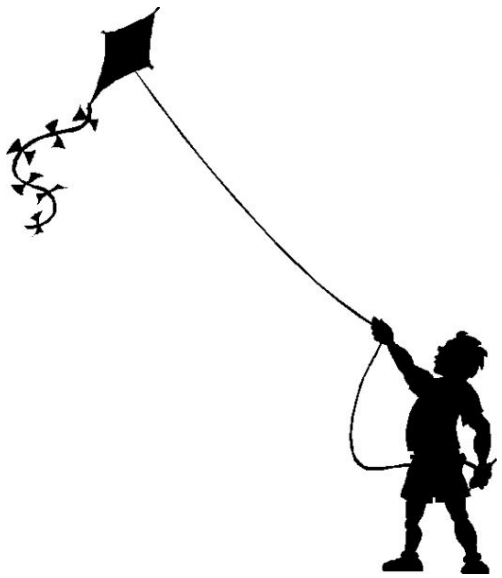

ENPH 479 Final Report
Group 1251 - Kite Runners

Prepared for: Dr. Jon Nakane
Prepared by: Adrien Emery, Liam Lawson, Craig Reitmeier

January 7, 2013

Executive summary

With energy consumption soaring, and the resulting emissions becoming a serious global environmental problem, we aim to participate in the ongoing research to create an innovative renewable energy system that generates power from high altitude winds. By using a kite system we are able to harness the immense power of high altitude winds that are unreachable by traditional wind turbines. Kites also have the advantage of being low cost, simple to build, modular, and highly mobile. Our system will harness this power by attaching the kite tether to a spool/drum that is connected in-line with a generator. The kite is flown at a high angle of attack (AOA), or in "power mode", and in a crosswind figure-eight pattern to maximize speed and lift. As the kite is pushed out by the wind the drum is free to rotate and drive the generator until the end of the tether is reached. At this point the kite is "depowered", or flown at a low AOA, and reeled in by the motor using only a small fraction of the energy generated during the power cycle. This cycle then is repeated continuously generating power.

Over the course of the spring semester of 2012 we were able to fully design and build our first prototype of the kite controller that will enable the kite system to fly in the pattern required to generate power. The kite controller itself is made up of DC motors that reel in or out the kite line and was controlled by an onboard micro-controller. This prototype, although not yet autonomous, allowed us to refine our remote control software and improve our mechanical system to better meet our future goals.

This semester we moved forward with two primary focuses. Our first was to improve the kite's autonomous flight capabilities which involved designing and building a tracking apparatus that could provide feedback data in order to control the position of the kite. Our other focus was to begin designing and assembling a ground power generation unit that could be integrated with our existing kite system to generate power.

Table of Contents

List of Figures

List of Tables

1.0 Introduction

1.1 Background

This project is self-sponsored and is actively seeking sponsorship through private and public funds. This project was first inspired by a presentation by Saul Griffith on TED on generating electricity with tethered airfoils. His company, Makani Power, is currently researching and developing prototype technology at their Hawaii headquarters. Further details on Makani and other leading researchers will be discussed later in this section.

The drive to harness the power of the wind has been around for thousands of years. Recent large-scale wind turbine projects have resulted in an increased interest in other technologies for generating wind power. The most notable is the use of large kites to create power. The general concept has been around since Miles L. Loyd's article in the Journal of Energy (1980) titled "Crosswind Kite Power." It is still a benchmark referenced by many today. And, since it was published publicly none of the ideas are patentable.

The main motivating factors for using kites over wind turbines are as follows:

- Wind speed and thus power available typically increases exponentially with elevation (Figure 1) due to boundary layer effects on the earth's surface. Since kites can fly much higher than wind turbines, they are able to harvest this energy.

- Power generation goes up by the cube of the velocity. Therefore, even a small increase in wind speed at higher altitude can lead to very large gains in available power.

- The entire Kite maintains the same relative wind speed and thus the entire surface area contributes to power generation. Whereas wind turbines have only the outer 20% of the blades contributing to the power generation since their relative wind speed is much higher than the blade area near the rotor shaft (Figure 2).

- Structurally, kites are simpler than wind turbines and are made of lightweight, flexible materials, which make for easy transport and maintenance.

- Since the generator can be placed on the ground, there is no need for expensive rare earth magnets like those used in generators on wind turbines. Rare earth magnets are used, as they are much lighter than conventional magnets and thus better structurally for a large generator supported far above the ground. Kite power generation can use conventional generators since the unit is mounted at ground level, reducing the cost of the generator unit.

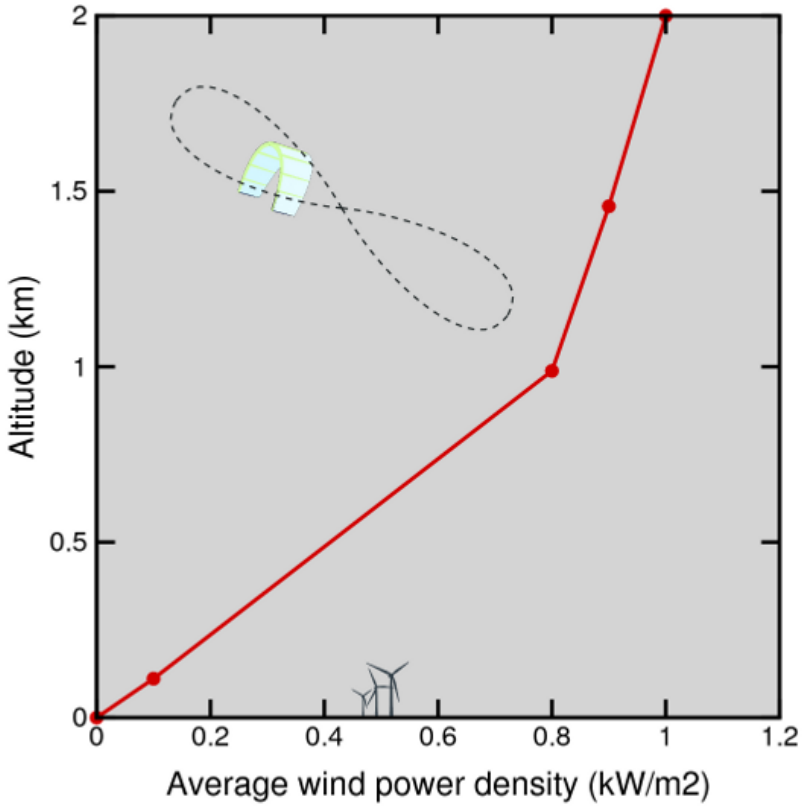

Figure 1 - Average wind power density vs. Elevation

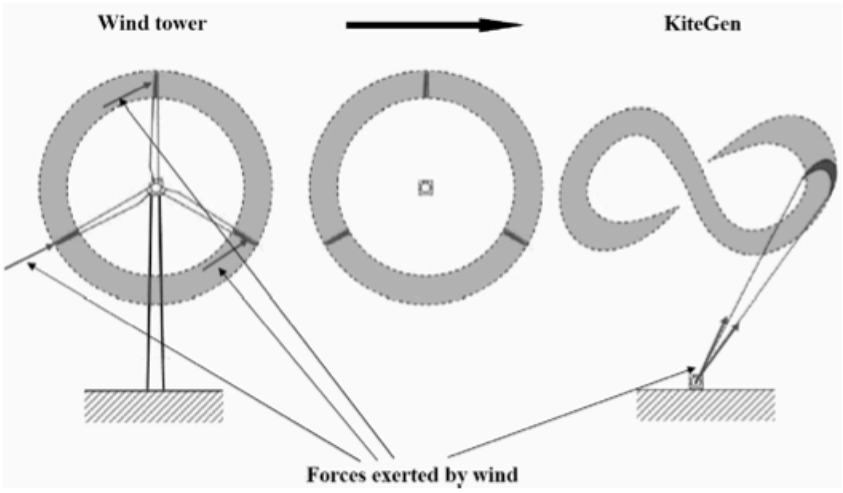

Figure 2 - Area contribution to power generation

1.1.1 Previous Results and Findings

Last semester our team worked to develop a Kite Controller Unit (KCU) that was designed to fly in the air at a fixed length from the kite (Figure 3). Figure 4 shows our prototype developed last semester. The idea was based around the Laddermill project at U of Delft. The advantages to an air based KCU are:

- The steering lines remain a fixed distance from the kite which reduces any sag in the lines
- With less total line in the air, the overall drag of the system is reduced
- Any disproportionate elongation in the lines due to unbalanced loading can be ignored.

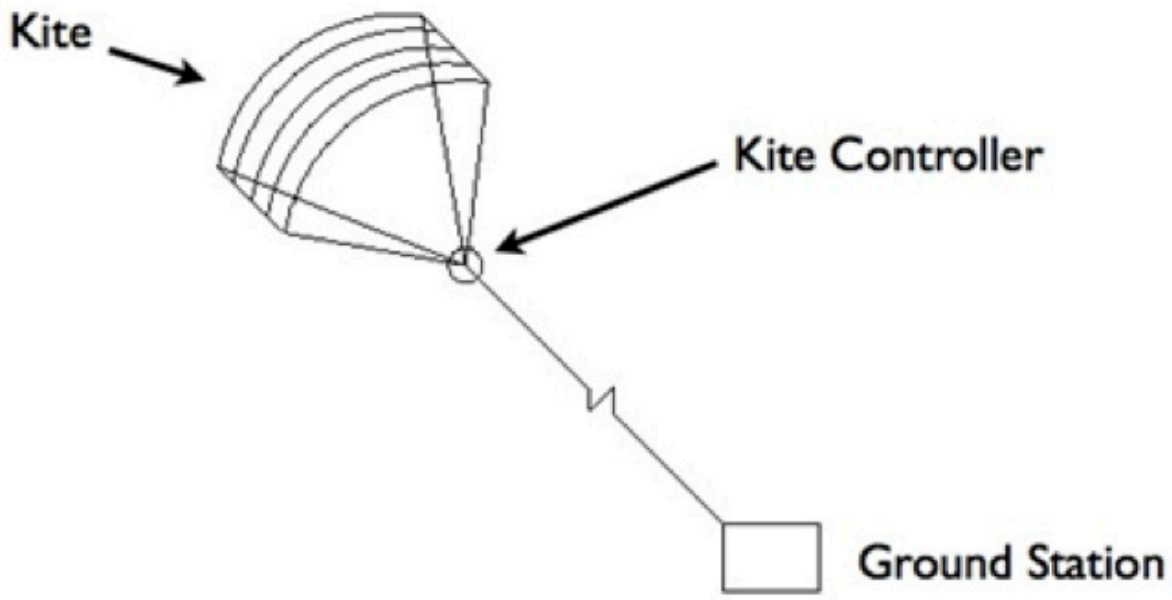

Figure 3 - Schematic of full setup

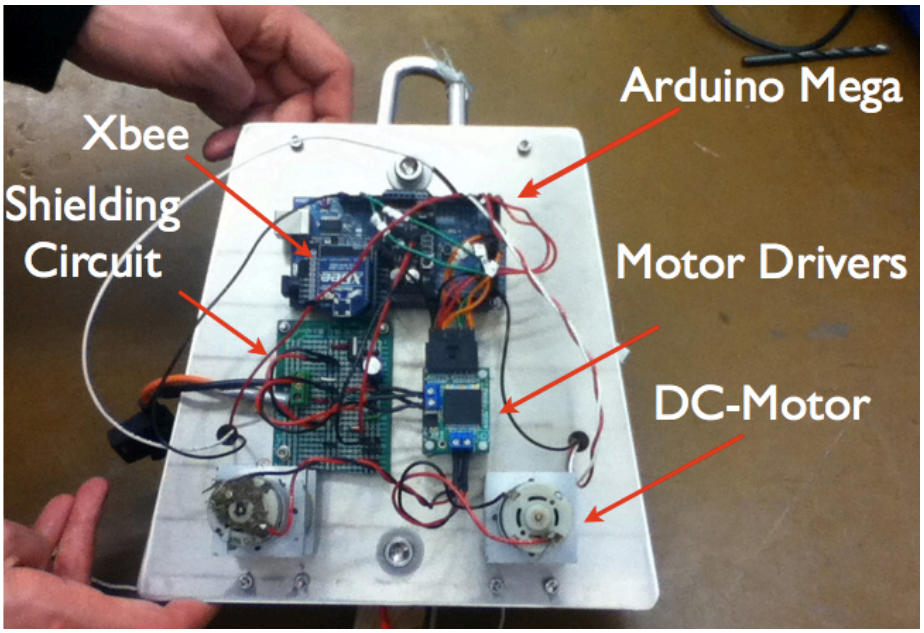

Figure 4 - Kite Controller prototype V1 (top view)

We discovered that the forces generated by our previous, 14m^2, kite were difficult to manage without more costly equipment. Therefore, we decided to scale down to a 7.5 m^2 kite for our current project. Also, we realized that although there are advantages to having an air based kite controller unit, we would redesign it to be ground based so that in the early autonomous testing stages we don't have to worry about it falling out of the air. More details on the current kite and ground based kite controller can be found in the methods section.

1.1.2 Current State of Technology

Currently, the most advanced research and prototype development has come from universities and companies in the Netherlands, Switzerland and Italy. Most notably, the Laddermill project at the University of Delft successfully generated a peak of 20kW of power using a 20 m^2 kite in 2011 [2]. However, of all companies and universities working on high altitude kite power generation, there are currently no systems in full time operation. Recent advancements in kite technology, stemming from a surge in kite sports, have made kite power more feasible but the technology is still in its infancy.

Three major organizations working on kite (or kite-like) technology are, Makani Power, Kite Gen, and Laddermill. A summary of each company will follow.

Makani Power was founded in 2006 in the USA. They are developing tethered fixed wing airfoils that have small propellers driving on board generators. They fly the wing at 250-600m altitude in a vertical circle to take advantage of crosswind effects. Power is then sent down the tether to a ground-based power station. Makani are currently testing a 30kW system but do not

have any system that is commercially available. They have plans to build 600kW and 5MW systems in the future based on their 30kW model.

The *Laddermill* project started around 2004 at the University of Delft (Netherlands). Their idea is closest related to our own - using large inflatable kites to generate power at high altitude by using the tension in the tether to drive a generator on the ground. They operate kites in the range of 14-50 m^2 to generate a peak power of 20kW. Since 2010 they have completed 160 pumping power cycles. They are also researching non-traditional kite designs, including inflatable "Kite Planes." The Laddermill group are also working in conjunction with two universities in Switzerland to further their research and development.

KiteGen is an Italian research company founded in 2007. They have developed a prototype that produces 5kW on average and 40 kW peak. It differs from our design in that there is a long arm attached to the base of the tether that helps to launch the kite. They also have ideas for a large carousel (with a generator in the middle) to be driven in a circle by multiple kites connected to the outside.

There are of course potential roadblocks that could be slowing the development of these technologies. Primarily, there is the issue of aerial regulations that would require special zoning for any permanent flying structure. Secondly, with any new large-scale power generating technology, long term reliability and feasibility tests must be conducted. And, this often requires long time scales and thus makes it difficult to find willing investors to see such a project through to the end.

1.3 Scope and Limitations

The initial objective was to design and build a kite control system that can autonomously and by remote control, control the flight pattern and AOA of the kite. Initially we would not be generating power. Once we had the kite in level remote controlled flight we would experiment with letting out and bringing in the line with the control-box attached at a fixed length below the kite, with all the control lines of the kite running into the control-box. From this point a single tether would be connected, running all the way to ground. Inside the control-box will be two motors which will reel in/out the control and trim lines to adjust the flight path and force of the kite. Although as time was limited, we decided to focus just on the remote control aspect and investigate autonomous possibilities if time allowed.

The biggest limitation in this process was the weather. Our tests were done at low altitude, in open fields in Vancouver where there was a lot of precipitation, and winds were largely unreliable. This slowed down the testing cycle more than we anticipated. Also, securing the control box to the ground proved to be challenging since the large forces on the kite require a very secure anchoring system.

1.4 Project Objectives

The objective for the project this term was to continue to design and build a high altitude kite power and control system. In order to achieve this we set the following objectives primary milestones:

- Maintaining autonomous level flight of kite on the edge of the wind window
- Autonomously making one full pass through the wind window and idling the kite on the opposite edge
- Successfully completing two or more power cycles under RC control
- Successfully completing two or more power cycles under autonomous control
- Generating more than 100W of electricity

1.5 Organization

This recommendation report will explore our solution to this specific design challenge in a variety of aspects. Beginning with the discussion, we examine our design process, manufacturing method, testing phase, and software integration. In the conclusion we will focus on the most important results and what we can draw from these results to move forward. Project deliverables provides a follow-up on our initial expectations, a general financial and parts summary, as well future commitments to this project. In the final section, we consider the future with our recommendations in order to guide any future work.

