Rotan Rotary Rocket

Winter Quarter Senior Design Project Report

Andrew Wu

Gabriel Orpia

Oliver René

Table of Contents

C. Process
D. Results
E. Improvements

X. Next Quarter Deliverables?

XI. Conclusion

I. Abstract

With the privatization of space travel, several companies have been researching cost efficient and reusable methods to build and launch spacecraft. This report describes our efforts to provide an alternative method of achieving affordable space travel. The objective of this proposal is to design and implement a flight controller that is able to autonomously navigate and maintain stability during flight. We attempt to realize this goal by combining fluid dynamics, control theory and mechatronic design. Our strategy is to use rotary wings that spin with the fuselage about an axis, which allows the entire system to be gyroscopically stable. By using simple motors and sensors, we intend to prove that a small scaled version of our flight controller can be a feasible concept. If the prototype proves to be practical, then the design may be advantageous to implement at full scale.

II. Introduction

Multi-stage to orbit rockets see a loss in efficiency as they lose their boosters after launch. Single stage to orbit (SSTO) rockets are majorly inefficient, requiring a 94% fuel-to-mass ratio. Both have trade-offs that within efficiencies during and after launch. Recent milestones such as Blue Origin's single stage to lower orbit spacecraft and SpaceX's first stage landing prove that we are inching ever closer to low cost space travel. The Falcon 9 costed $54 million to build while it burned $200,000 worth of fuel. Despite these low fuel costs, alternative methods for space travel can further reduce this amount.

The Roton Rotary Rocket group aspires to innovate new ideas geared towards low cost access to orbit and feasibility of an SSTO spacecraft. We strive to prove the viability of a rotary winged aircraft that improves cost efficiency of space travel and reusability of stages. A significant deterrent to conventional space travel is the atmospheric density 60,000 feet above sea level. Our project, on the other hand, aims to manipulate the characteristics of the atmosphere. The objective is to build a flight controller capable of autonomously guiding a rotary winged vehicle. Sensors will be used to provide input to a feedback loop which dictates the output of the actuators. We will delve into applied fluid dynamics, feedback control theory, and overall mechatronic design to achieve this proof of concept.

III. Physics

Because our project resembles a rotary wing flight aircraft, each wing can be analyzed using linear airfoil theory, while being integrated with respect to the radius from the axis of rotation. As true with all aircraft, we must understand the lift, and drag forces generated as well as multiple moment balances about the vehicle. Lift and drag are always perpendicular and parallel to the angle of attack respectively.

A. Symbols and Terms

Air density	ρ
Mass	m
Inertia	I
Vertical velocity	$\dot{z}$
Wing pitch	θ
Tail pitch	δ
Wing length	r
Spin rate	Ω
Wing radius	b
Chord length	c
Lift	L
Drag	D
Thrust	T
Moment	M
Edge of the fuselage	w
Profile Drag	C_{DO}
Number of wings	N
Aspect Ratio	$AR = \frac{b}{c}$
Downwash airflow	$\frac{\varepsilon}{2} = \frac{\alpha}{AR+2}$
Coefficient of Lift	$C_{L\alpha} = \frac{2\pi AR}{AR+2}$
Angle of relative wind	$\varphi = tan^{-1}(\frac{\dot{z}}{\Omega r})$

Angle of attack $\alpha = \theta - \varphi$

Relative velocity $v = \sqrt{\Omega^2 r^2 + \dot{z}^2}$

B. Lift

For fixed wing flight the lift is linear and described as

$L = \frac{1}{2}\rho v^2 bc(C_{L\alpha} * \alpha)$

However because the wing is rotating and seeing different velocities and angles as the radius increases we are forced to integrate this equation to know the total lift of a wing. The lift of a single cross section of the wing is as follows

$l = \frac{1}{2}\rho v^2(r)c(C_{L\alpha} * \alpha(r))$

This is integrated from the wall of the fuselage(where the wing begins) to the tip of the wing as

$$L = \int_w^b \frac{1}{2}\rho v^2(r)c(C_{L\alpha} * \alpha(r))dr = \frac{1}{2}\rho c C_{L\alpha}\int_w^b v^2(r)\alpha(r)dr$$

However there is a subtly in this equation. Lift is perpendicular to the angle of attack, but we need the total force perpendicular to the flat plane of the Roton. This means multiplying the lift term by $sin(\alpha) = \frac{\Omega r}{v}$ and $cos(\alpha) = \frac{\dot{z}}{v}$ to get $L_{\perp}$ and $L_{\parallel}$ respectively. However with these new terms, we are able to eliminate a velocity term from the integral and simply multiply by the numerator of the trigonometric term. These two integrals given a set number of inputs results in a vector that gives us a magnitude and angle of this force on a single wing. Our resulting total lift is as follows

$$L_{total} = \frac{1}{2}\rho c C_{L\alpha}(\Omega\int_w^b v(r)\alpha(r)rdr + \dot{z}\int_w^b v(r)\alpha(r)dr)$$

Because of the nature of these equations we were unable to get a closed form solution, regardless of help from computing resources. Square roots and inverse tangent functions would not allow us to find a function. However, because of resources such as Matlab, we were able to numerically solve for the area under the curve whenever necessary, which yields the same results.

C. Drag

The same process had to be repeated for the linear equation for drag

$$D = \frac{1}{2}\rho bcv^2(C_{DO} + \frac{C_{L\alpha}^2 \alpha^2}{\pi AR})$$

After being adapted for a rotary wing, and split up into $D_{\perp}$ and $D_{\parallel}$ we resulted with

$$D_{total} = \frac{1}{2}\rho c[\Omega \int_w^b v(r)r(C_{DO} + \frac{C_{L\alpha}^2 \alpha^2(r)}{\pi AR})dr + \dot{z}\int_w^b v(r)(C_{DO} + \frac{C_{L\alpha}^2 \alpha^2(r)}{\pi AR})dr]$$

These terms got even more complicated as an inverse tangent function was squared. There was no way to find a closed form solution. Again with the power of Matlab, we were able to discretely calculate the net force for the entire wing, given inputs. This results in a general Lift to Drag ratio of 25. Typical airfoil ratios are between 20 and 100.

