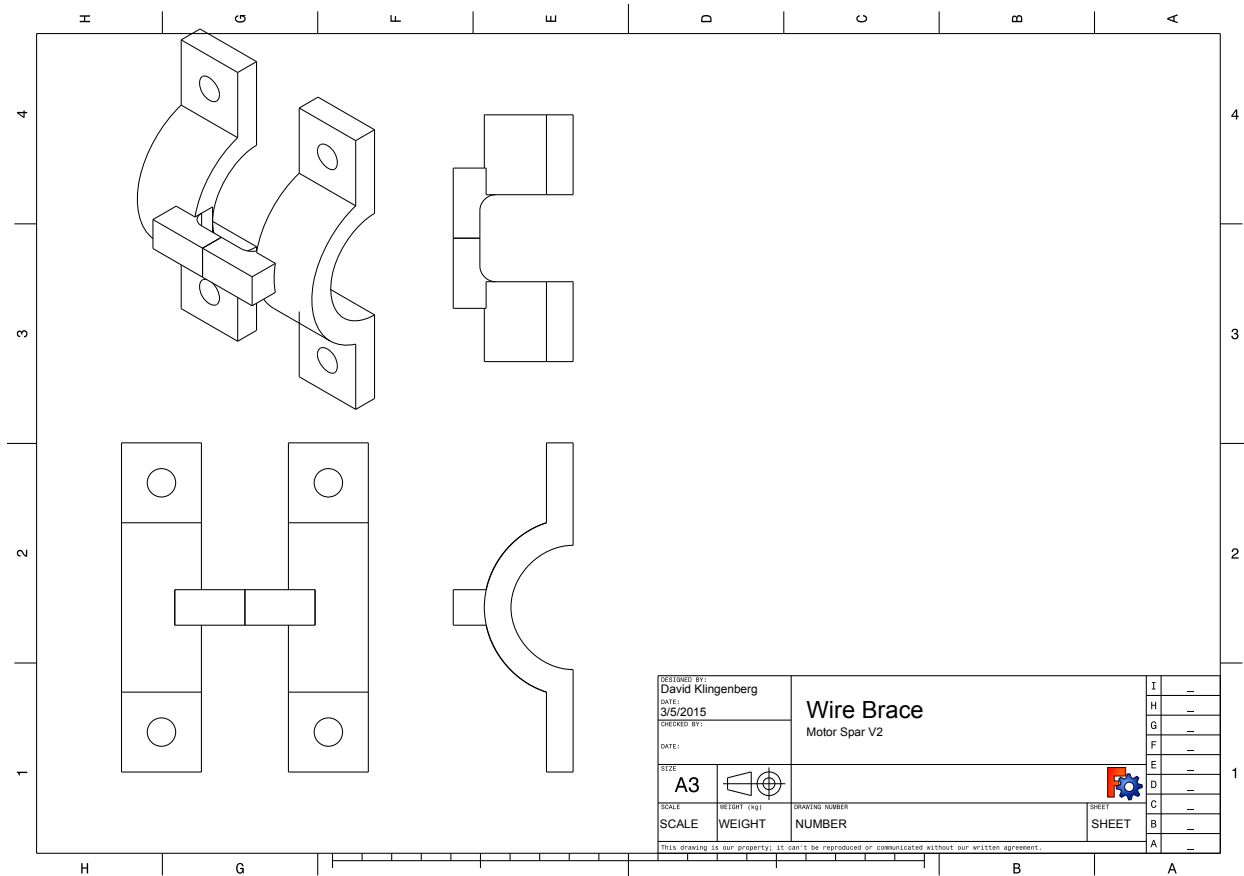


Figure 16: Wire Brace

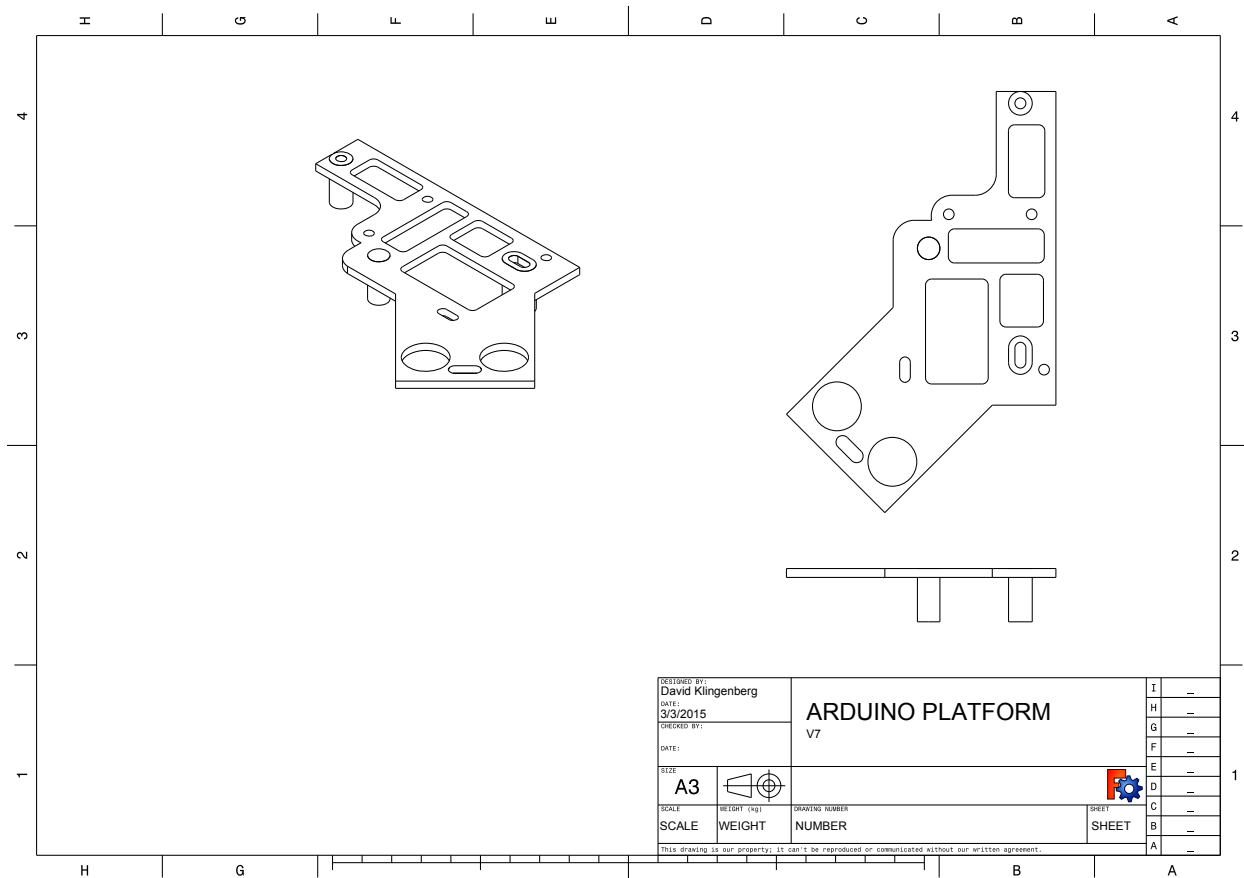