2.0 Discussion

2.1 Theory

All wind power is based on the lift equation,

$$L = \frac{1}{2} C_L \rho A V^2$$

Where, C_L is the lift coefficient that depends on the shape of the airfoil and the angle-of-attack (the angle of the leading edge with respect to the wind direction). A reasonable maximum estimate of C_L for most supported leading edge (SLE) kites is 1.3. The other three parameters are air density, ρ, surface area, A, and relative wind speed V. The maximum available power in the wind is thus,

$$P = \frac{1}{2} C_L \rho A V^3$$

By flying crosswind (perpendicular to the direction of the wind) it is possible to fly faster than the wind and thus increase the relative wind speed V. This is why it is ideal to fly in a "figure-eight" pattern as is shown in Figure 2 since the kite will always be flying crosswind and thus maximizing power output.

The concept of harnessing this power with a kite is, in theory, relatively simple. The kite tether is attached to a drum that is in turn connected to a generator. The kite is flown at a high angle-of-attack in a crosswind figure-eight pattern (as to not cross the lines). As the kite is pushed out by the wind the drum is free to rotate and drive the generator until the end of the tether is reached. At this point the angle-of-attack is reduced and the kite is flown on the edge of the wind widow. This greatly reduces the tension in the tether thus the kite can be reeled in by a motor - using only a small fraction of the energy that was generated during the power cycle. This cycle is repeated continuously (for as long as there is wind). For more detailed theory please refer to [2].

2.2 Methods

2.2.1 Our Kite

We used supported leading edge (SLE) kites of both 14m^2 and 7.5m^2 surface area. The two sizes permit flight in different wind conditions. The advantage of the SLE kite is the bridal system (Figure 5). The bridal transfers the majority of the load (working load) through the two leading edge lines. As a result, the kite can be maneuvered, with very little force, by pulling on the

two trailing edge lines. This design is what allows kite surfers to be pulled along at top speed but still control the kite with minimal effort.

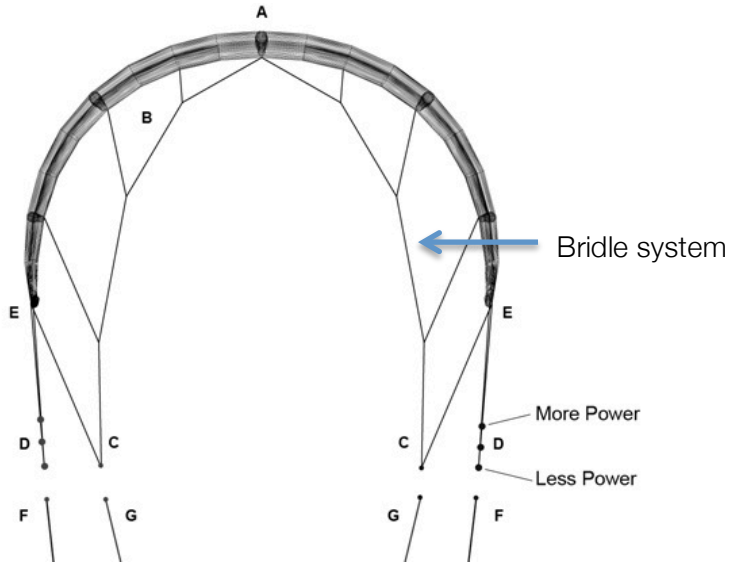

Figure 5 - SLE bridal system

2.2.2 Kite Control Unit

The kite control unit (KCU) was designed to be mounted on the flat bed of a truck or on the ground (Figure 6). The following is a summary of the main components,

- 7.5V 5000 mAh LiPo Battery Pack
- Two 30A Pololu motor drivers
- Decoupling circuit
- Two BaneBots DC motors
- Arduino Mega
- XBee
- Quadrature encoders
- Logitech webcam
- Two mini servos
- Rope guides

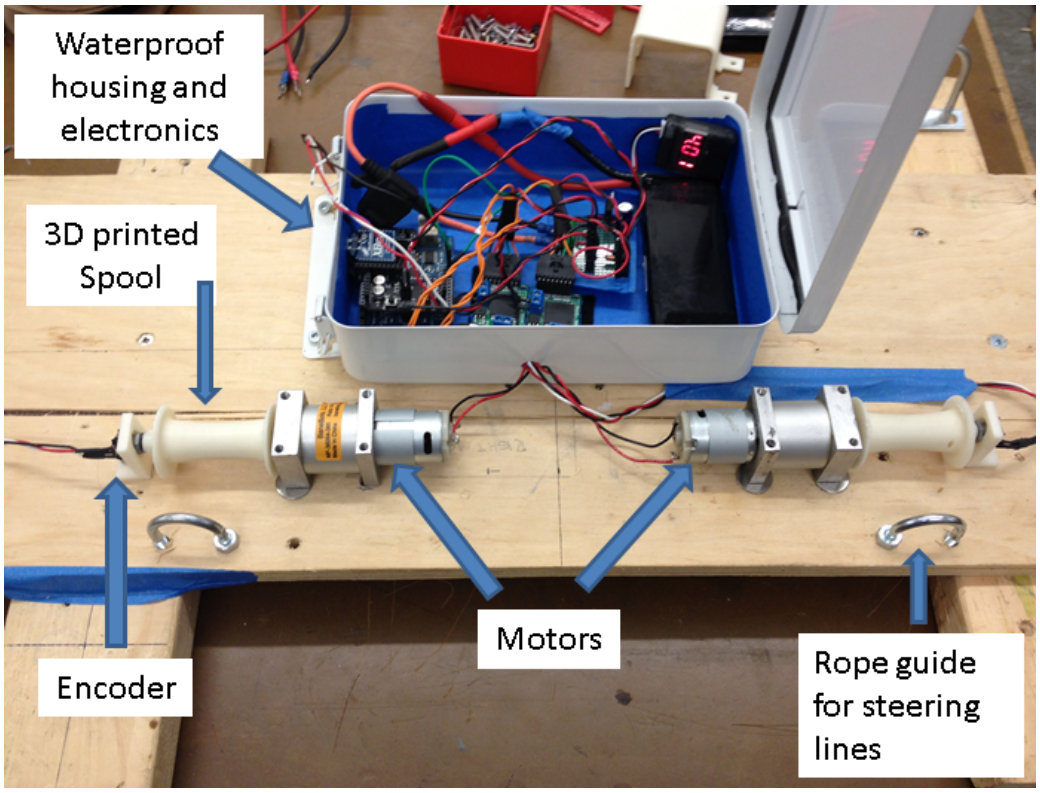

Figure 6 - Kite Control Unit

Hardware

- The spools, spool shields, encoder mounts, and pan servo mount were 3D printed.
- Spool shields were used to prevent the steering line from coming off the spool when reeling in slack line.
- The pan/tilt brackets were purchased from Robot Shop Canada.
- Rope guides were used to keep the line spooling smoothly regardless of the kite's position.

The motors were tested by spooling line around them and reeling in the line while pulling back with a force gauge. We confirmed that the motors could provide 10 in-lbs of torque while drawing 6 amps, which met our requirements and was well below the 17-amp rated stall current of the motors. Underneath, each motor has a 12-ppm quadrature encoder mounted to its shaft for position feedback control. Since we were using 1" spools this provided us with a possible positional accuracy of 1/48th of a revolution (0.065" arc length).

Electronics

The electronics on board the kite-controller consist of an Arduino Mega, XBee Maxstream, 5000mAh LiPo battery, two Pololu motor drivers, two quadrature encoders and a decoupling circuit. The only electronic component that was not purchased was the decoupling circuit. The decoupling circuit was required to maintain a consistent digital ground for the Arduino, as there can be power spikes due to the motors quickly switching direction. A schematic of the electronics layout is shown below (Figure 7).

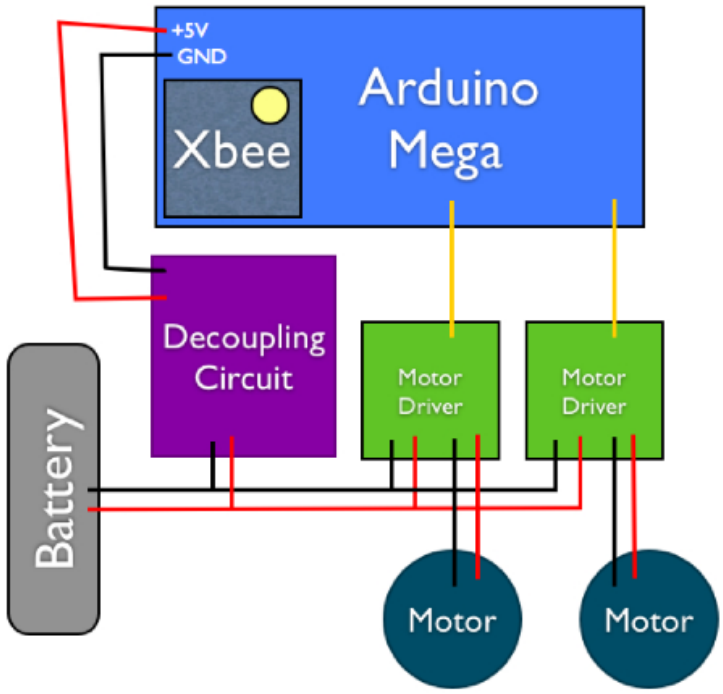

Figure 7 - KCU electronics schematic

2.2.2 Color Tracking

To fly the kite autonomously we need to know a few key things:

1. azimuth
2. elevation
3. heading
4. speed

We decided to collect this data using image processing from a video feed from a webcam mounted on the KCU. The advantage of this method is that we don't need to mount any sensors on the kite and risk having them fall off while in flight. Also the sensors required cost much more than a webcam. Since webcams don't have a very large field of view (FOV), we implemented a servo-controlled pan/tilt system to move the webcam such that the kite is always in frame. Using

the OpenCV C++ library we were able to filter the input image and determine the kite's position (Figure 8).

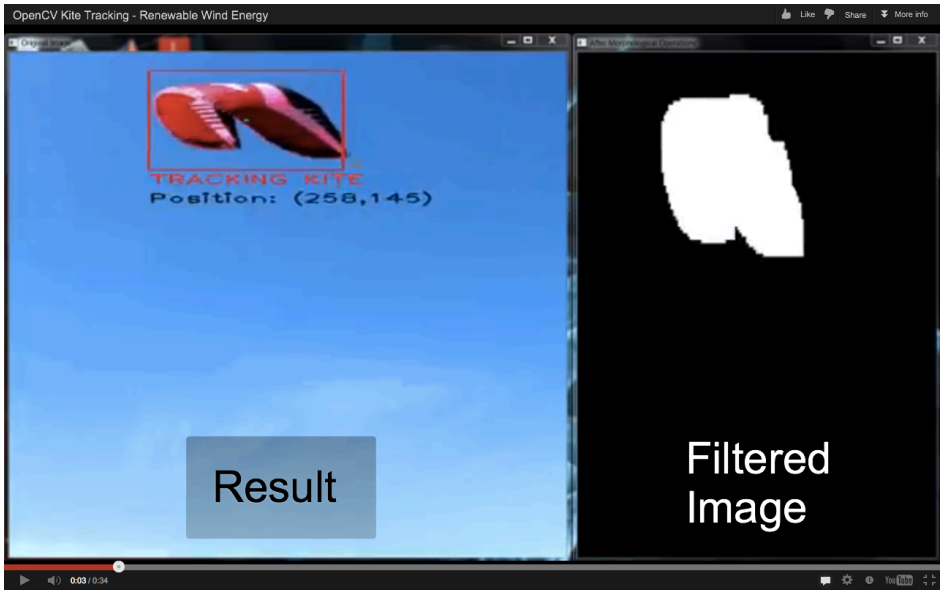

Figure 8 - Kite color tracking with OpenCV

To filter the input image the following procedure was employed:

1. Convert image to HSV (Hue-Saturation-Value) color space (this is standard color tracking practice)
2. Adjust the minimum and maximum threshold values of H, S, and V such that all pixels that fall within the range are given a value of 255 (white) or 0 (black).
3. Use OpenCv's built in contourFinder() function to draw a bounding rectangle around the filtered image.
4. Determine the centroid of that rectangle to input into our flight control algorithm.

2.2.4 Flight Control Algorithm

Based on the input data from the color tracking we are able to determine the kite's centroid position in a 2D plane. The position is then input to the following vector diagram. In the white region the green lines represent the desired path in a given direction. The error between the centroid and desired path is calculated and input into a PID loop, which computes how much to turn left or right in order to minimize the error value.

When the kite enters the yellow region it will bank around and attempt to re-enter the white region at the start of the flight path. If not a new flight path will be generated based on the kites entry point to reduce large fluctuations. If the kite is in the red region it will implement a maximum turn state in the direction that directs it away back into the white region.

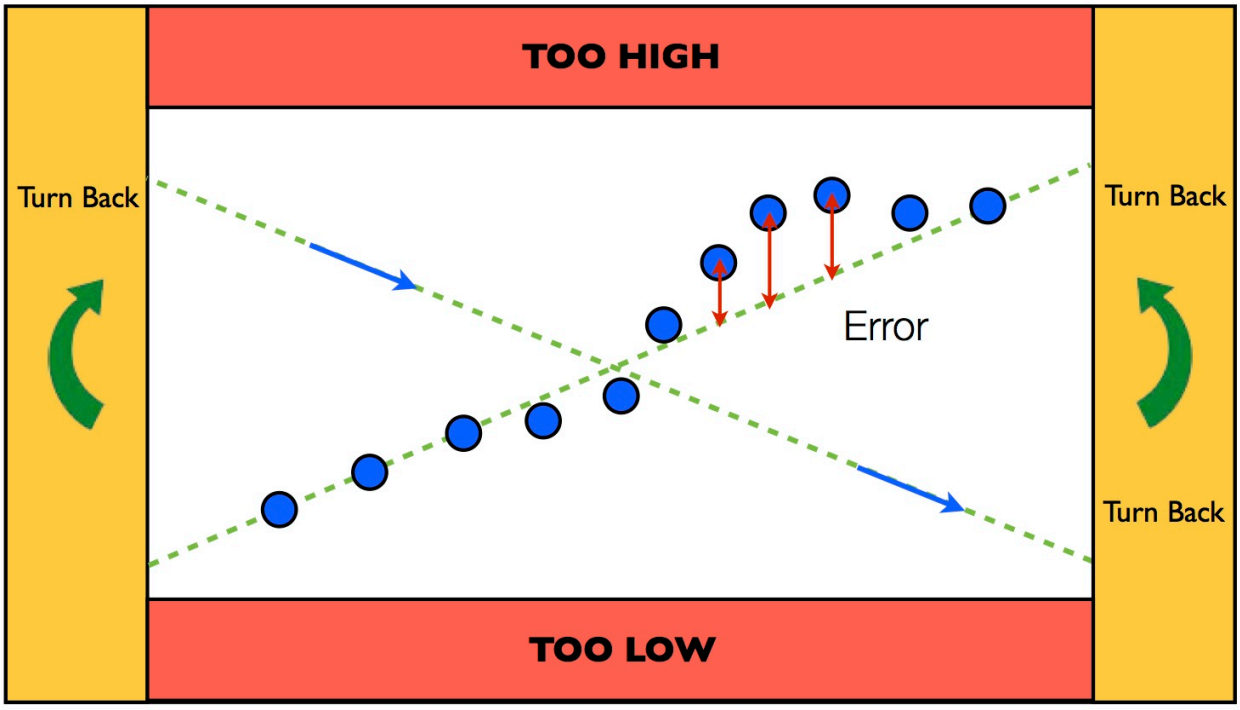

Figure 9 - Flight path algorithm schematic

2.2.5 Control Software

The force generated by the kite (based on angle-of-attack) and roll angle are controlled by the following two parameters:

1. How much each line is let in or out (ie. how far left or right you want to turn)
2. How far in our out both lines are relative to a zero (maximum power) position
3. To control both of these parameters simultaneously we implemented the following algorithm on the Arduino:
4. Determine the power state (ie. how far both lines are let out relative to a “zero” power point).
5. Determine how far left or right you want to turn (a percentage value of the maximum turn value which is to be calibrated).
6. If turning left, add a “turn state” to the power state of the left motor (ie. reel it in) and subtract from the power state of the right motor (ie. reel it out).
7. If turning right, do the opposite of step 3.
8. Tell the motors to turn until they reach their respective power + turn state.
9. If the increasing or decreasing power - add or subtract respectively from both motors power states the same amount while maintaining their turn state offset.