D. Tail Moment Balance

Because we want a small actuator system to control the pitch of our wing, and thus its angle of attack will iterate out. Because the tail needs to be relatively small, it only needs to be calculated at one radius. Thankfully we do not need to integrate any function and the straightforward equation works well for our purposes.

$$L_{tail} = \frac{1}{2}\rho v(\tfrac{3}{4}b)b_{tail}c_{tail}C_{L\alpha\, tail}(\alpha(\tfrac{3}{4}b) + \delta - \tfrac{\xi}{2}) \quad \frac{d\theta}{d\alpha} < 0$$

The lift of the tail creates a torque on the wing, based on the wing's angle of attack. The second term emphasizes that the wing will settle at a certain pitch and thus angle of attack. For this to happen the torque must be zero. However the distance, multiplied by the force, is not zero. All the multiplicative terms are not zeros so we are left adjusting the last term, which ultimately boils down to the angle of attack of the tail.

$$0 = \alpha(\tfrac{3}{4}b) + \delta - \tfrac{\xi}{2} = \alpha(\tfrac{3}{4}b) + \delta - \frac{\alpha(\frac{3}{4}b)}{AR+2}$$

Which simplifies to

$$\theta = \frac{\delta + \frac{AR-1}{AR+2}\varphi_0}{\frac{AR-1}{AR+2}}$$

Where φ_0 denotes the relative angle of air on the wing at the $\frac{3}{4}b$ point. This equation is huge because this maps out exactly where our theta throughout the wing with a change to δ which we change simply with one servo for one wing. The other large advantage is that it is completely linear.

E. System Moment Balance

Up to this point we have been blind to the thrust we have attached to our wing. This needs to create a thrust that will pull us around and create the rotational speed that we need to make any of the preceding equations work. These thrusts generate force mostly in the horizontal direction, but must be taken into account their angle, which pulls the wings into a rotary rotation. There is a small part of its vertical force that must be taken into account, but we will deal with that later. With the $L_{\parallel}$ and $D_{\parallel}$ both creating a force opposing our thrusters, we created a moment balance that would meet at a certain Ω and that would be the current spin rate for that thrust. To find these we had to multiply these lift and drag terms by a moment arm, which would also be integrated. The thrust moment arm could simply be the thrust output multiplied by the distance, which would also be $\frac{3}{4}b$.

$$M_{system} = N(cos(\theta)T\tfrac{3}{4}b - \tfrac{1}{2}\rho c(C_{L\alpha}\dot{z}\int_{w}^{b} v(r)\alpha(r)r\,dr + \Omega\int_{w}^{b} r^{2}v(r)(C_{DO} + \tfrac{C_{L\alpha}^{2}\alpha^{2}(r)}{\pi AR})\,dr))$$

F. Solved Values

Solving for necessary values to achieve good lift and to be within the scope of a 20 week project we chose the following values for our project. We played with these to physical attributes in order to find optimal flight times and force. We used SI units for all measurements.

$b = 0.5\ [m]$

$c = 0.1\ [m]$

$w = 0.1\ [m]$

$T_{max} = 1.5\ [N]$

$N = 4$

$b_{tail} = c$

$c_{tail} = \frac{c}{2}$

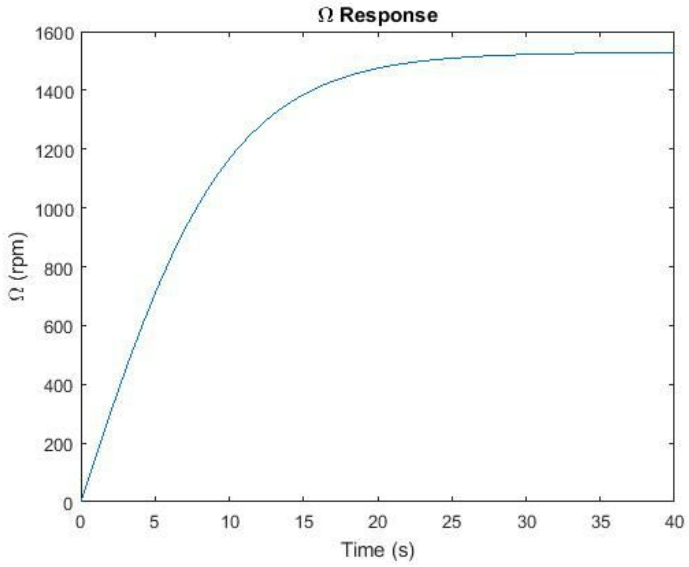
IV. Simulations

This section depends largely on the physics equations stated above. This time they must be in the form of an ordinary differential equation (ODE) which is time sensitive and must have initial conditions. Matlab's ODE45 function was especially helpful in these calculation, simultaneously calculating multiple coupled equations.

A. Ω Simulations

This seemed like the logical place to begin, because at the beginning of each flight, the motors will have to fire, creating a moment to spin, before we can create any lift. Thus we must leave our angle of attack at 0 for a long time, while we allow our motors to speed up, so as to create no lift and minimal drag. The initial condition would be zero as to simulate a true start up. The ODE solver evaluated the following time respective process over a series of points and yielded the following simulation. The lift and drag equations are there to emphasize that they are functions of Ω.

$$\frac{d\Omega}{dt} = \frac{1}{I}N(\frac{3}{4}b\ T\ cos(\theta) - L_{\parallel}(\Omega)D_{\parallel}(\Omega))$$

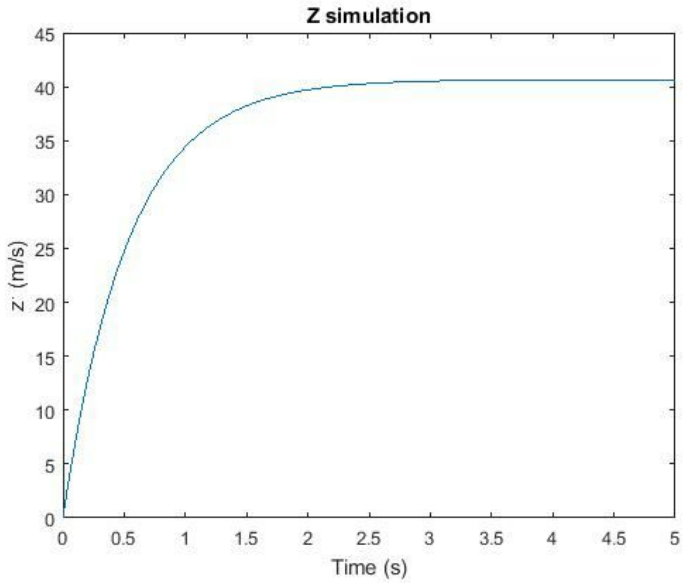


After any longer the simulation stayed at roughly 1528.2 rotations/minute. In this part we briefly converted to rotations/minute instead of radians/second in order to make them more intuitive.