Figure 17: Arduino Platform

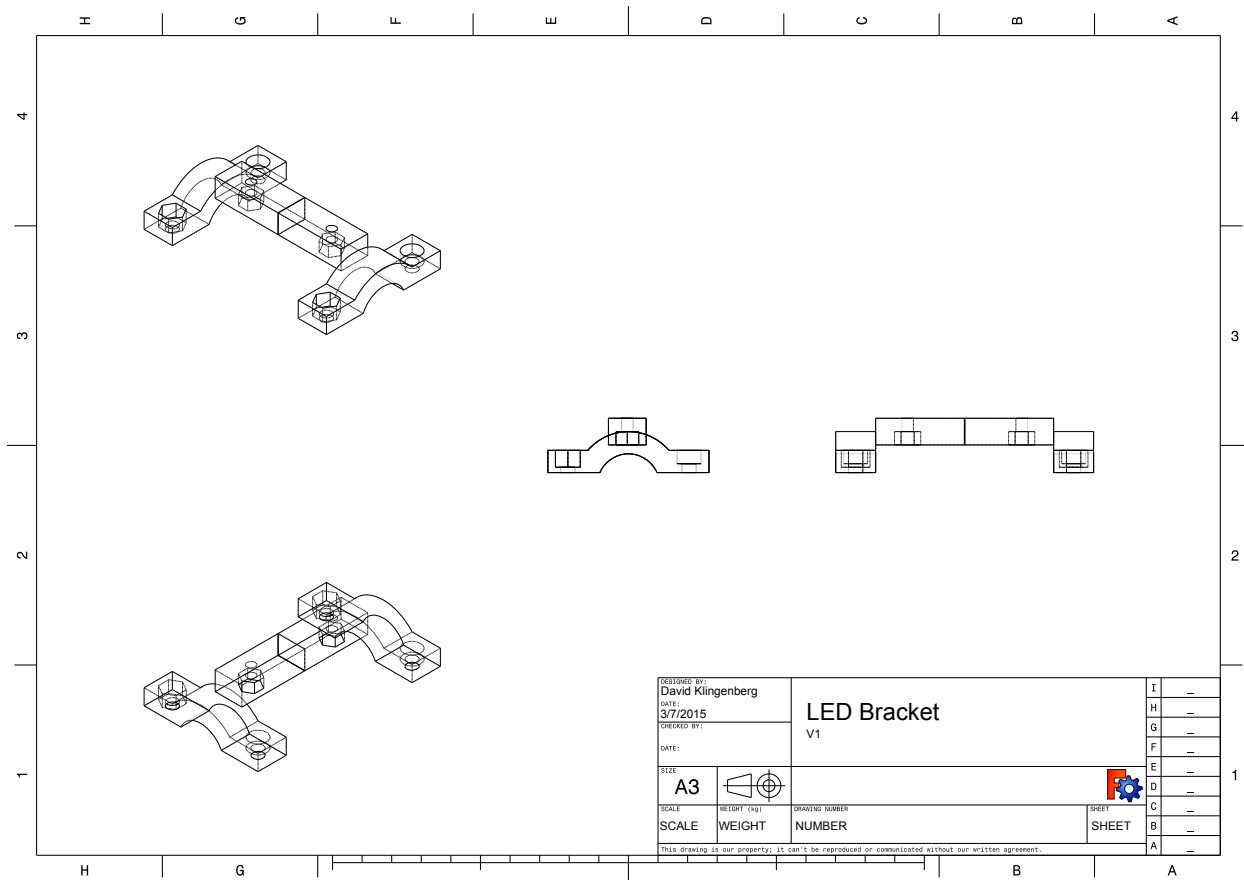


Figure 18: LED Bracket