Using this algorithm we were able to simulate the same control movements that were requiredto manually fly the kite (Refer to Appendix A for the full Arduino code). With the motors being ensioned by the steering lines it turned out that only proportional gain control was needed to provide accurate position control.

2.2.6 User Interface

On the host computer a UI (Figure 10) was designed with the following functionality:

1. Serial monitor to Arduino.
2. Calibrate positions/angle of DC and servomotors.
3. Tune PID variables
4. Display image filtering data.

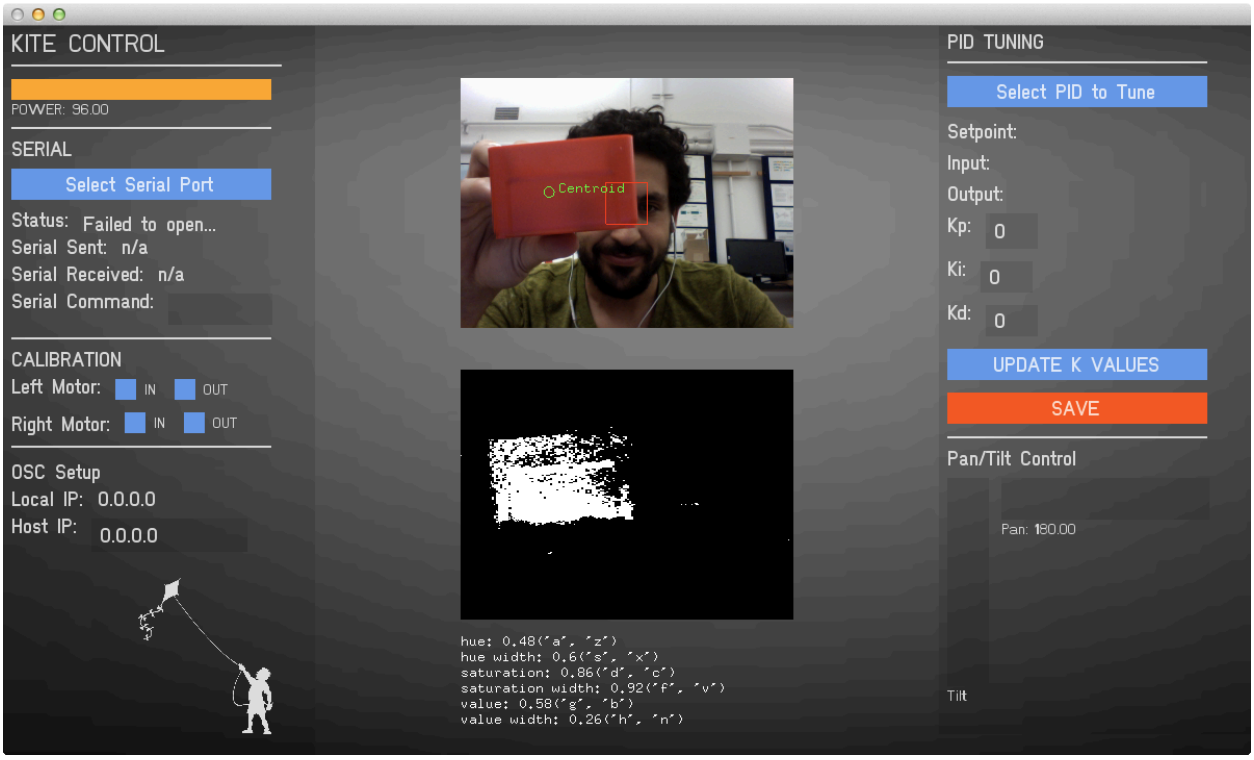

Figure 10 - Kite control UI

The UI was written in C++ using the Open Frameworks libraries. This allowed us to quickly get a graphical interface up and running with all the functionality described above. Being cross-platform Open Frameworks also allowed us to run our software on Windows or Mac operating systems. Finally, Open Frameworks comes with the OpenCV libraries already packaged.

Using the iPhone app TouchOSC, we created a custom UI (Figure 11 and Figure 12) that communicated over Wi-Fi using the OSC (open sound control) protocol. This allowed us to control useful parameters without being at the computer as well as fly the kite manually.

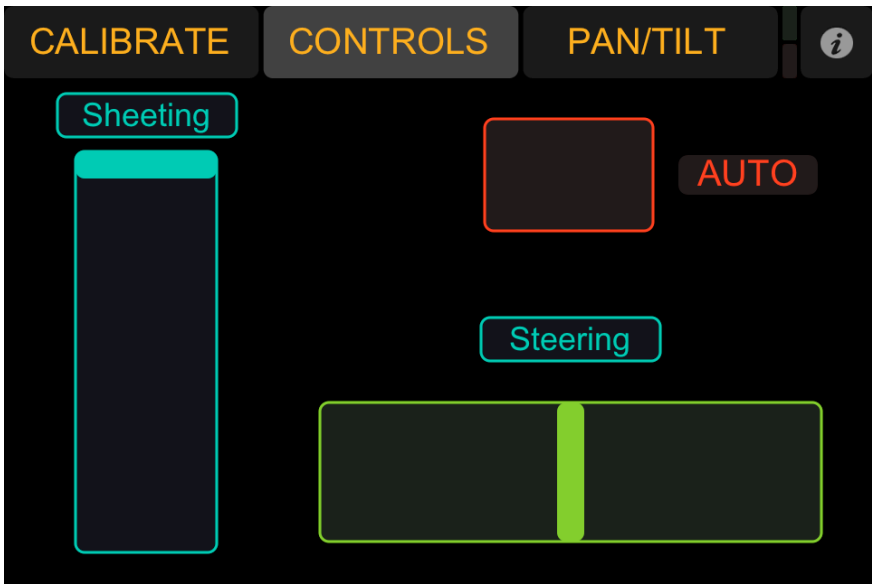

Figure 11 - iPhone UI 1

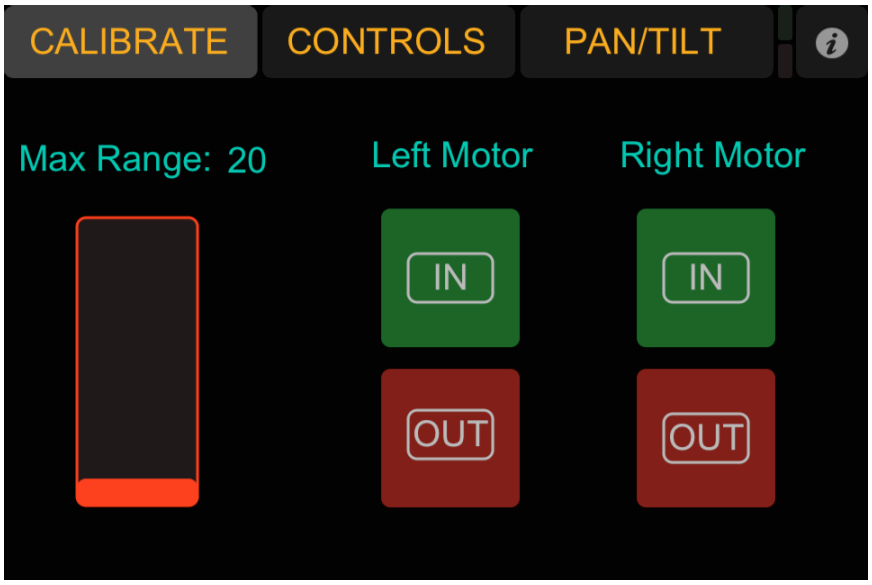

Figure 12 - iPhone UI 2

Controllable parameters include:

- Steering
- Sheeting
- Toggle autopilot
- Calibrate motors
- Adjust maximum steering range (ie max possible turn state sent to Arduino)

2.2.7 Power Generation

The power generation system consists of two separate stages; generating and motoring. The generating stage takes the alternating current and voltage output by the motor and converts

them into direct current and voltage, which can then be used to charge a battery or perform some task. The motoring stage uses battery power to rotate the motor and reel in the line. Only one stage can be active at any given time and the active stage is controlled by the Arduino microcontroller. The motor, having two phases, requires two of each stage, one per phase.

The motor is mounted on a custom designed plate and is connected to the driving shaft via a v-belt connection (Figure 13). A v-belt was used since they are inexpensive and are less dependent on precision alignment, which is useful for prototyping stages. A 4:1 ratio was used, and can be adjusted to 8:1 if necessary. An encoder on the power line spool marks the length of line that has been extended or retracted and passes the information to the Arduino which can then determine the optimum time to switch between generating and motoring.

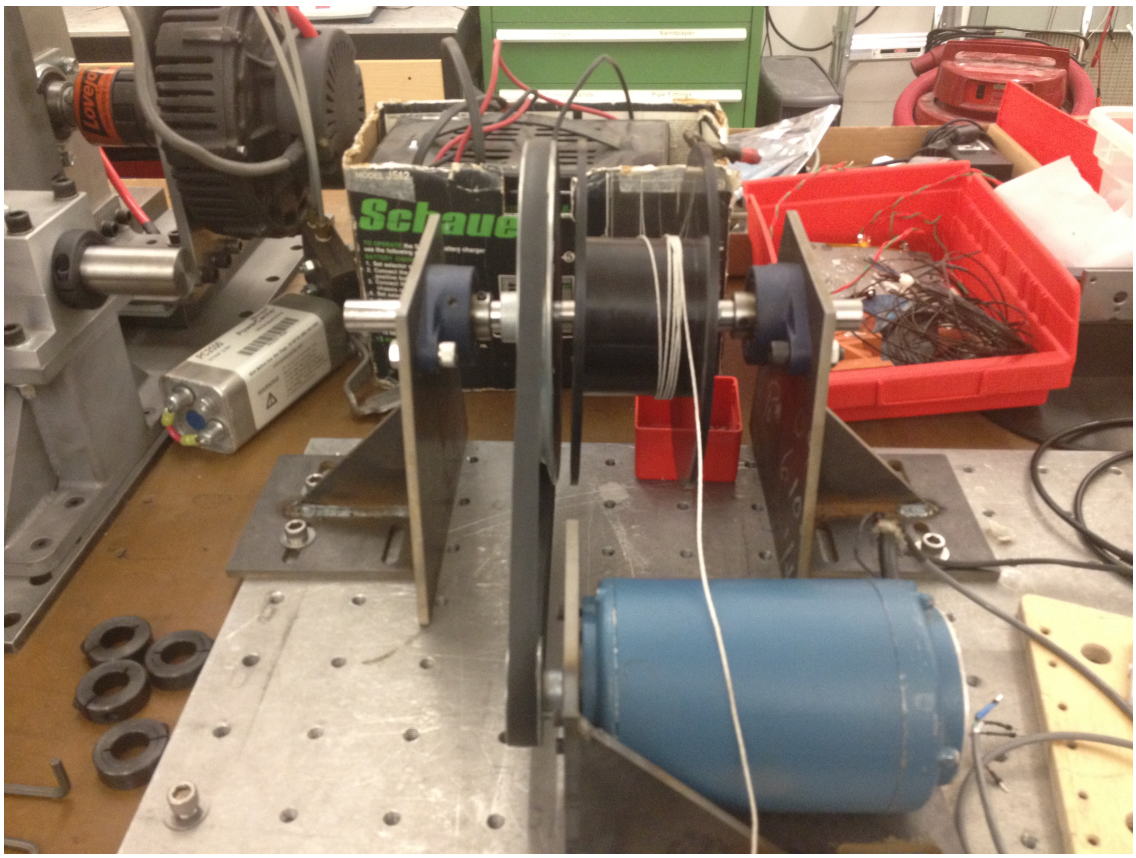

Figure 13 - Generator drive train

The generating stage of the power generation circuit currently consists of a full-wave bridge rectifier connected to a DC low-pass filter and a voltage regulator (Figure 14). The rectifier sets the current such that it will always pass the same direction through the load (or battery) and the filter removes the sinusoidal nature from the voltage to produce a nearly DC voltage. The voltage output from the filter, while DC, still varies with the rotation speed of the motor and requires regulation so as not to overcharge any battery system attached.

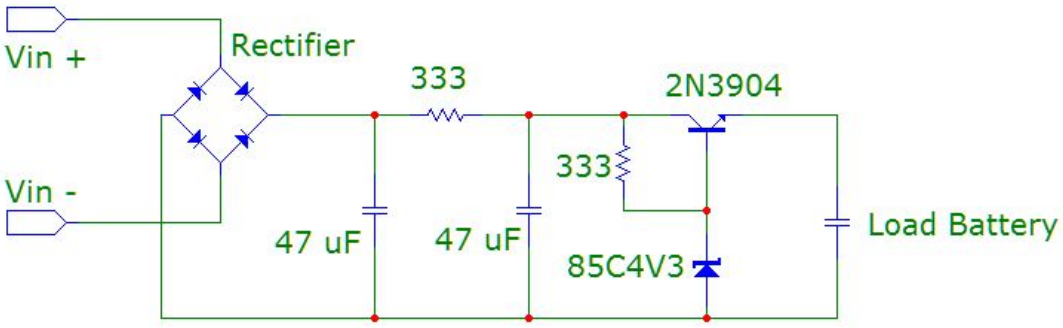

Figure 14 - Rectifier - Filter - Regulator Circuit

The motoring stage of the power generation circuit is designed to power the motor phases using the PWM outputs on the Arduino. The driver circuit is made completely of Darlington transistor switches that can change the current direction in the inductors of the motor. The Arduino requires protection against current overload when the Darlington transistors are switching, due to an inductive kick that can arise; however, the ULN2003 has most of the components for protection built in and only requires the addition of current reducing resistors to prevent a high current shock to the arduino.

2.4 Results

Initial testing with the new kite control unit confirmed that a ground-based unit is appropriate for testing purposes. This unit also allowed for a more secure platform with various locations to attach earth augers avoiding the problem we encountered in ENPH 459 where the force of the kite would irreparably damage the auger used. The Banebots planetary gearmotors (64:1) that we use for controlling the steering lines, which have a maximum of 10 lbs of pulling force while drawing 6.0 amps at 7.5 volts, have increased effectiveness through the use of the new 7.5m^2 kite versus the older 14m^2 model. The preliminary results from testing the motion capture system both in the lab and with the kite were fairly successful, being able to track the given object with very few errors. However, the wind and weather conditions during testing this semester were uncooperative meaning we were not able to test any autonomous cycles with the kite aloft.

For any power generation system, the key performance indicator is the efficiency of power generation; that is, how well does the system output one type of power when another type of power has been input. Here we input mechanical power and output electrical power, for testing purposes we input human power instead. To measure the efficiency of the power generation circuit, the voltage across a test load resistor and the motor voltage and current are all that is required. Based on testing, it was determined that the efficiency of the generation circuit was only 13%,

except at very low motor speeds. This efficiency is much lower than expectations and the most power loss is due to the voltage regulator circuit, which dumps excess voltage as current through the zener diodes. The output of the power generation circuit was a smooth DC output, which is the desired outcome. At the time of this report, the motoring stage has been designed but not built and tested.

3.0 Conclusion

Our most important result is that the work accomplished this semester, through trial and error, brings us closer to our desired end goals. This semester allowed us to make some key realizations that will allow us to reach a final prototype more quickly. The first, was to move to a smaller kite which allowed for easier handling and control using the motors we have, as well as decreasing the forces generating making it easier to secure to the ground. Secondly, we designed a ground based kite control unit that permits easier access during testing without the concern of it falling from the air and damaging itself. Thirdly, we found that power generation is possible but our current system needs to be improved for a better efficiency.

The current power generation method results in too much power lost regulating while the voltage to the required levels for charging a battery system. Future iterations will include a complete battery management system to better regulate and protect the batteries being charged by the system. With a large proportion of the flight control algorithm ready to be tested and evaluated, an availability of wind is now needed to begin implementation and move forward towards fully autonomous flight. Finally, we are confident that the results from this semester further prove our kite system is a viable source of renewable power for the future.