B. ż Simulations

While this simulation was entirely necessary for the understanding of our project, it didn't serve much actual truth about our project because it undermines our any coupling of the the force and moment balances. Specifically, our rotational moment was dependent on our vertical velocity. In this simulation we stayed blind to our spin rate, making it a constant. Similarly, we made the initial condition of our vertical velocity 0, simulating the moment we change our angle of attack, and begin ascending. We also get to add some force from the thrusters, because our theta is not zero, there is some small component of that thrust aiding our vertical velocity.

$$\frac{d\dot{z}}{dt} = \frac{1}{m}(N(\frac{3}{4}b\ T\ sin(\theta) + L_{\perp}(\dot{z})D_{\perp}(\dot{z})) - mg)$$



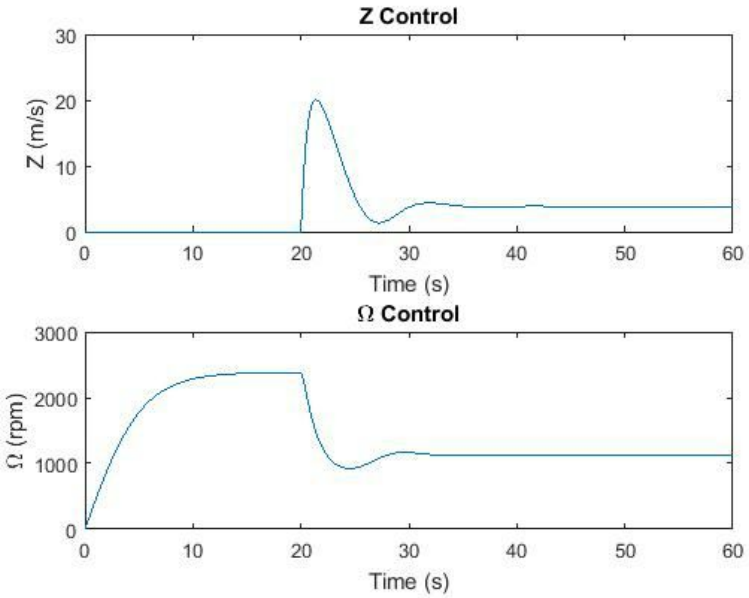
While these were exciting, and hopeful results, we had to bare in mind that they were unrealistic in that the previously stated spin rate was unaffected by vertical velocity.

C. Combined Simulations

After some research, we came to the understanding that the ODE solver can actually process multiple equations simultaneously. In this case we put the equations that described our functions above in state space like form, where we could choose the ODE's inputs in certain locations of a matrix of equations, to result in a vector that when plotted, can visibly understand its coupled nature.

$$\frac{d\Omega}{dt} = \frac{1}{I}N(\frac{3}{4}b\ T\ cos(\theta) - L_{||}(\dot{z},\Omega)D_{||}(\dot{z},\Omega))$$

$$\frac{d\dot{z}}{dt} = \frac{1}{m}(N(\frac{3}{4}b\ T\ sin(\theta) + L_{\perp}(\dot{z},\Omega)D_{\perp}(\dot{z},\Omega)) - mg)$$



In this plot, one can visualize what we were doing. We simulated the thrusters at a constant thrust throughout the simulation, beginning in the same way as the Ω simulation, assuming no angle of attack. Then once we set a positive angle of attack a few things happen to our spin rate. An entirely new form of drag comes in and drastically slows our spin rate. The thrusters are no longer entirely fighting our drag, but only a component of the thrust is. This new wing pitch and vertical velocity means our angle of attack is shallower than it is for desired flight.

The moment of change is clearly set at 20 seconds, both results seem to quickly stabilize at their respective points. This means so long as the air density and thrust stay the same, it will continue to climb. The total distance covered by the Roton in these 40 seconds of flight time is calculated as 197.7 meters, which translates to an average speed of 6.4 meters/second.

We have continued making calculations changing our control parameters (the tail angle and thrust), to gain an intuitive understanding of the Roton's behavior.

V. System Behavior

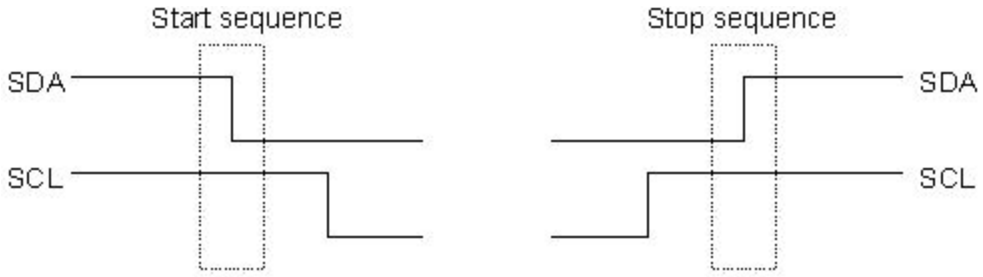
There are four rotary wings that spin in a two dimensional plane about a fixed axis in the middle of body of the vehicle. Each wing is propelled by a brushless DC motor and has a tail wing that is actuated by a servo to control the angle of attack. When the system spins above a certain rate, Ω_0, the system will become passively stable. What we aim to control, after keeping Ω above Ω_0, is $\dot{z}$ and Ω at all times. We need to implement a feedback control loop because simply assuming the actuators will take us to the correct speeds is far too reckless for our purposes. Using a magnetometer, we can calculate our vehicle's spin rate, when the digital compass sees North, the microcontroller knows when one rotation has completed. Additionally, using a gyroscopic sensor we could determine the roll and pitch of the system. The pitch or roll represents the orientation of the system, which allows it to recover stability to a neutral orientation by phasing a wing's servo.

VI. Software/ Sensors

A. I2C

The protocol used to communicate with the HMC5883L and MPU6050 for the project is I2C. I2C, Inter-Integrated Circuit, is a multi-master and multi-slave serial bus technology. For the project's application, the technology allows our microcontroller, the master, to communicate with our sensors, the slaves. I2C utilizes two bidirectional open-drain wires. Serial Data Line (SDA) is used to transmit data bytes between the masters and slaves. Serial Clock Line (SCL) acts as the clock line for the devices to synchronize to. Both the SDA and SCL lines are pulled high with a resistor in its idle state, so the lines are driven low by the devices when there is communication occurring.

Any command from the master to a slave must begin with a start sequence, and end with a stop sequence. These start and stop sequences are used whenever the master wishes to read, write or point to a register.



When a transfer has begun, the master specifies which slave to communicate to, along with the command. As per I2C protocol, the first byte sent on the SDA line is the 7 bit address of the device along, with the last bit being the read or write command. The HMC5883L has an address of 0x1E, and the MPU6050 has an address of 0x68. The write command has a bit value of 0 and the read command has a bit value of 1. The remaining bytes sent in the transfer are the register location and the value to be written (if a write command was issued).

Device	Address	Write	Read
HMC5883L	001 1110 (0x1E)	0011 1100 (0x3C)	0011 1101 (0x3D)
MPU6050	110 1000 (0x68)	1101 0000 (0xD0)	1101 0000 (0xD1)

The I2C library was written using Microchip's peripheral library as grounds to build upon. A key feature of I2C is the ability to share the bus line with multiple devices, thus the software library contains function calls that constantly check for transmitter and receiver overflows, arbitration loss and other statuses. Other functions were created as well to make communication over I2C more efficient by allowing burst reads and writes.