4.0 Project Deliverables

4.1. List of Deliverables

To design and build an operational kite system we laid out the following milestones:

Major Milestones

1. Purchase or acquire all necessary equipment and parts. (Completed)
2. Design, build, and test sensor package for kite and motor/generator system separately. (Partially completed)
3. Integrate the kite control and power generation systems, and achieve semi autonomous flight with two or more cycles of power generation. (Did not complete)

Minor Milestones

1. Redesign ground based kite control unit (KCU). (Completed)
2. Design/upgrade user interface for kite software. (Completed)
3. Find suitable testing grounds. (Retained old testing grounds until more mobile)
4. Get MPPT's working with motor/generator. (Not required for this project)
5. Create control algorithm for both power/depower cycles. (Partially Completed)
6. Mount everything to mobile testing platform (pickup truck). (Did not complete)

4.2. Financial Summary

Outlined below (Table 1) is an outline all the purchases made for this project.

Table 1 - Project cost summary

Item	Vendor	Part No.	Subtotal ($)	Purchased By
Velcro Hook & Loop	McMaster Carr	94985K811	8.41	ENPH Project Lab
V-Belt 35”	McMaster Carr	6191K28	5.75	ENPH Project Lab
V-Belt Pulley 2” OD, 3/4“ Bore	McMaster Carr	6245K17	3.92	ENPH Project Lab
V- Betl Pulley 8” OD, 3/4” Bore	McMaster Carr	6245K57	14.92	ENPH Project Lab
7.5 m^2 Naish Kite	Airtime Board Sports	N/A	200	Adrien Emery
Water Jet	UBC PHAS	00-00-01	2	ENPH Project Lab
6-axis Compass	Sparkfun	SEN-10703	29.95	ENPH Project Lab

Pressue Sensor	DigiKey	MPXH6115A6T1CT-ND	10.95	ENPH Project Lab
Water Jet	UBC PHAS	00-00-01	45	ENPH Project Lab
Keyed Shaft 3/4”	McMaster Carr	1497K161	21.2	ENPH Project Lab
V-Belt Pulley 2” OD, 1/2” Bore	McMaster Carr	6245K15	3.92	ENPH Project Lab
Shaft Colllar	McMaster Carr	6435K16	6.96	ENPH Project Lab
Steel Keystock - 3/16” x 3/16”	McMaster Carr	98830A150	1.8	ENPH Project Lab
Webcam	Best Buy		40	Adrien Emery
Pan/Tilt Brackets and Servos	RobotShop	RB-Dag-29	14.95	ENPH Project Lab
2mm Polyester Rope (30m)	West Marine	12946810	19.59	Adrien Emery
Rope Guide	West Marine	121186	11.98	Adrien Emery
		Total	432.89	

4.3. Ongoing Commitments

This project will continue development with Adrien Emery and Kyle Hounslow primarily through the course EECE 496. Liam Lawson and Craig Reitmeier have committed to provide further input and aid when needed in developing this kite system. The goals for the near future will be to improve stability, flight control, autonomous flight, as well as the efficiency of the power conversion systems.

5.0 Recommendations

As a course of action to follow, we suggest the following steps to take our kite system to a commercially viable product:

1. Achieve autonomous flight. This is a large and encompassing goal that can be taken in steps to include takeoff and landing procedures, crosswind or figure eight flight paths, pumping cycle, and emergency actions.

2. Increase flying altitude. This involves building a more durable control box that is weather resistant and can withstand impact with the ground, designing a ground station that can reel in and out the kite system, and investigating a more reliable communication method between the control box (now in the air) and ground station.

3. Continue to evaluate an efficient power generation, conversion, and storage technique. Investigating, building, and testing possible power conversion systems with the kite to generate electricity.

4. Design and build a mechanical launching and capturing system that is integrated into the ground station. This system would allow minimize the risk of the kite having contact with the ground and extend the longevity of the kite while providing increased safety for any personnel on the ground.

5. Integrate this into a reliable, highly mobile, and customer friendly product.

6.0 Appendices

6.1 Appendix A – Source Code

6.1.1 Arduino Source

```
#include <Encoder.h>
#include <Servo.h>
#include <PID_v1.h>

#define TICKS_PER_REV 48

//*** Define Variables ***//

// Pan/Tilt

Servo pan;
Servo tilt;
int newPanPos = 20;
int newTiltPos = 20;
int lastPanPos= 90;
int lastTiltPos = 90;

// Victor Motor Driver stuff

Servo leftVic;
Servo rightVic;
int maxSpeedVic = 127;
int minSpeedVic = 0;
int zeroPointVic = 127;
int leftVicPin = 9;
int rightVicPin = 8;

// Define Encoder Variables
Encoder knobLeft(20,21);
Encoder knobRight(18,19);

long currentLeft, currentRight;
long positionLeft  = 1;
long positionRight = 1;
long powerState = 96;
long newPowerState = 96;

long error1 = 0;
```

```
long error2 = 0;

// PID stuff defined here
double Setpoint, Input, Output;
double Kp,Ki,Kd = 1.0;

double leftSet, leftInput, leftOutput;
double lastLeftSet, lastLeftInput, lastLeftOutput;
double Kpl,Kil,Kdl = 1.0;

double rightSet, rightInput, rightOutput;
double lastRightSet, lastRightInput, lastRightOutput;
double Kpr,Kir,Kdr = 1.0;

PID myPID(&Input, &Output, &Setpoint,Kp,Ki,Kd, DIRECT);
PID leftPID(&leftInput, &leftOutput, &leftSet, Kpl, Kil, Kdl, DIRECT);
PID rightPID(&rightInput, &rightOutput, &rightSet, Kpr, Kir, Kdr,
DIRECT);

// Serial Contol Variables

const char notWorking = 'z';
const char changeKvals = 'k';
const char getInfo = 'i';
const char leftCalLeft = 'a';
const char leftCalRight = 'b';
const char rightCalLeft = 'c';
const char rightCalRight = 'd';
const char stopMotors = 'S';
const char autoPilotON = 'L';
const char autoPilotOFF = 'l';
const char zeroAzimuth = 'e';
const char rollSetpointCommand = 'R';  // used to send new setpoint
const char rollUpdate = 'r';           // used to update current roll
angle
const char turn = 't';
const char power = 'p';
const char depower = 'd';
const char Speed = 's';
const char terminateChar = '/';
const char handshake = 'x';
const char panCommand = 'P';
const char tiltCommand = 'T';


String command = "";
```

```
String positionStr = "";
String motorSpeedStr = "";
String powerStr = "";

int lastState = 0;
int currentState =0;
char incomingByte = 'a';
int count = 0;

long leftMax = 20*TICKS_PER_REV;
long rightMax = 20*TICKS_PER_REV;

float powerMax = 500;
float depowerMax = 500;
float horizontalCount = 50;
float deltaX = 0.001;
long newPositionR = 96;
long newPositionL = 96;
long lastPosition = 0;
long turnLeftPosition = 0;
long turnRightPosition = 0;
long currentPosition = 0;
float howFar = 0.0;

// Motor pin connections for PWNM control
//  ** THESE PINS MUST BE INITIALIZED ** //

// TODO - ASSIGN PIN MAPPINGS TO MOTOR DRIVE PINSS

int pwmPinL = 10;
int pwmPinR = 11;

int leftMotorLeft = 40;
int leftMotorRight = 41;

int rightMotorLeft = 26;
int rightMotorRight = 27;

int maxSpeed = 255;
int motorSpeed = 255;

boolean calibrating = false;
boolean powerChange = false;
boolean leftMovingLeft = false;
boolean leftMovingRight = false;
boolean rightMovingLeft = false;
```

```
boolean rightMovingRight = false;
boolean movingLeft = false;
boolean movingRight = false;
boolean changeDirection = false;
boolean atSetPoint = false;
boolean print2screen = true;
boolean bprintData = false;
boolean bauto = false;
boolean usingVictorDrivers = false;

// Define Functions

void leftMotorL(int motorSpeed = maxSpeed)
{
  if(usingVictorDrivers){
    motorSpeed = zeroPointVic - motorSpeed;
    leftVic.write(motorSpeed);
  }
  else{

    if(motorSpeed >255) motorSpeed = 255;
    if(motorSpeed < 0) motorSpeed = 0;
  }

  digitalWrite(leftMotorLeft,HIGH);
  digitalWrite(leftMotorRight, LOW);
  analogWrite(pwmPinL, motorSpeed);
}

void rightMotorL(int motorSpeed = maxSpeed)
{
  if(usingVictorDrivers){
    motorSpeed = zeroPointVic - motorSpeed;
    rightVic.write(motorSpeed);

  }else{

    if(motorSpeed >255) motorSpeed = 255;
    if(motorSpeed < 0) motorSpeed = 0;

    digitalWrite(rightMotorLeft, HIGH);
    digitalWrite(rightMotorRight, LOW);
    analogWrite(pwmPinR, motorSpeed);
  }

}
```

```
void leftMotorR(int motorSpeed = maxSpeed)
{

  if(usingVictorDrivers){
    motorSpeed = zeroPointVic - motorSpeed;
    leftVic.write(motorSpeed);

  }else{

    if(motorSpeed >255) motorSpeed = 255;
    if(motorSpeed < 0) motorSpeed = 0;

    digitalWrite(leftMotorLeft, LOW);
    digitalWrite(leftMotorRight,HIGH);
    analogWrite(pwmPinL, motorSpeed);
  }


}

void rightMotorR(int motorSpeed = maxSpeed)
{

  if(usingVictorDrivers){
    motorSpeed = zeroPointVic - motorSpeed;
    rightVic.write(motorSpeed);

  }else{

    if(motorSpeed >255) motorSpeed = 255;
    if(motorSpeed < 0) motorSpeed = 0;

    digitalWrite(rightMotorLeft, LOW);
    digitalWrite(rightMotorRight, HIGH);
    analogWrite(pwmPinR, motorSpeed);
  }
}


void holdPosition()
{
  // TODO
  digitalWrite(leftMotorLeft, LOW);
  digitalWrite(leftMotorRight,LOW);
  analogWrite(pwmPinL, 0);
```

```
  digitalWrite(rightMotorLeft, LOW);
  digitalWrite(rightMotorRight, LOW);
  analogWrite(pwmPinR, 0);

}

void stopLeft()
{
  if(usingVictorDrivers){

    leftVic.write(zeroPointVic);  // stops leftMotor

  }else{

    digitalWrite(leftMotorLeft, LOW);
    digitalWrite(leftMotorRight,LOW);
    analogWrite(pwmPinL, 0);
  }
}

void stopRight()
{
  if(usingVictorDrivers){
    rightVic.write(zeroPointVic);

  }else{

    digitalWrite(rightMotorLeft, LOW);
    digitalWrite(rightMotorRight, LOW);
    analogWrite(pwmPinR, 0);
  }
}

void printData(){
  Serial.print("\nInput: ");
  Serial.print(Input);
  Serial.print(" Setpoint: ");
  Serial.print(Setpoint);
  Serial.print(" Output:  ");
  Serial.println(Output);

  Serial.print("Kp: ");
  Serial.print(Kp);
  Serial.print("  Ki: ");
  Serial.print(Ki);
```

```
  Serial.print("  Kd: ");
  Serial.println(Kd);

  Serial.print("L: ");
  Serial.print(turnLeftPosition);
  Serial.print("  R: ");
  Serial.println(turnRightPosition);

  bprintData = false;

}


void setup() {

  Serial.begin(115200);   // open serial port
  Serial.flush();
  Serial.println("Kite Control 1.23");

  // Initialize PIDs
  Setpoint = 0;
  Input = 0;
  myPID.SetMode(AUTOMATIC);
  myPID.SetOutputLimits(-30,30);
  myPID.SetSampleTime(100);
  myPID.SetTunings(1,0,0);

  leftSet = 96;
  leftInput = 96;
  Kpl = 4.0;
  Kdl = 0.0;
  Kil = 0.0;

  rightSet = 96;
  rightInput = 96;
  Kpr = 4.0;
  Kir = 0.0;
  Kdr = 0.0;

  leftPID.SetMode(AUTOMATIC);
  leftPID.SetOutputLimits(-127,127);
  leftPID.SetSampleTime(200);
  leftPID.SetTunings(Kpl,Kdl,Kil);

  rightPID.SetMode(AUTOMATIC);
  rightPID.SetOutputLimits(-127,127);
```

```
  rightPID.SetSampleTime(200);
  rightPID.SetTunings(Kpr,Kir,Kdr);

  knobLeft.write(TICKS_PER_REV*12);
  knobRight.write(TICKS_PER_REV*12);

  // initialize victor motor driver pins
  leftVic.attach(leftVicPin);
  rightVic.attach(rightVicPin);

  // initialize pan/tilt servos
  pan.attach(7);
  tilt.attach(5);

  pan.write(90);
  tilt.write(90);

  // set motor speeds to zero
  stopLeft();
  stopRight();

}

void loop() {

   count ++;
   if(count == 150){
     count = 0;
   }

  // Do stuff if data is available to read
  if(Serial.available() > 0)
  {
    command = "";
    incomingByte = Serial.read();

    while(incomingByte != terminateChar)  // Read from serial until
terminating char is recieved
    {
      if(Serial.available() > 0 ){
        command += incomingByte;    // concatonate command with
incoming byte
        incomingByte = Serial.read();  // Read byte from serial port
      }
```

```
//      if(abs(t0 - millis()) > 2000){  // break out of serial reading
after 2 seconds
//        command = " ";
//        break;
//      }

    }
    Serial.print('r');        // done reading data, so give PC ok to
write more data

    // Check first index in incoming command to determine the command
given
    if(command[0] == turn)
    {
      positionStr = command.substring(1);  // assigns rest of position
command string
      turnLeftPosition = - long(positionStr.toFloat());  // convert to
float
      turnRightPosition = -turnLeftPosition;
      // check that turn command doesn't exceed max turn
//      if(turnLeftPosition > 20) turnLeftPosition = 20;
//      if(turnLeftPosition < -20) turnLeftPosition = -20;
//      if(turnRightPosition > 20) turnRightPosition = 20;
//      if(turnRightPosition < -20) turnRightPosition = -20;

      newPositionL = turnLeftPosition + powerState;
      newPositionR = turnRightPosition + powerState;

    } else if(command[0] == power){
      powerStr = command.substring(1);
      newPowerState = int(powerStr.toFloat());
      powerState = newPowerState;
      powerChange = true;
      newPositionL = turnLeftPosition + newPowerState;
      newPositionR = turnRightPosition + newPowerState;

    } else if(command[0] == Speed) {
      motorSpeedStr = command.substring(1);  // assings rest of speed
command to string
      motorSpeed = int(motorSpeedStr.toFloat()); // convert to int
      if(motorSpeed > 255) motorSpeed = 255;
      if(motorSpeed < 0 ) motorSpeed = 0;
      // TODO do stuff with user input for moto speed
      //Serial.print("New motor speed: ");
      //Serial.println(motorSpeed);
```

```
    } else if(command[0] == leftCalLeft) {
      leftMotorL(75);
      calibrating = true;

    } else if(command[0] == leftCalRight){
      leftMotorR(75);
      calibrating = true;

    } else if(command[0] == rightCalLeft){
      rightMotorL(75);
      calibrating = true;

    } else if(command[0] == rightCalRight){
      rightMotorR(75);
      calibrating = true;

    } else if(command[0] == stopMotors){    // reset all values to
default
      stopLeft();
      stopRight();
      delay(200);
      knobLeft.write(TICKS_PER_REV*12);
      knobRight.write(TICKS_PER_REV*12);
      powerState = 96;
      newPositionL = 96;
      newPositionR = 96;
      calibrating = false;

    } else if(command[0] == autoPilotON){
      //rollPID.SetMode(AUTOMATIC);  // turn ON roll PID
      bauto = true;
      myPID.SetMode(AUTOMATIC);
      //Serial.println("AUTOPILOT ON");

    } else if(command[0] == autoPilotOFF){
      //rollPID.SetMode(MANUAL);  // turn OFF roll PID
      bauto = false;
      myPID.SetMode(MANUAL);
      //Serial.println("AUTOPILOT OFF");


    } else if(command[0] == rollSetpointCommand && bauto){
      Setpoint = double(command.substring(1).toFloat());   // update
setpoint
      //Serial.print("roll setpoint: ");
      Serial.println(Setpoint);
```

```
    } else if(command[0] == rollUpdate && bauto){
      Input = double(command.substring(1).toFloat());  // update
current roll angle
      //Serial.print("roll input: ");
      Serial.println(Input);

    }else if(command[0] == getInfo){
       bprintData = true;

    }else if(command[0] == changeKvals){

      float temp[3];
      String tempStr;
      int j = 0;

      for(int i = 2; i < command.length(); i++){
         if( command[i] != ','){
            tempStr += command[i];
         }else{
           temp[j] = tempStr.toFloat();
           j++;
           tempStr = "";
         }
      }
      temp[j] = tempStr.toFloat();  // catch last value if no comma is
typed after value

      Kp = temp[0];
      Ki = temp [1];
      Kd = temp [2];

      if(command[1] == 'k'){
        myPID.SetTunings(Kp,Ki,Kd);
      }else if(command[1] == 'l'){
        leftPID.SetTunings(Kp,Ki,Kd);
      }else if(command[1] == 'r'){
        rightPID.SetTunings(Kp,Ki,Kd);
      }else{
        //Serial.println("Invalid PID value entry");
      }

//      Serial.print("Kp: ");
//      Serial.print(Kp);
//      Serial.print("  Ki: ");
```

```
//      Serial.print(Ki);
//      Serial.print("  Kd: ");
//      Serial.println(Kd);

      stopLeft();
      stopRight();
      knobLeft.write(TICKS_PER_REV*12);
      knobRight.write(TICKS_PER_REV*12);
      powerState = 96;
      newPositionL = 96;
      newPositionR = 96;
      calibrating = false;

    } else if(command[0] == panCommand){
      newPanPos = int(command.substring(1).toFloat());
      if(newPanPos != lastPanPos){
          pan.write(newPanPos);
          lastPanPos = newPanPos;
          //Serial.print("PAN position: ");
          //Serial.println(newPanPos);
      }

    }else if(command[0] == tiltCommand){
      newTiltPos = int(command.substring(1).toFloat());
      if(newTiltPos != lastTiltPos){
          tilt.write(newTiltPos);
          lastTiltPos = newTiltPos;
          //Serial.print("TILT position: ");
          //Serial.println(newTiltPos);
      }

    }else {
       // Put all catch commands in this if statement
       //Serial.println("Invalid Input!");
    }

  }

  // Compute PID output to minimze error in roll angle when in
autopilot mode
  if(bauto){
    myPID.Compute();
    turnLeftPosition = - long(Output);
    turnRightPosition = -turnLeftPosition;
    newPositionL = turnLeftPosition + powerState;
    newPositionR = turnRightPosition + powerState;
```

```
  }

  /*
     ENCODER TRACKING ALGORITHM
  1. Determine Current position
  2. Compare current Position to setPostion
  3. Use PID to calculat motor speed required to get to new position
  4. If current setPosition is less than current position Move left
  5. Else Move right

  */

  // read in current encoder position
  currentLeft = knobLeft.read();
  currentRight = knobRight.read();

  // Convert to 1/8th revolution resolution
  double numRevs = currentLeft/TICKS_PER_REV;
  double numHalfRevs = currentLeft/(TICKS_PER_REV/8);
  double numHalfRevsRight = currentRight/(TICKS_PER_REV/8);

  // Update PID input values
  leftInput = numHalfRevs;
  rightInput = numHalfRevsRight;

  // Compute new PID outputs
  leftPID.Compute();
  rightPID.Compute();
  myPID.Compute();

  // print output values if they have changed
  if(lastLeftOutput != leftOutput || lastRightOutput != rightOutput){
//     Serial.print("Left PID output: ");
//     Serial.print(leftOutput);
//     Serial.print("  Right PID output: ");
//     Serial.println(rightOutput);
//     Serial.print("Revs - L:");
//     Serial.print(numHalfRevs);
//     Serial.print(" R: ");
//     Serial.println(numHalfRevsRight);
//     Serial.print("Left setpoint: ");
//     Serial.print(leftSet);
//     Serial.print("  Right setpoint: ");
//     Serial.println(rightSet);
  }
```

```
  // update last output readings
  lastLeftOutput = leftOutput;
  lastRightOutput = rightOutput;


  leftOutput = -leftOutput;
  rightOutput = - rightOutput;

 if(bprintData){
      printData();
  }
  if(!calibrating)
  {
    if(usingVictorDrivers){

      if(newPositionL < numHalfRevs){    // reel IN line
          leftMotorL(leftOutput);
      } else if(newPositionL > numHalfRevs){  // reel OUT line
          leftMotorR(leftOutput);
      }
      if(newPositionR < numHalfRevsRight){  // reel OUT line
          rightMotorR(rightOutput);
      } else if(newPositionR > numHalfRevsRight){  // reel IN line
          rightMotorL(rightOutput);
      }

      if(newPositionL == numHalfRevs){        // stop left motor
          stopLeft();
      }
      if(newPositionR == numHalfRevsRight) { // stop right motor
          stopRight();
      }

      if(abs(newPositionL - numHalfRevs) <= 2) {  // lowers motor
speed if close to setpoint
         stopLeft();
      }

      if(abs(newPositionR - numHalfRevsRight) <= 2) {  // lowers motor
speed if close to setpoint
        stopRight();
      }

    }else{
```

```
      if(newPositionL < numHalfRevs){    // reel IN line
          leftMotorL(motorSpeed);
      } else if(newPositionL > numHalfRevs){  // reel OUT line
          leftMotorR(motorSpeed);
      }
      if(newPositionR < numHalfRevsRight){  // reel OUT line
          rightMotorR(motorSpeed);
      } else if(newPositionR > numHalfRevsRight){  // reel IN line
          rightMotorL(motorSpeed);
      }

      if(newPositionL == numHalfRevs){
          stopLeft();
      }
      if(newPositionR == numHalfRevsRight) {
          stopRight();
      }

      if(abs(newPositionL - numHalfRevs) <= 1) {  // lowers motor
speed if close to setpoint
          //stopLeft();
          motorSpeed = 100;
      }else {
        motorSpeed = 255;
      }

      if(abs(newPositionR - numHalfRevsRight) <= 1) {  // lowers motor
speed if close to setpoint
        //stopRight();
        motorSpeed = 100;
      }else {
        motorSpeed = 255;
      }
    }

     if(numHalfRevs!= positionLeft || numHalfRevsRight !=
positionRight)
     {
//       Serial.print("Revs - L:");
//       Serial.print(numHalfRevs);
//       Serial.print(" R: ");
//       Serial.println(numHalfRevsRight);
       positionLeft = numHalfRevs;
       positionRight = numHalfRevsRight;
     }
 }
```

```
  // TODO
  // Check battery levels of each cell in LiPo Pack
  // if they are low cut power to motors and send Message to PC

}

// End of file
```

6.1.2 Open Frameworks Source

testApp.h

```
#pragma once

#include "ofMain.h"
#include "ofxOsc.h"
#include "ofxSimpleSerial.h"
#include "ofxUI.h"
#include "ofxOpenCv.h"
#include <sstream>
#include "colorTracking.h"

#define HOST "169.254.47.98" // This is ip address of my local
computer
#define INPORT 8000       // This port must match the output port
number in TouchOSC
#define OUTPORT 9000    // This port must match the input port on
TouchOSC
#define DEVICE 0        // This serial port number must match the port
of the XBEE
#define BAUDRATE 115200 // Buadrate must match that of the ARDUINO

class testApp : public ofBaseApp{


public:

    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
```

```
void windowResized(int w, int h);
void dragEvent(ofDragInfo dragInfo);
void gotMessage(ofMessage msg);

void exit();
void guiEvent(ofxUIEventArgs &e);

// Color Tracking
ofVideoGrabber          vidGrabber;
    ofVideoGrabber          cam2;
    ofImage                 staticImage;

    ofxCvColorImage             colorImg;
    ofxCvColorImage             colorImgHSV;

    ofxCvGrayscaleImage      hueImg;
    ofxCvGrayscaleImage      satImg;
    ofxCvGrayscaleImage      valImg;
    ofxCvGrayscaleImage     filteredImg;

    ofxCvContourFinder contourFinder;

    ofImage                 calder;

    unsigned char *         colorTrackedPixels;
    ofTexture               trackedTexture;

    int minHue, maxHue, minSat, maxSat, minVal, maxVal;
    float hue, hueWidth, sat, satWidth, val, valWidth;

protected:

    colorTracking CT;

private:

    bool                videoAsInputSource;


// Pan/Tilt Control
float kitex;
float kitey;
float errorx;
float errory;
float centerx;
float centery;
float minErrorX;
float minErrorY;
float panVal;
float tiltVal;
bool trackKite;
```

```
// GridTracking

GridTracking grid;

ofVec2f p1;
ofVec2f p2;
ofVec2f heading;

// Color Tracking

ofVec2f min1left;
ofVec2f min2left;
ofVec2f min1right;
ofVec2f min2right;
ofVec2f max1left;
ofVec2f max2left;
ofVec2f max1right;
ofVec2f max2right;
ofVec2f leftMaxBound;
ofVec2f leftMinBound;
ofVec2f rightMaxBound;
ofVec2f rightMinBound;
ofVec2f leftPath1;
ofVec2f leftPath2;
ofVec2f rightPath1;
ofVec2f rightPath2;
ofVec2f leftPath;
ofVec2f rightPath;

// Video input

ofVideoGrabber vid;

float vidx,vidy;
float error;

bool flyingLeft;
bool flyingRight;
bool turningAroundLeft;
bool turningAroundRight;
bool drawControlData;

ofImage controlbg;



// ofxUI stuff //

ofxUICanvas *gui1;
ofxUICanvas *gui2;
ofxUILabel *serialSent;
```

```
ofxUILabel *serialReceived;
ofxUITextInput *serialTxtInput;
ofxUILabel *leftMotorPos;
ofxUILabel *rightMotorPos;
ofxUIButton *quit;
ofxUISlider *powerSlider;
ofxUIButton *rightMotorIn;
ofxUIButton *rightMotorOut;
ofxUIButton *leftMotorIn;
ofxUIButton *leftMotorOut;
ofxUIButton *bothMotorIn;
ofxUIButton *bothMotorOut;
ofxUILabel *localIP;
ofxUITextInput *hostIP;
ofxUILabelButton* saveButton;
ofxUILabelButton* updateKButton;
ofxUIDropDownList* pidList;
ofxUITextInput* KpInput;
ofxUITextInput* KiInput;
ofxUITextInput* KdInput;
ofxUIDropDownList* serialMenu;
ofxUILabel* serialStatus;
ofxUISlider* tiltSlider;
ofxUISlider* panSlider;

float gui1_w;

int red, green, blue;
int leftMotorInID,leftMotorOutID,rightMotorInID,rightMotorOutID;
int turnValue;

ofxSimpleSerial serial;
string serialPort;

ofxOscSender sender;
ofxOscReceiver receiver;
string hostAddress;

float Kp,Ki,Kd;
float xpos;
float ypos;

float powerMeter_h;
float powerMeter_w;
float powerMeter_y;
float powerMeter_x;

ofImage bg;
ofImage logo;
ofImage kite;

ofQuaternion iphoneQuat;
```

```
// Define Variables //

string depower;
string power;
string left;
string right;
string serialInfo;
string xval;
string yval;
string zval;
string serialCommand;
string portName;
string msg_string;

unsigned char incomingByte;

float theta;
int motorSpeed;
float currentPos;
float lastPos;
int turnRange;

float pitch;
float roll;
float yaw;
float currentHeading;
float lastHeading;

int lastRollAngle;
int numBytes;
int powerLevel;
int low;
int med;
int maximum;
int key;
int i;
vector<float> posAvg;

int countCycles;
int maxCycles;

bool pitchFlip;
bool turn;
bool printByte;
bool holdMotorSpeed;
bool holdPosition;
bool holdPower;
bool powerON;
bool depowerON;
  bool moveright;
  bool moveleft;
bool speedUp;
bool slowDown;
```

```
    bool Bbutton;
    bool arduinoReady;
    bool bautoPilot;
    bool bcalibrateAngle;
    bool connectedToOscHost;

    bool keyDown;
    bool home;

    bool serialCheck;

    void onNewMessage(string & message);

    void writeToArduino(string command);

    string floatToString(float value)
    {
        std::stringstream ss;
        ss << std::setprecision(2);
        ss << value;
        return ss.str();
    }

    float getAvg(vector<float> x);

};
```