B. HMC5883L

The HMC5883L is a low powered 3-axis magnetometer often used for compassing and magnetometry. For the project's application, the magnetometer is used to calculate the spin rate of the rotating fuselage. The magnetometer reports its orientation relative to magnetic north, which is used as a marker to count rotations of the vessel.

There are several options that allow the HMC5883L to be configured. The software library that has been written provides the capability to change the sample rate, data rate, operation mode and Gaussian range. The spin rate, or Ω, is expected to be at most 2500 RPM, which is a frequency of 42 Hz. And as the Nyquist theorem holds, the magnetometer must achieve a sampling output rate of at least 84 Hz.

$$f_{\text{HMC}} \geq 2f_{\Omega}$$

Such sampling rate is possible with the HMC5883L, however, it requires a specific set of options to be enabled on the device. The standard data output rates on the device only go up to 75 Hz, which is just short of the desired frequency. As a result, the device is configured to run on single-measurement mode, which allows the magnetometer to be sampled up to 160 Hz.

Upon initializing the unit, calibration is required. The calibration function runs a variable amount of tests specified by the user to calculate the offsets of each axis. Resolution of the offsets is determined by the number of tests the user specifies, a greater number of tests achieves a higher resolution, but consumes more time. The calculated offsets are then used in part to normalize the data received from the HMC5883L.

The raw data of the device must be converted and normalized into tangible numbers, such as heading degrees. Raw data is adjusted according to the calculated offset, and multiplied by the a scale dependant of the Gauss scale (milli-Gauss per count).

$$\text{Axis}_{\text{NORM}} = (\text{Axis}_{\text{RAW}} - \text{Axis}_{\text{OFFSET}}) * \text{mG/LSb}$$

Once the axes have been normalized, it allows for the computation of heading in degrees.

$$\text{HEADING} = a\mathit{tan}\ (\text{Y}_{\text{NORM}} / \text{V}_{\text{NORM}})$$

C. MPU6050

The MPU6050 is a motion processing unit that uses a MEMS 3-axis gyroscope and a 3-axis accelerometer. This device allows our prototype to calculate the roll and pitch of the system. Roll and pitch of the system determines the vessel's tilted orientation about the horizontal and vertical axes.

Like the HMC5883L, the MPU6050 requires configuration upon startup. The unit requires to be powered up with a reference to a clock source. We use the clock of the gyroscope because it provides high stability. The gyroscope and accelerometer are also configured to provide several levels of resolution. As for the sampling rate, it is not as strict as that of the HMC5883L -- however we still want it to be quick. The sampling rate of the device is based on the clock of the gyroscope and a coefficient the user determines.

Sample Rate = Gyroscope Output Rate / (1 + User_Coefficient)

Calibration of the MPU6050 is required and follows a similar convention to the HMC5883L. The calibration function takes averages of several samples from the gyroscope and accelerometer and uses it as its offsets.

The raw data is converted and normalized in similar fashion to the magnetometer. Raw data for the accelerometer is scaled according to the gravity experienced by that axis. As for the gyro, the raw values represent the angular rate of the device.

$$\text{Accel\_Axis}_{NORM} = (\text{Accel\_Axis}_{RAW} - \text{Accel\_Axis}_{OFFSET}) / (\text{LSb/g})$$

$$\text{Gyro\_Axis}_{NORM} = (\text{Gyro\_Axis}_{RAW} - \text{Gyro\_Axis}_{OFFSET}) / (\text{LSb/°/s})$$

The roll and pitch can be achieved from computing the normalized accelerometer data.

$$\text{Roll} = (180/\pi) * \textit{atan} (\text{Accel\_Y}_{NORM}/\sqrt{(\text{Accel\_X}_{NORM}^2 + \text{Accel\_Z}_{NORM}^2)})$$

$$\text{Pitch} = (180/\pi) * \textit{atan} (\text{Accel\_X}_{NORM}/\sqrt{(\text{Accel\_Y}_{NORM}^2 + \text{Accel\_Z}_{NORM}^2)})$$

VII. Transmitter/Receiver

A. Interfacing

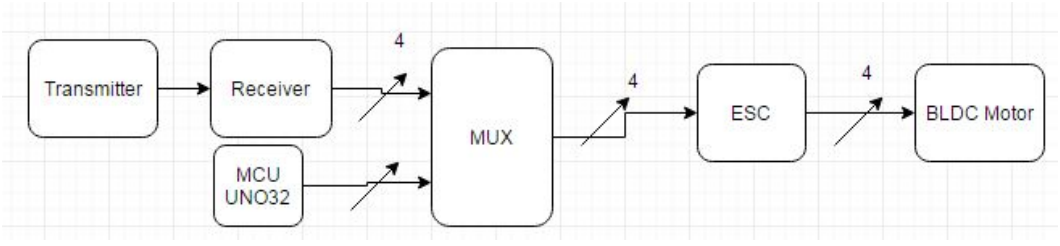
In order to test and operate the prototype safely, we need to use a transmitter to act as the master control to the motors. The transmitter acts as an override to the entire system if the UNO32 fails to perform correct or expected operations. This override includes failsafe switches to turn on and off motors, as well as throttle control.

The transmitter used is the Turnigy 9x, and is paired with the Turnigy 9X8Cv2 receiver. Both the devices require some configuration to be bound together.

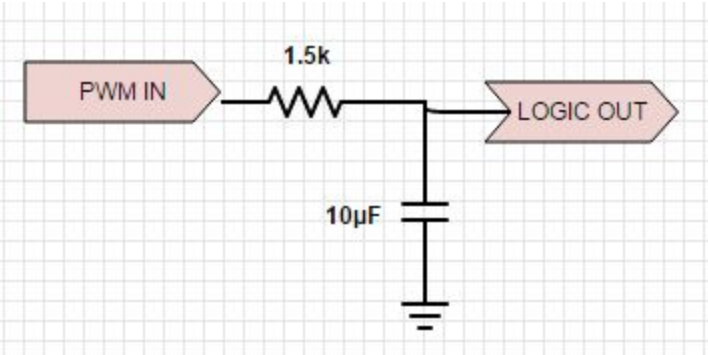
1. Short the bind pin of the receiver with ground
2. Power the receiver
3. Hold the bind button on the back of the transmitter
4. Turn on the transmitter
5. Turn off both transmitter and receiver

After being bound together, the transmitter has many options to map switches and controls to specific output channels. In our application, we need to output throttle and a failsafe switch. The throttle control has a PWM that ranges from 1 ms to 2 ms. Similarly, the failsafe switch has an off value of 1 ms and an on value of 2 ms.