testApp.cpp

```
#include "testApp.h"
#include <iostream>
#include <fstream>

//--------------------------------------------------------------
void testApp::setup()
{

    videoAsInputSource = true;

    //cout << cam2.listDevices() << endl;

    // Video grabber
    vidGrabber.setVerbose(true);
    vidGrabber.initGrabber(320,240);

    trackKite = false;
```

```
    staticImage.loadImage("image.png");
    staticImage.resize(320,240);

    // Allocations
    colorImg.allocate(staticImage.width, staticImage.height);
           // Static Image as imput source
    colorImg.allocate(vidGrabber.width, vidGrabber.height);
           // Live Cam as input source
    colorImgHSV.allocate(vidGrabber.width, vidGrabber.height);
           // HSV Image
    hueImg.allocate(vidGrabber.width, vidGrabber.height);
           // Hue Image
    satImg.allocate(vidGrabber.width, vidGrabber.height);
           // Saturation Image
    valImg.allocate(vidGrabber.width, vidGrabber.height);
           // value Image
    filteredImg.allocate(vidGrabber.width, vidGrabber.height);

    colorTrackedPixels = new unsigned char [vidGrabber.width *
vidGrabber.height];   // Tracked Image
    trackedTexture.allocate(vidGrabber.width, vidGrabber.height,
GL_LUMINANCE);

    // HSV variables. by default track red color.
    hue = 0.68;
    hueWidth = 0.02;
    sat = 0.74;
    satWidth = 0.68;
    val = 0.58;
    valWidth = 0.52;

    //ofSetDataPathRoot("../Resources/");

    ofBackground(255, 255, 255);

    serialCheck = false;
    //serialCheck = serial.setup();       // Opens serial port at
115200 baudrate by default- returns false if failed to open serial
    //serialCheck = serial.setup(0, 115200);  // use when arduino is
connected directly to computer

    if(serialCheck)
    {
        serial.startContinuesRead(false);
        ofAddListener(serial.NEW_MESSAGE,this,&testApp::onNewMessage);
    }
```

```
serial.flush();                         // flush serial port buffer
ofSetFrameRate(30);                     // set Frame Rate
ofSetVerticalSync(TRUE);                // Enable Verticle sync

if(serialCheck)
{
    serial.writeByte('x');
    serial.writeByte('/');
}

red = 233;
blue = 162.73;
green = 121.63;

serial.listDevices();
serialCommand = "n/a";
incomingByte = 0;

ofBackground(100);

// Initialize OSC connection
connectedToOscHost = false;
receiver.setup( INPORT );
ofxOscMessage m;

// Initialize Vidgrabber
//vid.initGrabber(320, 240);

// Load images from ../data folder
bg.loadImage("grey.jpg");
logo.loadImage("kite.gif");
kite.loadImage("naishKite.png");
kite.resize(400, 400);
controlbg.loadImage("controlbg.png");

//
Kp = Ki = Kd = 0;

//-----------------

// Initialize GUI stuff //

gui1_w = 300;

serialSent = new ofxUILabel("n/a", OFX_UI_FONT_MEDIUM);
```

```
    serialReceived = new ofxUILabel(ofToString("n/a"),
OFX_UI_FONT_MEDIUM);
    powerSlider = new ofxUISlider("POWER", 0, 96, 96, 250, 20);
    powerSlider->setIncrement(8);

    gui1 = new ofxUICanvas(0,0,gui1_w,ofGetHeight());
    gui1->setPadding(20);
    gui2 = new ofxUICanvas(ofGetWidth()-gui1_w,0,gui1_w,
ofGetHeight()-20);

    rightMotorIn = new ofxUIButton("IN", false, 20, 20);
    rightMotorIn->setColorFill(ofColor(0,0,blue));
    rightMotorIn->setDrawFill(true);
    rightMotorOut = new ofxUIButton("OUT", false, 20, 20);
    rightMotorOut->setColorFill(ofColor(0,0,blue));
    rightMotorOut->setDrawFill(true);
    leftMotorIn = new ofxUIButton("IN", false, 20, 20);
    leftMotorIn->setColorFill(ofColor(0,0,blue));
    leftMotorOut = new ofxUIButton("OUT", false, 20, 20);
    leftMotorIn->setColorFill(ofColor(0,0,blue));
    bothMotorIn = new ofxUIButton("IN", false, 20, 20);
    bothMotorIn->setColorFill(ofColor(0,0,blue));
    bothMotorOut = new ofxUIButton("OUT", false, 20, 20);
    bothMotorIn->setColorFill(ofColor(0,0,blue));

    leftMotorPos = new ofxUILabel("", OFX_UI_FONT_MEDIUM);
    rightMotorPos = new ofxUILabel("", OFX_UI_FONT_MEDIUM);


    gui1->addWidgetDown(new ofxUILabel("KITE CONTROL ",
OFX_UI_FONT_LARGE));
    gui1->addWidgetDown(new ofxUISpacer(gui1_w-2*gui1->getPadding(),
2));
    gui1->loadSettings("GUI/guiSettings.xml");
    gui1->addWidgetDown(powerSlider);


    serialTxtInput = new ofxUITextInput("Serial Command", "", 100);
    serialTxtInput->setAutoClear(true);

    gui1->addWidgetDown(new ofxUISpacer(250,2));
    gui1->addWidgetDown(new ofxUILabel("SERIAL", OFX_UI_FONT_MEDIUM));

    vector <string> serialPorts;
    for (int j = 0; j < serial.getDeviceList().size(); j++)
    {
```

```
serialPorts.push_back(serial.getDeviceList().at(j).getDeviceName());
    }

    serialMenu = new ofxUIDropDownList(250, "Select Serial Port",
serialPorts, OFX_UI_FONT_MEDIUM);
    serialMenu->setAutoClose(true);

    gui1->addWidgetDown(serialMenu);

    gui1->addWidgetDown(new ofxUILabel("Serial Status Label", "Status:
", OFX_UI_FONT_MEDIUM));

    serialStatus = new ofxUILabel("SerialStatus", "n/a",
OFX_UI_FONT_MEDIUM);

    gui1->addWidgetRight(serialStatus);

    gui1->addWidgetDown(new ofxUILabel("Serial Sent:",
OFX_UI_FONT_MEDIUM));
    gui1->addWidgetRight(serialSent);
    gui1->addWidgetDown(new ofxUILabel("Serial Received:",
OFX_UI_FONT_MEDIUM));
    gui1->addWidgetRight(serialReceived);
    gui1->addWidgetDown(new ofxUILabel("Serial Command:",
OFX_UI_FONT_MEDIUM));
    gui1->addWidgetRight(serialTxtInput);

    gui1->addWidgetDown(new ofxUISpacer(250,2));
    gui1->addWidgetDown(new ofxUILabel("CALIBRATION",
OFX_UI_FONT_MEDIUM));
    gui1->addWidgetDown(new ofxUILabel("Left Motor:   ",
OFX_UI_FONT_MEDIUM));
    gui1->addWidgetRight(leftMotorIn);
    gui1->addWidgetRight(leftMotorOut);
    gui1->addWidgetDown(new ofxUILabel("Right Motor:  ",
OFX_UI_FONT_MEDIUM));
    gui1->addWidgetRight(rightMotorIn);
    gui1->addWidgetRight(rightMotorOut);

    localIP = new ofxUILabel("0.0.0.0",OFX_UI_FONT_MEDIUM);
    hostIP = new ofxUITextInput("HOST_IP", "0.0.0.0", 150);
    hostIP ->setAutoClear(false);

    gui1->addWidgetDown(new ofxUISpacer(250,2));
```

```
    gui1->addWidgetDown(new ofxUILabel("OSC Setup",
OFX_UI_FONT_MEDIUM));
    gui1->addWidgetDown(new ofxUILabel("Local IP: ",
OFX_UI_FONT_MEDIUM));
    gui1->addWidgetRight(localIP);
    gui1->addWidgetDown(new ofxUILabel("Host IP: ",
OFX_UI_FONT_MEDIUM));
    gui1->addWidgetRight(hostIP);

    //---------------------- gui 2 setup

    gui2->setPadding(20);

    gui2->addWidgetDown(new ofxUILabel("PID TUNING",
OFX_UI_FONT_MEDIUM));
    gui2->addWidgetDown(new ofxUISpacer(250,2));


    vector <string> pidItems;
    pidItems.push_back("Left Motor");
    pidItems.push_back("Right Motor");
    pidItems.push_back("Kite");
    pidList = new ofxUIDropDownList(250, "Select PID to Tune",
pidItems, OFX_UI_FONT_MEDIUM);
    pidList->setAutoClose(true);

    gui2->addWidgetDown(pidList);

    gui2->addWidgetDown(new ofxUILabel("Setpoint: ",
OFX_UI_FONT_MEDIUM));
    gui2->addWidgetDown(new ofxUILabel("Input: ",
OFX_UI_FONT_MEDIUM));
    gui2->addWidgetDown(new ofxUILabel("Output: ",
OFX_UI_FONT_MEDIUM));
    gui2->addWidgetDown(new ofxUILabel("Kp: ", OFX_UI_FONT_MEDIUM));
    KpInput = new ofxUITextInput("Kp", ofToString(Kp), 50);
    gui2->addWidgetRight(KpInput);

    //gui2->addWidgetRight(pidList);
    gui2->addWidgetDown(new ofxUILabel("Ki: ", OFX_UI_FONT_MEDIUM));

    KiInput = new ofxUITextInput("Ki", ofToString(Ki), 50);
    gui2->addWidgetRight(KiInput);
    gui2->addWidgetDown(new ofxUILabel("Kd: ", OFX_UI_FONT_MEDIUM));

    KdInput = new ofxUITextInput("Kd", ofToString(Kd), 50);
```

```
    gui2->addWidgetRight(KdInput);

    updateKButton = new ofxUILabelButton(250, false, "UPDATE K
VALUES");
    gui2->addWidgetDown(updateKButton);
    saveButton = new ofxUILabelButton(250, false, "SAVE");
    gui2->addWidgetDown(saveButton);

    panVal = 90;
    tiltVal = 90;

    tiltSlider = new ofxUISlider( "Tilt", 180, 0, 90, 40, 200);
    panSlider = new ofxUISlider( "Pan", 180, 0, 90, 200, 40);
    tiltSlider->setLabelPrecision(0);
    //panSlider->setLabelPrecision(0);

    gui2->addWidgetDown(new ofxUISpacer(250, 2));
    gui2->addWidgetDown(new ofxUILabel("Pan/Tilt Control",
OFX_UI_FONT_MEDIUM));

    gui2->addWidgetDown(tiltSlider);
    gui2->addWidgetRight(panSlider);


    ofAddListener(gui1->newGUIEvent, this, &testApp::guiEvent);
    ofAddListener(gui2->newGUIEvent, this, &testApp::guiEvent);

    leftMotorInID = leftMotorIn->getID();
    leftMotorOutID = leftMotorOut->getID();
    rightMotorInID = rightMotorIn->getID();
    rightMotorOutID = rightMotorOut->getID();

    //-----------------
    // motor Speed parameters
    low = 0;
    med = 1;
    maximum = 2;

    motorSpeed = 100;

    powerLevel = 96;
    turnValue = 0;

    incomingByte = 0;
    countCycles = 0;
    maxCycles = 3;
```

```
    posAvg.resize(maxCycles);

    //-----------------
    // initialiaze control Boolians

    pitchFlip = false;
    speedUp = false;
    slowDown = false;
    holdPower = false;
    holdPosition = false;
    holdMotorSpeed = false;
    keyDown = false;
    turn = false;
    Bbutton = false;
    arduinoReady = false;
    if(serialCheck) arduinoReady = true;
    printByte = false;
    bautoPilot = false;
    bcalibrateAngle = false;

    lastRollAngle = 0;

    i = 0;
    key = 0;

    //-----------------
    // Initialize angle of rotation
    theta = 0;
    numBytes = 0;


    //-----------------
    //setup grid

    grid.setup(5, 5, ofGetWidth() - 290, ofGetHeight()/2 + 60, 280,
220);

    p1.set(ofGetMouseX(), ofGetMouseY());

    //-----------------
    // setup control band vectors

    vidx = 320;
    vidy = 280;
```

```
    float x0 = ofGetWidth()/2 - vidx/2;
    float x1 = ofGetWidth()/2 + vidx/2;
    float y0 = ofGetHeight()/2 + vidy/2;
    float y1 = ofGetHeight()/2 - vidy/2;

    min1left.set(ofGetWidth()/2 - vidx/2, ofGetHeight()/2 + vidy/2 -
(1.0/6) * vidy);
    max1left.set(ofGetWidth()/2 - vidx/2,ofGetHeight()/2 + vidy/2 -
(2.0/6) * vidy);
    min2left.set(ofGetWidth()/2 + vidx/2,ofGetHeight()/2 + vidy/2 -
(4.0/6) * vidy);
    max2left.set(ofGetWidth()/2 + vidx/2, ofGetHeight()/2 + vidy/2 -
(5.0/6) * vidy);

    min1right.set(x1, y0 - (1.0/6) * vidy);
    max1right.set(x1, y0 - (2.0/6) * vidy);
    min2right.set(x0, y0 - (4.0/6) * vidy);
    max2right.set(x0, y0 - (5.0/6) * vidy);


    leftMaxBound = max2left - max1left;
    leftMinBound = min2left - min1left;

    leftPath1.set(min1left.x, (min1left.y + max1left.y)/2);
    leftPath2.set(min2left.x, (min2left.y + max2left.y)/2);
    rightPath1.set(min1right.x, (min1right.y + max1right.y)/2);
    rightPath2.set(min2right.x, (min2right.y + max2right.y)/2);

    leftPath = leftPath2 - leftPath1;
    rightPath = rightPath2 - rightPath1;

    flyingLeft = true;
    flyingRight = false;

    drawControlData = false;


    // Initialize pan/tilt variables
    centerx = vidGrabber.width/2;
    centery = vidGrabber.height/2;
    errorx = 0;
    errory = 0;
    minErrorX = 20;     // these will be set experimentally
    minErrorY = 20;     // these will be set experimentally
    panVal = 90;
    tiltVal = 90;
```

```
    turnRange = 20;


}

//--------------------------------------------------------------
void testApp::update()
{

    // Get next frame of video
    vidGrabber.grabFrame();


    if (vidGrabber.isFrameNew())
    {


        //Calculate min and max thersholds values
        minHue = max((hue - hueWidth*0.5) * 255, 0.0);
        maxHue = min((hue + hueWidth*0.5) * 255, 255.0);
        minSat = max((sat - satWidth*0.5) * 255, 0.0);
        maxSat = min((sat + satWidth*0.5) * 255, 255.0);
        minVal = max((val - valWidth*0.5) * 255, 0.0);
        maxVal = min((val + valWidth*0.5) * 255, 255.0);

        // Live video or Static image: input source
        if(videoAsInputSource)
        {
            colorImg.setFromPixels(vidGrabber.getPixels(),
vidGrabber.width, vidGrabber.height);
        }
        else
        {
            colorImg.setFromPixels(staticImage.getPixels(),
staticImage.width, staticImage.height);
        }

        // HSV
        colorImgHSV = colorImg;
        colorImgHSV.convertRgbToHsv();
        colorImgHSV.convertToGrayscalePlanarImages(hueImg, satImg,
valImg);

        // Perform tracking calculations
```

```
        unsigned char * huePixels = hueImg.getPixels();
        unsigned char * satPixels = satImg.getPixels();
        unsigned char * valPixels = valImg.getPixels();
        int nPixels = vidGrabber.width * vidGrabber.height;

        for (int i = 0; i < nPixels; i++)
        {
            if ((huePixels[i] >= minHue && huePixels[i] <= maxHue) &&
                    (satPixels[i] >= minSat && satPixels[i] <= maxSat)
&&
                    (valPixels[i] >= minVal && valPixels[i] <=
maxVal))
            {
                colorTrackedPixels[i] = 255;
            }
            else
            {
                colorTrackedPixels[i] = 0;
            }
        }
        trackedTexture.loadData(colorTrackedPixels, vidGrabber.width,
vidGrabber.height, GL_LUMINANCE);
        filteredImg.setFromPixels(colorTrackedPixels,  vidGrabber.width,
vidGrabber.height);

        // ******Laser
        CT.calcColorRange(hue, hueWidth, sat, satWidth, val);


        contourFinder.findContours(filteredImg, 200, (340*240)/4, 4,
false, true);
    }

    // Check turn range is less than 30
    if(turnRange > 30)
    {
        turnRange = 30;
    }

    //----------------------
    // update gui stuff here
    gui1->update();
    saveButton->setColorFill(ofColor(255,69,0));    // orange
    saveButton->setDrawFill(true);
    leftMotorIn->setColorFill(ofColor(100,149,237));
    leftMotorIn->setDrawFill(true);
```

```
    leftMotorOut->setColorFill(ofColor(100,149,237));
    leftMotorOut->setDrawFill(true);
    rightMotorIn->setColorFill(ofColor(100,149,237));
    rightMotorIn->setDrawFill(true);
    rightMotorOut->setColorFill(ofColor(100,149,237));
    rightMotorOut->setDrawFill(true);
    updateKButton->setColorFill(ofColor(100,149,237));
    updateKButton->setDrawFill(true);
    powerSlider->setColorFill(ofColor(255,165,0));
    pidList->setDrawFill(true);
    pidList->setColorFill(ofColor(100,149,237));
    if(pidList->isOpen())
    {
        for(int j = 0; j < pidList->getToggles().size(); j++)
        {
            pidList->getToggles().at(j)->setDrawFill(true);
            pidList->getToggles().at(j)-
>setColorFill(ofColor(139,137,137));
        }
    }
    serialMenu->setDrawFill(true);
    serialMenu->setColorFill(ofColor(100,149,237));
    if(serialMenu->isOpen())
    {
        for(int j = 0; j < serialMenu->getToggles().size(); j++)
        {
            serialMenu->getToggles().at(j)->setDrawFill(true);
            serialMenu->getToggles().at(j)-
>setColorFill(ofColor(139,137,137));
        }
    }

    if(powerLevel > 96) powerLevel = 96;
    else if(powerLevel < 0) powerLevel = 0;

    if(serialCommand == "L/") bautoPilot = true;
    else if(serialCommand == "l/") bautoPilot = false;

    serialSent->setLabel(serialCommand);
    powerSlider->setValue(powerLevel);

    if(serialCheck)
    {
        serialStatus->setLabel("OPEN");
        serialStatus->setColorOutline(ofColor(0,255,0));
    }
```

```
else
{
    serialStatus->setLabel("Failed to open...");
    serialStatus->setColorOutline(ofColor(255,0,0));
}

tiltSlider->setColorFill(ofColor(100,149,237));
panSlider->setColorFill(ofColor(100,149,237));



// check for waiting OSC messages
while( receiver.hasWaitingMessages() )
{
    // get the next message
    ofxOscMessage m;
    receiver.getNextMessage( &m );

    // get host address on first time through
    if(!connectedToOscHost)
    {
        hostAddress = m.getRemoteIp();
        sender.setup(hostAddress, OUTPORT);
        hostIP->setTextString(hostAddress);
        connectedToOscHost = true;
    }


    // get arguments
    for ( int i=0; i<m.getNumArgs(); i++ )
    {
        // display the argument - make sure we get the right type
        if( m.getArgType( i ) == OFXOSC_TYPE_INT32 )
            msg_string += ofToString( m.getArgAsInt32( i ) );
        else if( m.getArgType( i ) == OFXOSC_TYPE_FLOAT )
            msg_string += ofToString( m.getArgAsFloat( i ) );
        else if( m.getArgType( i ) == OFXOSC_TYPE_STRING )
            msg_string += m.getArgAsString( i );
    }

    //cout << m.getAddress();
    //cout << "    " << msg_string << endl;

    if(m.getAddress() == "/setupIP")
    {
        localIP->setLabel(m.getRemoteIp());
```

```
        }

        if (m.getAddress() == "/gyrosc/gps")
        {
            cout.precision(4);
            cout << "Lat: " << m.getArgAsFloat(0) << " Long: " <<
m.getArgAsFloat(1) << endl;
        }

        if(m.getAddress() == "/gyrosc/gyro")
        {
            //cout << "pitch: " << m.getArgAsFloat(0) ;
            pitch = m.getArgAsFloat(0) / 1.5;