B. Implementation



A multiplexer is used to switch between the transmitter and microcontroller for a PWM signal. The selector bit of the multiplexer is dictated by the status of the failsafe switch of the transmitter. Since the switch on the transmitter outputs a PWM signal, we use a simple low pass filter to convert the signal into a DC voltage.



The advantage of this implementation is that it allows the microcontroller and transmitter to be independent of each other. Routing the signals from the transmitter through the microcontroller would not only make the transmitter dependent on the UNO32, but it would also give the program on it unnecessary complexity to deal with the signals.

VIII. Power/Electronics/Motors

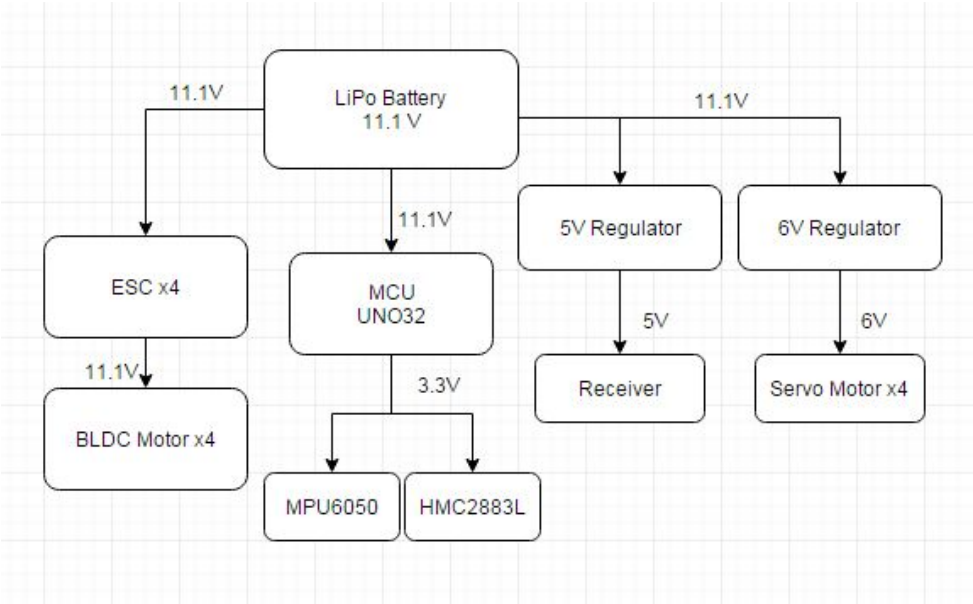
A. Power Distribution

The components that primarily governed the power distribution were the Electronic Speed Controllers (ESC) and Brushless DC Motors (BLDC). Other components such as the servo motors and our sensors were of course taken into account, but most of the current draw would come from the ESC's and BLDC motors. In the early stages of the project, we had to consider what type of motors to use. We wanted motors that could provide enough thrust while also keeping power consumption on the low side. To start off, we researched mostly motors that are used for hobby quadcopters. As per our simulations, we found that BLDC motors used on a standard quadcopter would be sufficient. We ended up purchasing a beginner's quadcopter kit, which included four sets of 6030 propellers, four ESC's, and four LD-Power MT2204-2300KV BLDC motors. In the same purchase, we had decided to purchase four servo motors that would serve as the actuation for the tail of the wing. While the BLDC motors have maximum specifications relative to the percent throttle they are run at, the ESC's maximum current rating is the specification worth paying attention to. For example, if our BLDC motor spins a 6045 propeller at 100% throttle under a supply voltage of 11.1V, the motor is rated to draw 16.6 A. This rating can cause damage if the entire Battery-ESC-BLDC chain is not considered. While the BLDC motor is rated for 16.6 A draw at maximum throttle, the ESC is the component that regulates the current draw of the chain. Our ESC's are rated for a maximum of 15A draw under either a 2S or 3S battery. The chart belows shows various components we are using as well their supply specifications.

Component	Supply Voltage (nominal) (V)
ESC	11.1
MT2204-2300KV BLDC	11.1
HobbyKing Servo Motors	6
UNO32 (MCU)	7 to 15
MPU6050 (IMU)	3.3
HMC2283L (Magnetometer)	3.3
Receiver	5

List of Components with nominal supply voltage

The chart above specifies each of the nominal supply voltages that each component is allowed to operate under. At this point, we condensed the information into a block diagram in order to visualize how many power rails we would essentially need. As in the chart above, we see that the highest voltage needed results from the ESC-BLDC chain, which is a nominal 11.1 V. This was taken to be a value we would base our supply sources around. The 11.1 V would serve as the main supply voltage and would be stepped down wherever necessary. As the diagram below shows, we essentially have three power rails at 11.1 V, 6 V, and 5V.



Power Distribution Block

The system will essentially be run off of an external power source. It was found that the best choice of a power source would be batteries that are specifically used to power quadcopters.

B. Power Component Selection

Once all the supply voltages were figured out, the next step was to draw out our worst case specifications. Doing this enabled us to conduct parts research which in turn resulted in components that would theoretically give our system decent overhead. The ESC was the what we based our parts research on, as it is the component that would draw the most power in they system. With that in mind, we also found the maximum specifications for our sensors as well as our other motors.

Our ESC's, and therefore our battery, will supply however much current the BLDC motors require at a certain throttle. Thus, while the power consumption for one BLDC motor is important, the power consumption of the ESC is more fruitful information. Multiplying power consumption governed by the ESC by four (since we are using four motors in total), we get a current draw of 50 A and power consumption of 555 W. Furthermore, we find that our four servos collectively draw 4 A while consuming 24 W. The microcontroller (MCU), MPU6050,

HMC2283L, and receiver are relatively low power, as they consume .9 W, .0234 W, .00048 W, and .115 W, respectively. To calculate our worst case current draw and worst case power consumption of the system, we add the current draw from each component and power consumption from each component. The worst case current draw was calculated to be 64.102 A while the worst case power consumption was calculated to be 691.03888 W. Each component worst case specification is summarized in the chart below.

Component	Supply Voltage (max) (V)	Current Draw (max) (A)	Power Consumption (W)
ESC	11.1	15	166.5
MT2204-2300KV BLDC	11.1	12.5	138.75
HobbyKing Servo Motors	6	1	6
UNO32 (MCU)	12	.075	.9
MPU6050 (IMU)	6	.0039	.0234
HMC2283L (Magnetometer)	4.8	.0001	.00048
Turnigy 9X8CV2 (Receiver)	5	.023	.115

List of components with maximum ratings

With the worst case specifications listed out and calculated, we were able to purchase a battery and accommodating step-down regulators. While there were many factors that affected the purchase of a battery, there were three factors (excluding price) that were most important to

consider: maximum allowable current draw, power supplied, and maximum flight time. The three are governed by the equations listed below.

$$Current\ Draw = \frac{Capacity}{1000} \cdot (C - rating) \quad [A]$$

$$Power\ Supplied = (Current\ Draw) \cdot (Battery\ Voltage) \quad [W]$$

$$Flight\ Time = \frac{Capacity}{1000} \cdot \frac{60}{Current\ Draw} \quad [minutes]$$

Above, capacity of a battery is given as [mA*hours], the C-rating specifies the rate at which the battery discharges, and the Battery Voltage is given by xS, where x is a number through 1 to 6 (each number represents a supply voltage in increments of 3.7 V). We wanted our projected Current Draw to be at least 65 A while the Power Supplied should be at least 700 W. With this in mind, we see that the constant Current Draw and Power Supplied will be valid only during the Flight Time. For example, if we had a 3S 3300 mA*h 60 C battery, our Current Draw would be 198 A, power supplied would be 2197.8 W, and total Flight Time would be 1 minute. This specific battery yields a poor flight time and is also does not meet our criteria. The battery we purchased was a Zippy Compact 3S 5000mA*h 25 C LiPo. Using the equations given above, this battery provides us with a maximum Current Draw of 125 A, a Power Supplied of 1387.5, and a maximum flight time of 2.4 minutes. Taking our system into account, the battery actually provides 4.68 minutes of flight time (and this is considering the BLDC motors running at 100% throttle). Fortunately, this battery also provides us with some overhead. That is, if we take the current draw and power consumption of the system and subtract the values from our maximum specifications, we get about 60.898 A and 696.46112 W of headroom.

Once a sufficient battery was chosen, the last step for power distribution was to figure out how many power rails we should have. Stated in the previous section, we found there to be four operation voltages: 11.1 V, 6 V, 5 V, and 3.3 V. The ESC, BLDC, and UNO32 run directly off the battery, which is a nominal 11.1 V. The other components then run off of a 6 V, 5 V, and 3.3 V supply. The servo motors are able to change angles the fastest under a 6 V power rail. As a result, we looked into step-down 6V regulators. Here, it was important to pay attention to the current draw of each servo. The more a servo has to keep its angle and the heavier the load it has

to actuate, the more current it should draw. It was found that under stress, the servos would draw about 1 A each. Thus, we looked for a 6 V regulator would be able to supply at least 4 A. Unfortunately there were not many choices, as 6 V is not necessarily a common voltage to step down to (where the most common voltages lie at 5 V and 3.3 V). We then ended up purchasing two of the same module: 6V, 2.5 A step-down regulator D24V22F6, This would provide our servos with more than enough headroom for current draw as well as a nominal voltage supply of 6 V. The receiver runs on 5 V and draws 23 mA, which is provided by the LM2937, a 5V regulator with a 500 mA output. Lastly, the MPU6050 and HMC2883L both run off of a 3.3 V rail. Both sensors are low power, with the first sensor drawing .0039 A and the latter sensor drawing .0001 A. The UNO32 has two onboard regulators, one 5 V regulator with an 800 mA output and one 3.3 V regulator with a 500 mA output. The current draw and power consumption from the sensors is well under the specifications for the onboard regulators, so we decided to power the MPU6050 and HMC2883L via the UNO32.

C. Motor Interfacing/Testing

The first steps to interfacing with both the BLDC and servo motors required building software test modules and obtaining a general idea of how to control them via software (using the UNO32 and not using a standard Transmitter/Receiver setup). To do this, we needed to know how exactly a BLDC and servo motor were controlled, that is, what kind of signal we would need to input. Both motors are controlled by a pulse-width-modulation (PWM) signal. The angle of servo motor is controlled by the width of pulse being sent in while the speed of the BLDC motor is controlled via the same way. Using an open source library, RC_Servo.h and RC_Servo.c, written by Professor Elkaim, that utilizes the timers onboard the UNO32, we were able to write code that outputs any desired PWM signal. Both motors take the same PWM signal oscillating at 20 Hz, with a high time ranging from 1 ms to 2 ms. For the BLDC motors, 1 ms should represent no speed while 2 ms should represent top speed. Once we had a general idea of how each motor can be controlled and how the RC_Servo library worked, we were able to interface with the motors.

The first step to interfacing with the BLDC motor was figuring out how to calibrate the ESC. Upon startup, the ESC does not actually know that a 1 ms width corresponds to no speed or that a 2 ms width corresponds to top speed. The ESC-BLDC chain requires a procedure to be completed in order to achieve these bounds. The general process to have the ESC recognize the bounds is listed in order below.

1. Send in maximum pulse (throttle all the way up)
2. Connect to power
3. Wait for 2 beeps from the BLDC motor
4. Lower PWM from maximum to minimum (throttle high to low)
5. Wait for one beep indicating calibration is over
6. Disconnect power

The above process not only allows the ESC to recognize the low and high bounds, but it also allows for synchronous operation. That is, two or more ESC's will be able to spin at the same speed given a specific pulse width. This process is handled by a function Motor_ESC_Calib(). Once it was ensured that each ESC could operate under the same conditions, an entire library was written to handle initializations, tests, speed control of the BLDC motors, and angle control of the servo motors. These functions were encapsulated under a source file Motors.c.

The initialization takes place in Motor_Init(). This initializes the necessary pins onboard the UNO32 in order to use the RC_Servo library within the Motor.c module. It further sets the pinout to output a PWM signal that corresponds to no speed. It also sets the servo angle to a "middle" position which is a relative 30 degree angle. 0 degrees is when the servo horn is vertically aligned with the motor itself.

All testing takes place within Motor_Test(int *mode*), where the parameter *mode* specifies which test the user wants to run. These tests are to ensure that the hardware is fully functioning. They can be also used for potentially new components that may want to be tested out. There are four parameters that can be sent to Motor_Test(): TOGGLE_TEST, CONT_TEST, INC_DEC_TEST, and SERVO_TEST. These tests respectively run the BLDC motor at a

medium speed then cut the throttle to zero, run the BLDC motor continuously, increment the motor speed to about 30% then decrease the motor speed to zero, and has the servo cover a full range of 60 degrees. The above parameters are #define'd macros that are allowed to be bitwise ANDed in order to run more than one test. After the test harnesses are finished, the BLDC motors are set to zero throttle and the servo motors angles are re-centered to 0 degrees.

The last two functions, Motor_SetThrottle(int *throttle*) and Servo_SetAngle(int *angle*) set the speed of the BLDC motor and set the angle of the servo motor, respectively. The first function takes in a number from 1000 to 2000 specifying how fast the motor will spin. The function makes sure that the parameter only accepts values between the numbers, including the bounds. The latter function also takes in a number from 1000 to 2000 specifying the angle of the servo motor. Like the previous function, Servo_SetAngle() only accepts values between 1000 to 2000 (including the respective lower and upper bounds).