            //cout << "   roll: " << m.getArgAsFloat(1) << endl;;


            //cout << "   yaw: " << m.getArgAsFloat(2) << endl;
            yaw = m.getArgAsFloat(2);
        }
        if(m.getAddress() == "/gyrosc/comp")
        {
            currentHeading = ofToFloat(msg_string);
            if(currentHeading-lastHeading > 60)
            {
                pitchFlip = !pitchFlip;
            }

            cout << "   HEADING:" << msg_string << endl;

            lastHeading = currentHeading;

        }

        // iPhone Quaterion Data
        if(m.getAddress() == "/gyrosc/quat")
        {
            iphoneQuat.set(m.getArgAsFloat(1), -m.getArgAsFloat(2),
m.getArgAsFloat(3), m.getArgAsFloat(0));

        }


        if(m.getAddress() == "/FlightControl/Auto")
        {
            if (msg_string == "1")
            {
```

```
                writeToArduino("L/");   // turn ON autopilot
                bautoPilot = true;
            }
            else
            {
                writeToArduino("l/");   // turn OFF autopilot
                bautoPilot = false;

            }

        }

        if(m.getAddress() == "/FlightControl/Steering" && !bautoPilot)
        {
            currentPos = ofToFloat(msg_string);
            if(int(currentPos/0.1) != int(lastPos/0.1))
            {

                if(currentPos != 0)
                {
                    writeToArduino("t" +
ofToString(int(currentPos*turnRange)) + "/");
                }
                else
                {
                    writeToArduino("t0/");
                }

                lastPos = currentPos;

            }
        }
        else if(m.getAddress() == "/FlightControl/Power"
&& !bautoPilot)
        {
            ypos = ofGetHeight() - (1 - ofToFloat(msg_string))*
ofGetHeight();
            //cout << "Ypos: " << ypos << endl;
            int power = 96 * ofToFloat(msg_string);

            // only send power values that are divisible by 6
            if(power % 6 == 0)
            {
                writeToArduino("p" + ofToString(power) + "/");
            }
        }
```

```
        if(m.getAddress() == "/Calibrate/leftIN" && msg_string == "1")
        {
            powerLevel = 96;
            writeToArduino("b/");
            //cout << "b/" << endl;
        }
        else if(m.getAddress() == "/Calibrate/leftIN" && msg_string ==
"0")
        {
            powerLevel = 96;
            writeToArduino("S/");
            //cout << "b/" << endl;

        }

        if(m.getAddress() == "/Calibrate/leftOUT" && msg_string ==
"1")
        {
            powerLevel = 96;
            writeToArduino("a/");
            //cout << "a/" << endl;
        }
        else if (m.getAddress() == "/Calibrate/leftOUT" && msg_string
== "0")
        {
            powerLevel = 96;
            writeToArduino("S/");
            //cout << "b/" << endl;

        }

        if(m.getAddress() == "/Calibrate/rightIN" && msg_string ==
"1")
        {
            powerLevel = 96;
            writeToArduino("c/");
            //cout << "c/" << endl;

        }
        else if(m.getAddress() == "/Calibrate/rightIN" && msg_string
== "0")
        {
            powerLevel = 96;
```

```
            writeToArduino("S/");
            //cout << "b/" << endl;

        }

        if(m.getAddress() == "/Calibrate/rightOUT" && msg_string ==
"1")
        {
            powerLevel = 96;
            writeToArduino("d/");
            //cout << "d/" << endl;

        }
        else if(m.getAddress() == "/Calibrate/rightOUT" && msg_string
== "0" )
        {
            powerLevel = 96;
            writeToArduino("S/");
            //cout << "b/" << endl;

        }

        if(m.getAddress() == "/Calibrate/bothIN" && msg_string == "1")
        {
            powerLevel = 96;
            // Write to arduino a serial command to bring both motors
in equal amounts
            //cout << "e/" << endl;

        }
        if(m.getAddress() == "/Calibrate/bothOUT" && msg_string ==
"1")
        {
            powerLevel = 96;
            // Write to arduino to let both motors out equal amounts
            //cout << "f/" << endl;

        }
        if(m.getAddress() == "/Calibrate/fader1")
        {

            if(int(m.getArgAsFloat(0)) != turnRange)
            {
                turnRange = int(m.getArgAsFloat(0));
                m.clear();
                m.setAddress("/Calibrate/MaxRangeLabel");
```

```
                m.addFloatArg(turnRange);
                sender.sendMessage(m);
            }
        }

        if (m.getAddress() == "/FlightControl")
        {
            m.setAddress("/FlightControl/Steering"); // put steering
slider back to 0 (center)
            m.addFloatArg(0);
            sender.sendMessage(m);
            m.clear();                              // clear osc
message
            m.setAddress("/FlightControl/Power");   // put power
slider back to 1 (bottom)
            m.addFloatArg(1);
            sender.sendMessage(m);
        }

        if(m.getAddress() == "/PanTilt/Pan" &&
int(m.getArgAsFloat(0))%2 == 0)
        {
            // pad to desired position
            writeToArduino("P" + ofToString(int(m.getArgAsFloat(0)))
+"/");

            // update pan slider
            panSlider->setValue(m.getArgAsFloat(0));

        }
        if(m.getAddress() == "/PanTilt/Tilt" &&
int(m.getArgAsFloat(0))%2 == 0)
        {
            // tilt to desired position
            writeToArduino("T" + ofToString(int(m.getArgAsFloat(0)))
+"/");

            // update tilt slider
            tiltSlider->setValue(m.getArgAsFloat(0));

        }

        msg_string = "";
    }

    // update grid tracker
```

```
    grid.track(ofGetMouseX(), ofGetMouseY());

    // update every second frame
    if(ofGetFrameNum() % 2 == 0)
    {
        // get new position
        p2.set(float(ofGetMouseX()), float(ofGetMouseY()));

        // Check if new position is differnt than last positon
        if(p1 != p2)
        {

            // calulate the vector between the new position p2 and
last posiiton p1
            heading = p2 - p1;
            // set the value of p1 to value of p2
            p1.set(p2.x, p2.y);
        }
    }

    if(drawControlData)
    {

        if(p1.x < ofGetWidth()/2 -vidx/2   && flyingLeft)
        {
            turningAroundLeft = true;
            turningAroundRight = false;
            leftPath2.y = (min2left.y + max2left.y)/2;

        }
        else if(p1.x > ofGetWidth()/2 + vidx/2 && flyingRight)
        {
            turningAroundLeft = false;
            turningAroundRight = true;
            rightPath2.y = (min2right.y + max2right.y)/2;

        }
        else if(turningAroundLeft)
        {
            flyingRight = true;
            flyingLeft = false;
            turningAroundLeft = false;
            rightPath2.y = p1.y;
            rightPath = rightPath2 - rightPath1;

        }
```

```
        else if(turningAroundRight)
        {
            flyingLeft = true;
            flyingRight = false;
            turningAroundRight = false;
            leftPath2.y = p1.y;
            leftPath = leftPath2 - leftPath1;
        }
    }

    int biggestBlob = 0;
    if(contourFinder.blobs.size() > 2)
    {
        for(int j=1; j < contourFinder.blobs.size(); j++)
        {
            if(contourFinder.blobs.at(j).area >
contourFinder.blobs.at(j-1).area)
            {
                biggestBlob = j;
            }
        }
    }

    if(contourFinder.blobs.size() > 0)
    {
        kitex = contourFinder.blobs.at(biggestBlob).centroid.x;
        kitey = contourFinder.blobs.at(biggestBlob).centroid.y;
    }

    if(trackKite)
    {
        // Update kitex and kitey here


        // calculate errorx and errory
        errorx = kitex - centerx;
        errory = kitey - centery;

        // check if error is bigger than minimum
        if( abs(errorx) > minErrorX )
        {
            int stepSize;
            stepSize = int( (abs(float(errorx)) /
(vidGrabber.width/2)) * 5 );
            if(stepSize == 0) stepSize = 1;
            // tell arduino to pan camera to minimize error
```

```
            if(errorx > 0)
            {
                // pan right
                panVal = panVal - stepSize;
                if(panVal > 180) panVal = 180;
                if(panVal < 0) panVal = 0;
                writeToArduino("P" + ofToString(panVal) + "/");

            }
            else
            {
                // pan left
                panVal = panVal + stepSize;
                if(panVal > 180) panVal = 180;
                if(panVal < 0) panVal = 0;
                writeToArduino("P" + ofToString(panVal) + "/");
            }
        }
        if( abs(errory) > minErrorY)
        {
            int stepSize;
            stepSize = int( (abs(float(errory)) /
(vidGrabber.height/2)) * 5 );
            if(stepSize == 0) stepSize = 1;

            // tell arduino to tilt camera to minimize error
            if(errory > 0)
            {
                // tilt down
                tiltVal = tiltVal + stepSize;
                if(tiltVal > 180) tiltVal = 180;
                if(tiltVal < 0) tiltVal = 0;
                writeToArduino("T" + ofToString(tiltVal) + "/");
            }
            else
            {
                // tilt up
                tiltVal = tiltVal - stepSize;
                if(tiltVal > 180) tiltVal = 180;
                if(tiltVal < 0) tiltVal = 0;
                writeToArduino("T" + ofToString(tiltVal) + "/");

            }
        }
    }
```

```
}

//--------------------------------------------------------------
void testApp::draw()
{

    ofEnableAlphaBlending();

    // draw background
    ofSetColor(255);
    bg.draw(0,0,ofGetWidth(),ofGetHeight());

    // Draw kite logo
    ofSetColor(255);
    logo.draw(gui1_w-170,gui1->getRect()->height-170,130,150);

    // draw rotated line to indicate rotation of WiiMote/Joystick
    ofSetColor(255,255,255);
    ofSetColor(0, 0, 255);

    ofVec3f yaxis(0,-1,0);
    ofVec3f xaxis(1,0,0);
    ofVec3f zaxis(0,0,1);
    ofVec3f iphoneYaxis(0,-1,0);
    ofVec3f iphoneXaxis(1,0,0);


    ofPushMatrix();
    ofTranslate(ofGetWidth()/2,ofGetHeight()/2);

    ofVec3f axis;
    float  angle;
    float theta,phi;
    float calAngle;
    float x,y,z;

    if(bcalibrateAngle)
    {
        // get curent vector
        iphoneQuat.getRotate(angle, axis);
        axis = -axis;
        // compare it to our reference vector - the negative y axis
        theta = axis.angle(yaxis);
        phi = ofVec2f(axis.x,axis.z).angle(zaxis);

        // rotate about required axes to get to reference vector
```

```
    if(axis.x >0)
    {
        ofRotateY(-phi);
    }
    else
    {
        ofRotateY(phi);
    }

    // store that value to compensate for further calcs

}


iphoneQuat.getRotate(angle, axis);
ofRotate(angle,-axis.x, -axis.y, -axis.z);

iphoneYaxis.rotate(angle, -axis);
iphoneXaxis.rotate(angle, -axis);


float rollAngle;
rollAngle =  iphoneYaxis.angle(yaxis) ;
//cout << "Actual Angle:  " << int(rollAngle) << "  ";

// Quadrant 4
if(iphoneYaxis.y > 0 && iphoneYaxis.x > 0)
{
    rollAngle = (iphoneYaxis.angle(yaxis) - 90)  ;
}
// Quadrant 3
if(iphoneYaxis.y > 0 && iphoneYaxis.x < 0)
{
    rollAngle = 180 - rollAngle + 90;
}
//Quadrant 1
if(iphoneYaxis.y < 0 && iphoneYaxis.x > 0)
{
    rollAngle = - ( 90 - iphoneYaxis.angle(yaxis) )  ;
}
// Quadrant 2
if(iphoneYaxis.y < 0 && iphoneYaxis.x < 0)
{
    rollAngle = - (90 +iphoneYaxis.angle(yaxis))  ;
}
```

```
    if(bautoPilot && int(rollAngle)%5 == 0 && lastRollAngle !=
int(rollAngle))
    {
        writeToArduino("r" + ofToString(int(rollAngle)) + "/");
    }

    ofSetColor(255,255,255);
    //kite.draw(-kite.width/2 , -kite.height/2);

    ofPopMatrix();


    if(bautoPilot)
    {
        ofSetColor(255, 0, 0);
        ofDrawBitmapString("AUTOPILOT ENABLED", ofGetWidth()/2-60,
40);
    }

    //--------------------------------
    // Draw color tracking algorithm data here

    // Draw webcam
    //vid.draw(ofGetWidth()/2 -vid.width/2,ofGetHeight()/2-
vid.height/2, vid.width, vid.height);


    // Draw background for control lines
    ofSetColor(255);
    //controlbg.draw(ofGetWidth()/2-vidx/2 - 80, ofGetHeight()/2-
vidy/2);

    ofNoFill();
    //ofRect(ofGetWidth()/2-vidx/2, ofGetHeight()/2-vidy/2,
vidx,vidy);
    //ofRect(ofGetWidth()/2-vidx/2 - 40, ofGetHeight()/2-vidy/2, vidx
+ 80,vidy);

    // Draw boundary lines
    ofSetColor(255, 0, 0); // Red
    //ofLine(min1left.x, min1left.y, min2left.x, min2left.y);
    //ofLine(max1left.x, max1left.y, max2left.x, max2left.y);
    //ofLine(min1right.x, min1right.y, min2right.x, min2right.y);
    //ofLine(max1right, max2right);
```

```
    if(drawControlData)
    {

        // Draw path lines
        ofSetColor(0, 255, 0);  // Green
        ofLine(leftPath1, leftPath2);
        ofLine(rightPath1, rightPath2);

        // get vector to current mouse position
        ofVec2f absPos(ofGetMouseX(), ofGetMouseY());

        ofVec2f transPos;

        if (flyingLeft)
        {
            transPos = absPos - leftPath1;
        }
        else if( flyingRight)
        {
            transPos = absPos - rightPath1;
        }


        // get projection vector onto leftMinBound

        ofVec2f leftProj = leftPath.normalize() *
transPos.dot(leftPath);
        ofVec2f rightProj = rightPath.normalize() *
transPos.dot(rightPath);

        // get vector between projection and transposed position
        ofVec2f errorVec;

        if(flyingLeft)
        {
            errorVec = leftProj - transPos;
            error = errorVec.length();
            if(errorVec.angle(leftPath) > 0)
            {
                error = - error;
            }
        }
        else if(flyingRight)
```

```
        {
            errorVec = rightProj - transPos;
            error = errorVec.length();
            if(errorVec.angle(rightPath) > 0)
            {
                error = - error;
            }
        }

        // draw current positoin
        //ofLine(min1left, min1left + transPos);

        // draw projection
        ofSetColor(255);    // White
        //ofLine(min1left, min1left + proj);

        // draw error
        ofSetColor(0, 0, 255);  // Blue
        if(flyingLeft && !turningAroundLeft)
        {
            ofLine(leftPath1 + transPos, leftPath1 + transPos +
errorVec);
        }
        else if(flyingRight && !turningAroundRight)
        {
            ofLine(rightPath1 + transPos, rightPath1 + transPos +
errorVec);
        }

        // Draw error Info
        ofSetColor(255);

        ofDrawBitmapString("Error: " + ofToString(error),
ofGetWidth()/2 - 60, ofGetHeight()/2 + vidy/2 + 40);

        //ofDrawBitmapString("Angle: " +
ofToString(leftMinBound.angle(yaxis)), ofGetWidth()/2 - 60,
ofGetHeight()/2 + vidy/2 + 80);

        ofDrawBitmapString("Range: 21.35", ofGetWidth()/2 - 60,
ofGetHeight()/2 + vidy/2 + 60);

    }

    ofSetColor(255);
```

```
    // Draw Input source
    colorImg.draw(ofGetWidth()/2 - vidGrabber.width/2, ofGetHeight()/2
- vidGrabber.height -60);

    //contourFinder.draw(ofGetWidth()/2 - vidGrabber.width/2,
ofGetHeight()/2 - vidGrabber.height  - 20);

    filteredImg.draw(ofGetWidth()/2 - vidGrabber.width/2,
ofGetHeight()/2 - 20);

    // Draw center of frame rectangle
    ofSetColor(255,0,0);
    ofRect(ofGetWidth()/2 - minErrorX, ofGetHeight()/2 -
vidGrabber.height/2 - 60 - minErrorY, minErrorX*2, minErrorY*2);

    // Draw centroid of detected object
    ofSetColor(0,255,0);
    ofCircle(ofGetWidth()/2 - vidGrabber.width/2 + kitex,
ofGetHeight()/2 - vidGrabber.height - 60 + kitey, 5);
    ofDrawBitmapString("Centroid", ofGetWidth()/2 - vidGrabber.width/2
+ kitex + 9, ofGetHeight()/2 - vidGrabber.height - 60 + kitey);

    // Display control tracking control variables
    ofSetColor(255);
    ofDrawBitmapString("hue: " + ofToString(hue) + "('a', 'z')",
ofGetWidth()/2 - vidGrabber.width/2, ofGetHeight()/2 +
filteredImg.height + 5);
    ofDrawBitmapString("hue width: " + ofToString(hueWidth) + "('s',
'x')",ofGetWidth()/2 - vidGrabber.width/2, ofGetHeight()/2 +
filteredImg.height + 20);
    ofDrawBitmapString("saturation: " + ofToString(sat) + "('d', 'c')",
ofGetWidth()/2 - vidGrabber.width/2, ofGetHeight()/2 +
filteredImg.height + 35);
    ofDrawBitmapString("saturation width: " + ofToString(satWidth) +
"('f', 'v')", ofGetWidth()/2 - vidGrabber.width/2, ofGetHeight()/2 +
filteredImg.height + 50);
    ofDrawBitmapString("value: " + ofToString(val) + "('g', 'b')",
ofGetWidth()/2 - vidGrabber.width/2, ofGetHeight()/2 +
filteredImg.height + 65);
    ofDrawBitmapString("value width: " + ofToString(valWidth) + "('h',
'n')",ofGetWidth()/2 - vidGrabber.width/2, ofGetHeight()/2 +
filteredImg.height + 80);

}

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