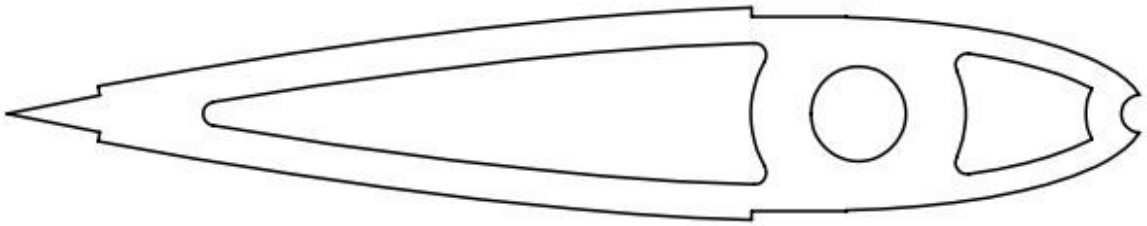
Once the Motor.c library was refurbished and made robust, we were easily able to test the capabilities of the hardware. The most imperative tests to run were tests to figure out average current draw. We first started with the BLDC motors. Using a continuous run test, specified by CONT_TEST, we spun a motor up to about 30% throttle over a time length of about 3 minutes. Using a watt meter, we easily measured the continuous current draw from the BLDC motor. At a 30% throttle, we found that the motor draws about 2.75 A on average. Furthermore, we ran similar tests for a servo motor to find out how much current it is capable of drawing. To do this, we set the servo to hold a certain position and stress the servo horn under this condition. Using a standard supply, we were able to see the current reading change in response to added stress to the horn. The highest current spike the servo motor saw was about a 1 A draw.

IX. Mechanical Prototyping

This project aims to get the Roton into flight, so we had have to come up with some sort of physical prototype, of which we can mount our motors, a battery, and where a microcontroller and receiver may live on the vehicle which will lift all these elements. This was simply a weight versus structural integrity process.

A. Design

We began the design process by designing the wing. We knew we would be able to use the laser cutter as a means of creating custom parts, so we began by creating the parts in 2 dimensional parts to make a 3d structure. Because a number of components needed to comfortably fit within the interior of the wing structure, we decided to use a NACA 0018 airfoil section as shown below.



The large hole in the middle houses a central tube that is the spine of the wing. We chose a tube, because a tube will always be stiffer than a rod. In a design such as this, we would like the whole system to be as rigid as possible. There would also be a rod at the front of the airfoil in order to make it as flat and rounded as possible (not pointed or geometric). The slots are for thin sheets of material in order to make the whole structure stiffer and to prevent the skin of the wing from crumpling in. We could also create an I-beam between these slots. The other carved out parts were simply for the purposes of reducing weight.

This is the base model of our wing's cross section. We needed two arms that would reach back and create a pivot point 0.150 meters between the central pivoting points of both the wing and the tail. One of which, the rib would have to be able to mount the servo motor directly controlling δ. Unfortunately that required the airfoil skin to flare up a bit, slightly disturbing airflow, but something we're required to do live with because of the servo's shape and size.

Between these two airfoil sections was a pylon system that would reach forward and mount a plate for the thruster propeller motor.

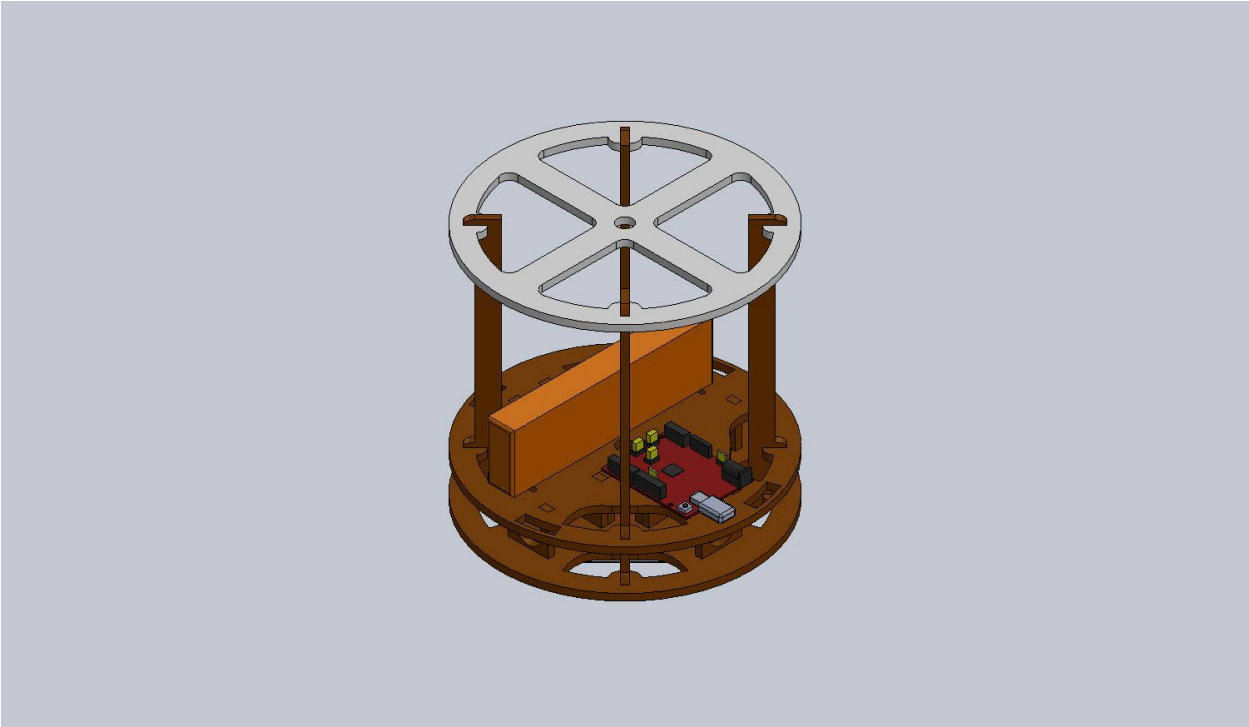
The tail's airfoil section was nearly the same on the outside, but there were two on the inside that would hold a pin that could hold on to the servo linkage, making the transfer of force from the servo to the tail as seamless and stiff as possible. It also has slots and an I-beam to make itself stiffer.

The last protruding part of the tube is to mount the wing onto the central body. In this image the top and bottom slots have not been folded down, nor is there any simulated skin over the wing.

The body was a somewhat simple process because we had a large volume to work with and without a large number of restrictions. The body would have to hold bearings that would allow the wings to swing freely, which have simple hold that have the outer diameter of the bearings cut out of them and pointed in 4 directions. In order to prevent them from falling out, it is a system of sandwiching so the bearings are pulling on the material in a way that the material is quite strong.