```
void testApp::keyPressed(int key)
{

    this->key = key;

    if(key == 't')
    {
        trackKite = !trackKite;
    }

    if(key == OF_KEY_RETURN)
    {

        // Toggle draw conrol data
    }

    if(key == OF_KEY_UP)
    {
        if(!trackKite)
        {
            tiltVal --;
            tiltSlider->setValue(tiltVal);
            writeToArduino("T" + ofToString(tiltVal) + "/" );
            //grid.setNumRows(grid.getNumRows() + 1);
        }else{
            minErrorY += 5;
        }



    }
    if(key == OF_KEY_DOWN)
    {
        if(!trackKite)
        {
            tiltVal ++;
            tiltSlider->setValue(tiltVal);
            writeToArduino("T" + ofToString(tiltVal) + "/" );
            //trackKite = !trackKite;
            //grid.setNumRows(grid.getNumRows() - 1);
        }else{
            minErrorY -= 5;
        }

    }
```

```
if(key == OF_KEY_RIGHT)
{
    if(!trackKite)
    {
        panVal --;
        panSlider->setValue(panVal);
        writeToArduino("P" + ofToString(panVal) + "/" );
        //flyingRight = true;
        //flyingLeft = false;
        //grid.setNumCol(grid.getNumCol() + 1);
    }else{
        minErrorX += 5;
    }
}
if(key == OF_KEY_LEFT)
{
    if(!trackKite)
    {
        panVal ++;
        panSlider->setValue(panVal);
        writeToArduino("P" + ofToString(panVal) + "/" );
        //flyingRight = false;
        //flyingLeft = true;
        //grid.setNumCol(grid.getNumCol() - 1);
    }else{
        minErrorX -= 5;
    }
}

switch(key)
{
case 'a':
    hue = min(hue + 0.02, 1.0);
    break;
case 'z':
    hue = max(hue - 0.02, 0.0);
    break;
case 's':
    hueWidth = min(hueWidth + 0.02, 1.0);
    break;
case 'x':
    hueWidth = max(hueWidth - 0.02, 0.0);
    break;
case 'd':
    sat = min(sat + 0.02, 1.0);
    break;
```

```
case 'c':
    sat = max(sat - 0.02, 0.0);
    break;
case 'f':
    satWidth = min(satWidth + 0.02, 1.0);
    break;
case 'v':
    satWidth = max(satWidth - 0.02, 0.0);
    break;
case 'g':
    val = min(val + 0.02, 1.0);
    break;
case 'b':
    val = max(val - 0.02, 0.0);
    break;
case 'h':
    valWidth = min(valWidth + 0.02, 1.0);
    break;
case 'n':
    valWidth = max(valWidth - 0.02, 0.0);
    break;
case 'i':
    if(videoAsInputSource)
    {
        videoAsInputSource = false;
    }
    else
    {
        videoAsInputSource = true;
    }
    break;
case 'p':
    vidGrabber.videoSettings();
    break;
case '1':               // kite color
    hue = 0.68;
    hueWidth = 0.02;
    sat = 0.74;
    satWidth = 0.68;
    val = 0.58;
    valWidth = 0.52;
    break;
case '2':               // blue ball
    hue = 0.54;
    hueWidth = 0.3;
    sat = 0.64;
```

```
        satWidth = 0.5;
        val = 0.56;
        valWidth = 0.68;
        break;
    }



}

//--------------------------------------------------------------
void testApp::keyReleased  (int key)
{

    this->key = 0;

}


//--------------------------------------------------------------

void testApp::exit()
{
    gui1->saveSettings("GUI/guiSettings.xml");
    delete gui1;
    delete gui2;

}
//--------------------------------------------------------------

void testApp::guiEvent(ofxUIEventArgs &e)
{
    if(e.widget->getName() == "BACKGROUND VALUE")
    {
        ofxUISlider *slider = (ofxUISlider *) e.widget;
        ofBackground(slider->getScaledValue());
    }
    if(e.widget->getName() == "FULLSCREEN")
    {
        ofxUIToggle *toggle = (ofxUIToggle *) e.widget;
        ofSetFullscreen(toggle->getValue());

    }
    if(e.widget->getName() == "QUIT")
    {
        ofxUIButton *quit = (ofxUIButton *) e.widget;
```

```
        if(quit->getValue())
        {
            exit();
            //ofExit();
        }
    }

    if(e.widget->getID() == serialMenu->getID())
    {
        if(serialMenu->getSelected().size() > 0)
        {

            if(serialMenu->getSelected().at(0)->getName() !=
serialPort)
            {

                serialPort = serialMenu->getSelected().at(0)-
>getName();
                serial.close();
                serialCheck = serial.setup(serialPort, 115200);
            }
        }


    }

    if(e.widget->getName() == "Serial Command")
    {

        if(serialTxtInput->getTriggerType() ==
OFX_UI_TEXTINPUT_ON_ENTER)
        {
            writeToArduino(serialTxtInput->getTextString());
            serialSent->setLabel(serialCommand);
        }
        if(serialCommand[0] == 'p')
        {
            powerLevel = int(ofToFloat(serialCommand.substr(1,2)));
        }
        if(serialCommand[0] == 't')
        {
            theta = ofToFloat(serialCommand.substr(1,3))/20 * 80;
            if(theta > 110) theta = 110;
            if(theta < -110) theta = - 110;
        }
    }
```

```
    if(e.widget->getName() == "HOST_IP")
    {
        if(hostIP->getTriggerType() == OFX_UI_TEXTINPUT_ON_ENTER)
        {
            sender.setup(hostIP->getTextString(), OUTPORT);
        }
        else if(hostIP->getTriggerType() ==
OFX_UI_TEXTINPUT_ON_UNFOCUS)
        {
            sender.setup(hostIP->getTextString(), OUTPORT);
        }
    }

    if(e.widget->getName() == "Kp")
    {
        ofxUITextInput *textinput = (ofxUITextInput *) e.widget;

        Kp = ofToFloat(textinput->getTextString());

        textinput = NULL;
        delete textinput;
    }

    if(e.widget->getName() == "Ki")
    {
        ofxUITextInput *textinput = (ofxUITextInput *) e.widget;

        Ki = ofToFloat(textinput->getTextString());

        textinput = NULL;
        delete textinput;
    }

    if(e.widget->getName() == "Kd")
    {
        ofxUITextInput *textinput = (ofxUITextInput *) e.widget;

        Kd = ofToFloat(textinput->getTextString());

        textinput = NULL;
        delete textinput;
    }

    if(e.widget->getName() == "UPDATE K VALUES")
    {
```

```
        if(pidList->getSelected().size() > 0)
        {
            if(pidList->getSelected().at(0)->getName() == "Kite")
            {
                writeToArduino("kk" + ofToString(Kp) + "," +
ofToString(Ki) + "," + ofToString(Kd) + "/");
            }
            else if(pidList->getSelected().at(0)->getName() == "Right
Motor")
            {
                writeToArduino("kr" + ofToString(Kp) + "," +
ofToString(Ki) + "," + ofToString(Kd) + "/");
            }
            else if(pidList->getSelected().at(0)->getName() == "Left
Motor")
            {
                writeToArduino("kl" + ofToString(Kp) + "," +
ofToString(Ki) + "," + ofToString(Kd) + "/");
            }
        }

    }

    if(e.widget->getID() == pidList->getID())
    {

        // Display saved PID values

        if(pidList->getSelected().size() > 0)
        {

            if(pidList->getSelected().at(0)->getName() == "Kite")
            {
                ifstream tempfile;
                string tempstr;
                int size;
                tempfile.open("pidKite_vals.txt");
                getline(tempfile, tempstr);
                Kp = ofToFloat(tempstr);
                getline(tempfile, tempstr);
                Ki = ofToFloat(tempstr);
                getline(tempfile, tempstr);
                Kd = ofToFloat(tempstr);
                tempfile.close();
```

```
            }
            else if(pidList->getSelected().at(0)->getName() == "Right
Motor")
            {
                ifstream tempfile;
                string tempstr;
                int size;
                tempfile.open("pidRight_vals.txt");
                getline(tempfile, tempstr);
                Kp = ofToFloat(tempstr);
                getline(tempfile, tempstr);
                Ki = ofToFloat(tempstr);
                getline(tempfile, tempstr);
                Kd = ofToFloat(tempstr);
                tempfile.close();

            }
            else if(pidList->getSelected().at(0)->getName() == "Left
Motor")
            {
                ifstream tempfile;
                string tempstr;
                int size;
                tempfile.open("pidLeft_vals.txt");
                getline(tempfile, tempstr);
                Kp = ofToFloat(tempstr);
                getline(tempfile, tempstr);
                Ki = ofToFloat(tempstr);
                getline(tempfile, tempstr);
                Kd = ofToFloat(tempstr);
                tempfile.close();
            }
        }

        KpInput->setTextString(ofToString(Kp));
        KiInput->setTextString(ofToString(Ki));
        KdInput->setTextString(ofToString(Kd));

    }

    if(e.widget->getName() == "SAVE")
    {

        if(pidList->getSelected().size() > 0)
        {
```

```
            if(pidList->getSelected().at(0)->getName() == "Kite")
            {
                ofstream tempFile;
                tempFile.open("pidKite_vals.txt");
                tempFile << Kp << "\n" << Ki << "\n" << Kd;
                tempFile.close();

            }
            else if(pidList->getSelected().at(0)->getName() == "Right
Motor")
            {
                ofstream tempFile;
                tempFile.open("pidRight_vals.txt");
                tempFile << Kp << "\n" << Ki << "\n" << Kd;
                tempFile.close();

            }
            else if(pidList->getSelected().at(0)->getName() == "Left
Motor")
            {
                ofstream tempFile;
                tempFile.open("pidLeft_vals.txt");
                tempFile << Kp << "\n" << Ki << "\n" << Kd;
                tempFile.close();
            }
        }
    }

    // send motor calibration commands to Arduino
    if(leftMotorIn->getValue())
    {
        powerLevel = 96;
        writeToArduino("b/");
    }
    if(leftMotorOut->getValue())
    {
        powerLevel = 96;
        writeToArduino("a/");
    }
    if(rightMotorIn->getValue())
    {
        powerLevel = 96;
        writeToArduino("c/");
    }
    if(rightMotorOut->getValue())
    {
```

```
        powerLevel = 96;
        writeToArduino("d/");
    }
    if(bothMotorIn->getValue())
    {
        powerLevel = 96;
        // Write to arduino a serial command to bring both motors in
equal amounts
    }
    if(bothMotorOut->getValue())
    {
        powerLevel = 96;
        // Write to arduino to let both motors out equal amounts
    }

    //cout << e.widget->getID() << endl;

    if(e.widget->getID() == leftMotorInID) if(!leftMotorIn-
>getValue()) writeToArduino("S/");
    if(e.widget->getID() == leftMotorOutID) if(!leftMotorOut-
>getValue()) writeToArduino("S/");
    if(e.widget->getID() == rightMotorInID) if(!rightMotorIn-
>getValue()) writeToArduino("S/");
    if(e.widget->getID() == rightMotorOutID) if(!rightMotorOut-
>getValue()) writeToArduino("S/");


    if(e.widget->getName() == "Pan")
    {
        writeToArduino("P" + ofToString(panSlider->getScaledValue()) +
"/");
        panVal = panSlider->getScaledValue();
    }

    if(e.widget->getName() == "Tilt" )
    {
        writeToArduino("T" + ofToString(tiltSlider->getScaledValue())
+ "/");
        tiltVal = tiltSlider->getScaledValue();
    }



}
//--------------------------------------------------------------
void testApp::mouseMoved(int x, int y )
```

```
{


}
//--------------------------------------------------------------
void testApp::mouseDragged(int x, int y, int button)
{
}

//--------------------------------------------------------------
void testApp::mousePressed(int x, int y, int button)
{

}
//--------------------------------------------------------------
void testApp::mouseReleased(int x, int y, int button)
{

}
//--------------------------------------------------------------
void testApp::windowResized(int w, int h)
{

}
//--------------------------------------------------------------
void testApp::gotMessage(ofMessage msg)
{

}
//--------------------------------------------------------------
void testApp::dragEvent(ofDragInfo dragInfo)
{

}

void testApp::writeToArduino(string command)
{

    serialCommand = command;
    if(serialCheck)
    {
        serial.flush();
        int t0 = ofGetElapsedTimeMillis();
        unsigned char serialChar[command.size()];

        for(int i=0; i < command.size(); i++)
```

```
        {
            serialChar[i] = command[i];
        }
        for(int i=0; i < command.size(); i++)
        {
            //cout << serialChar[i];
            serial.writeByte(serialChar[i]);
        }
        cout << endl;

        while(serial.available() <1)  // wait untill arduino is ready
to recieve more data
        {
            if(ofGetElapsedTimeMillis() - t0 > 100) break;
        }

        incomingByte = serial.readByte();  // Read data from serial
port
        //serial.close();
    }

    return;

}

float testApp::getAvg(vector<float> x)
{
    float sum = 0;
    for(int i=0; i<x.size(); i++)
    {
        sum += x.at(i);
    }
    return sum/float(x.size());
}

void testApp::onNewMessage(string & message)
{
    cout << "Serial message: " << message << "\n";
    serialReceived->setLabel(message);

}
```

main.cpp

```
#include "ofMain.h"
#include "testApp.h"
#include "ofAppGlutWindow.h"

//========================================================================
====
int main( ){

    ofAppGlutWindow window;
      ofSetupOpenGL(&window, 1200,700, OF_WINDOW); // <-------- setup
the GL context

      // this kicks off the running of my app
      // can be OF_WINDOW or OF_FULLSCREEN
      // pass in width and height too:
      ofRunApp( new testApp());

}
```

7.0 References

[1] Lorenzo Fagiano, Mario Milanese, and Dario Piga “High–altitude wind power generation Technical report”
[2] Miles L. Loyd “Crosswind Kite Power”, 1980, Journal of Energy
[3] Liquid Force 2008 Kite User Manual
[4] ENPH 253 - 2010 Lecture Slides “Shielding and Grounding”
[5] http://www.kitepower.eu/
- http://arduino.cc/
- http://www.kitepower.eu/technology/2-core-concepts/46-automatic-operation.html
- http://kiteenergy.blogspot.com/
- http://oceanrodeo.com/kiting/about-ocean-rodeo/contact_ocean_rodeo/
- http://www.swisskitepower.ch/collaborators.html
- http://www.jobyenergy.com/
- http://www.makanipower.com/
- http://www.energykitesystems.net/0/WPI/index.html
- http://www.kitesurfingschool.org/kite.htm
- http://www.kiteboardingevolution.com/kiteboarding-kite.html
- http://kitegen.com/
- http://www.brighton-webs.co.uk/energy/rayleigh_wind_2.htm
- http://www.cbo-eco.ca/en/index.cfm/guides/financial-support-science-and-technology/
- http://kiteenergy.blogspot.com/

Parts Sources:
www.digikey.com
www.robotshop.com
http://www.airtimeboardsports.com/