The body houses the microcontroller, the battery, the receiver, and all other circuits and components that are needed. The problem with the body is that the battery we selected is quite heavy, and thus the center of mass is not directly along the axis of rotation, according to

Solidworks. We plan to model this as much as possible and whatever ballast is required we will put on the end product. The reason we can't put the battery dead center, is because in the testing phases we will have a wire running down the middle in order to ensure that the vehicle does not fly off and potentially hurt someone or itself. Underneath the whole structure is a lazy susan bearing that will allow the body to spin with relatively little friction.



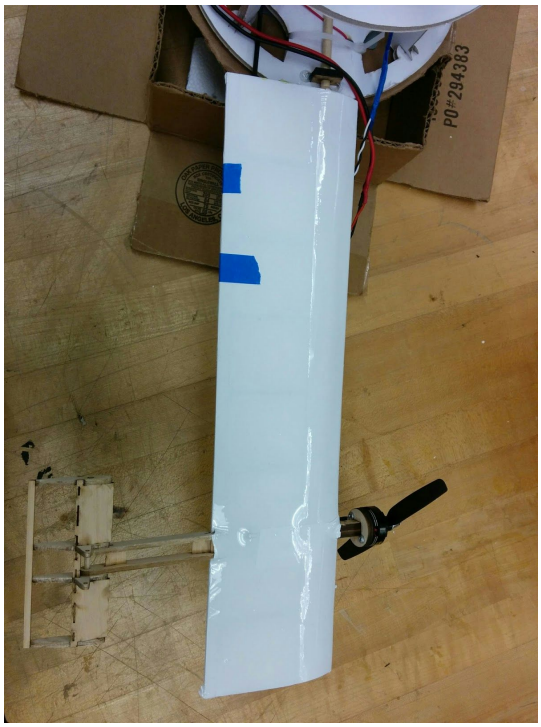
B. Tools and Materials

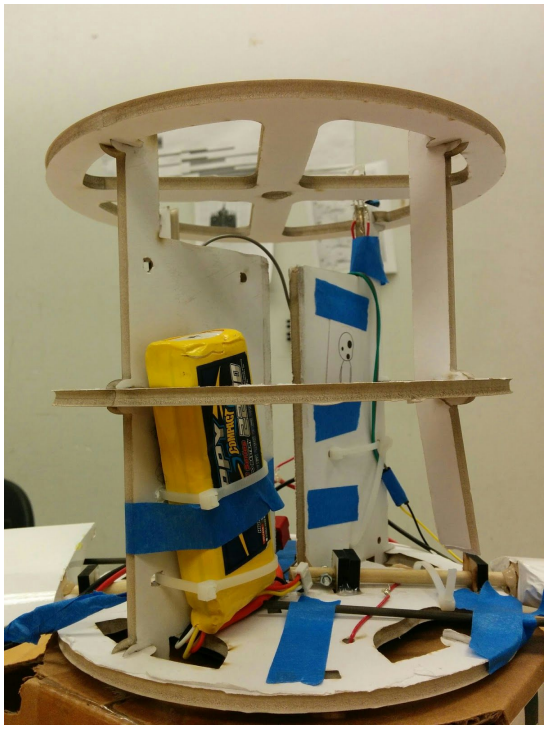
There were several factors we took into consideration when it came to looking for materials for prototyping. The most important factor was the weight of the material. To achieve lift for our system, we needed to stay under a weight budget. But light materials are often weak as well. So it was crucial that we found a material that had a balance between weight and strength. Carbon fiber was the ideal material, but was not as practical because of the price. In the end we settled for balsa, basswood, mdf, foam and a few carbon fiber pieces.

Material	**Density (kg/m^3)**	**Use**
Balsa	160	Wing ribs/support
Basswood	300	Motor mount
MDF	720	Fuselage
Monokote	591	Wrapping wings
Foam	24.5	Wing fillers
Foamcore	64	Fuselage top
Carbon Fiber	1800	Wing rods

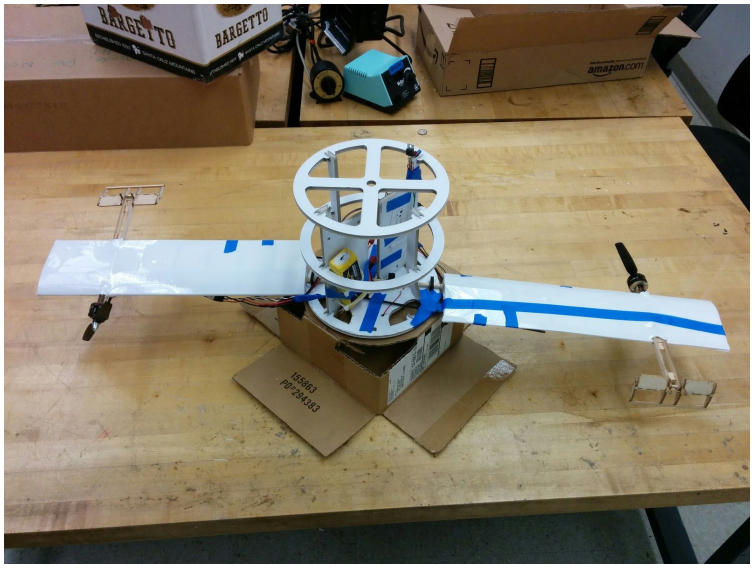
C. Results

After the designing, cutting, assembling, and film process were finished, we are relatively happy with our creation. We have the process down to an art. The wing and body is imaged below





And the system is imaged below



D. Improvements

As evident from our images in our results, we would like to use less tape. As previously mentioned, the design of the body will now use MDF and one can notice other subtle and obvious differences between the printed and used body shape. Much of the wing's innards are housed within the monokote. This prototype only has two wings (we created a simulation that would simulate this).

X. Next Quarter Deliverables

While we are pleased with our own progress, we are excited about where this project will go and how far it will go. Before the end of Spring break March 27, 2016, we would like to get a final prototype going, which will carry itself into flight. Before the end of the academic school year, we would like to be able to fly vertically, without a test harness on it, fly 150 meters high, and be able to move horizontally as well by phasing our servo motors at certain points in their rotation. With all of this, it will be able to draw a square in the with a width of 150 meters, and safely land in the same area it took off from.

XI. Conclusion

We have a modular prototype of our rocket ready to be integrated with our sensors and actuators. The results of our computated physics and subsequent simulations represent desired behavior of our system. Furthermore, the software libraries interfacing the sensors allow the system to accurately elicit data according to its position and orientation. To ensure safety of the system, override capabilities have been implemented to manually control the vessel. Along with the manual override, there is a software library that autonomously controls speed of the motor and angle of the servo. The power budget allows for sufficient flight time even when considering for worst case power consumption. All the individual components have been tested and are being integrated into a unified system. Our progress shows that a completed proof of concept is within sight.