Crash-Proof Drone

ECE592-063 E2

Dr. Mihail Sichitiu

Eric Beppler, Alex Gavric, Anmol Nayak, Diwakar Posam, Meet Thakkar

Abstract

The goal of this project was to make a drone (in this case a hexacopter) with the software and hardware components to give it the capability to prevent any collision with the ground in all situations even if it meant overruling the input given by the pilot when a “crash proof” flight mode is activated. This is an important safety feature for a drone since it will detect dangerous flight patterns that might lead to a catastrophic event for the drone. The “crash proof” flight mode continuously monitors the velocity and altitude of the drone preventing the drone from dropping below a set altitude threshold, which the system autonomously accomplishes through suitably decreasing the descent velocity by increasing the throttle to the motors. Once the drone reaches the threshold altitude, the drone loiters at that altitude until it receives a throttle input which is large enough to put the drone over the threshold thereby restoring user input.

Introduction

Drones are becoming an increasingly popular consumer product as well as a progressively more viable commercial technology with many applications. One of the greatest barriers to entry and sources of continued cost is the tendency of hobby and commercial vehicles alike to crash. Inexperience with controls, hardware, and flying conditions as well as sudden changes in weather and wind can all result in crashes; meaning both professionals and amateurs are at risk. Crashes, in addition to causing frustration, can break expensive and difficult to replace hardware, cause damage to property, and interfere with flight plans and time sensitive objectives. Many solutions to this problem rely on making robust drones or including protective enclosures. While protective enclosures are effective in some cases, they add weight, reduce flight times and responsivity, and decrease the overall appeal of autonomous drones.

A more holistic solution is needed to create a better autonomous system. A drone with the intelligence to detect crash conditions and take preventative action will ensure that new hobbyist and professionals have an easier time learning the ropes, as well as help manage unexpected conditions in any level of flight system. Amateur pilots are prone to crash expensive hardware frequently when manually operating drones; a crash-proof system is needed to help reduce these occurrences. An additional application for a crash-proof system is with fully autonomous systems. A drone operating with no user input cannot compensate its sensors for changes in atmospheric pressure. This leaves the drone with no accurate sense of its altitude, and can lead to crash conditions. By preventing such a system from coming within pre-set ranges in the vertical or horizontal ranges it is possible to prevent damage in the case of changing weather or if the vehicle encounters unexpected obstacles (buildings, vehicles, terrain).

To meet the needs of the community, a “standard” system was created using popular software, hardware, and sensors that is capable of preventing a crash. ArduPilot software running modified firmware is implemented on a Pixhawk Controller to make the system easily replicable and adaptable. A MAXbotix EZ0 ultrasonic sensor was used to establish a definite vertical position and prevent crashes from a rapidly descending drone.

Figure 1 - Ultrasonic Sensor

Related work

There are two approaches to a crash-proof drone. The first, and least eloquent solution is to create a drone that is not impacted by a crash. These crash-ready systems are advantageous for their ease of integration and potentially lower cost. Both can be seen across the consumer space in the form of small, flexible cages encapsulating the drone in multiple dimensions or a single plane or in flexible rotors that allow the drone to “bounce” off of a crash. The first system requires the addition of rigid hardware that limits the payload and flight time of the system. While this does work to prevent most damage to the drone drops from a height or high speed collisions will still impact the drone.

Figure 2 - Crash Ready Drone

The second crash-ready system requires the addition of expensive materials and software that are beyond the capabilities and budget of many operators. An insect inspired system developed and published by a research team from École Polytechnique Fédérale in Switzerland relies on the concept of passive deformation. This technology allows the drones to deform only in the case of collisions and remain rigid and load bearing under normal operation. Unfortunately, this is not a complete solution and breakages were still incurred during testing. A further confounding factor is the level of expertise and resources required to implement such a system as pictured. Again, this solution is “crash-ready” or “crash-resilient” rather than crash proof and still leaves many situations unanswered.

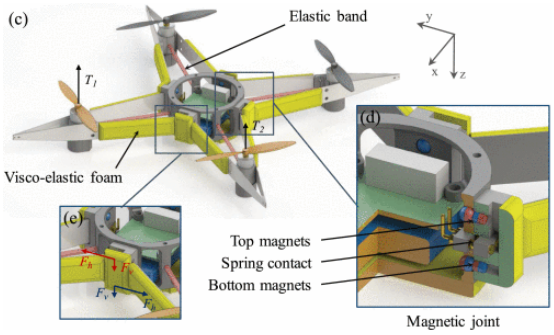

Figure 3 - Resilient Frame Design

We also looked at the DJI Phantom 4 obstacle avoidance feature reviews. Although its performance for avoiding obstacles to the front of the drone is nearly impeccable, it does not have any features to prevent it from colliding with obstacles below it such as preventing ground collision. There have been several issues of the Phantom 4 crashing into trees while landing using the Return to home feature.

Approach

Hardware

For the main functionality of our drone common systems were used, the Pixhawk and the MAXbotix EZ0 ultrasonic sensor are the main components that allow for the ground crash avoidance system. The Pixhawk is the brain of the drone and runs modified ardupilot allowing for drone flight and control. The Pixhawk communicates with the Maxbotix EZ0 ultrasonic sensor and detects its distance from the ground. If the drone detects a decreasing altitude and a cross over the lower threshold limit the drone will respond accordingly. The actual drone components used can be seen in Appendix A. Appendix B shows the testing and hardware revisions and why specific components were selected or adjusted.

Software

The existing ALT_HOLD flight mode was used as a starting point for the crash proof mode software. The control_althold.cpp file was augmented. The threshold the drone recognizes as its minimum acceptable height was added as a variable named “cp_min_alt” and is settable using the param set feature of mavlink. This is used to determine the state the software operates in as shown in figure 4. A float named target_stop_alt is calculated and represents the altitude the drone would reach if the throttle is fully applied at the time of the code executing. This is achieved using a simple parabolic calculation based on the current velocity and acceleration as shown below.

```
float target_stop_alt = current_alt -
current_velocity*current_velocity / (2*
acceleration) ;
```

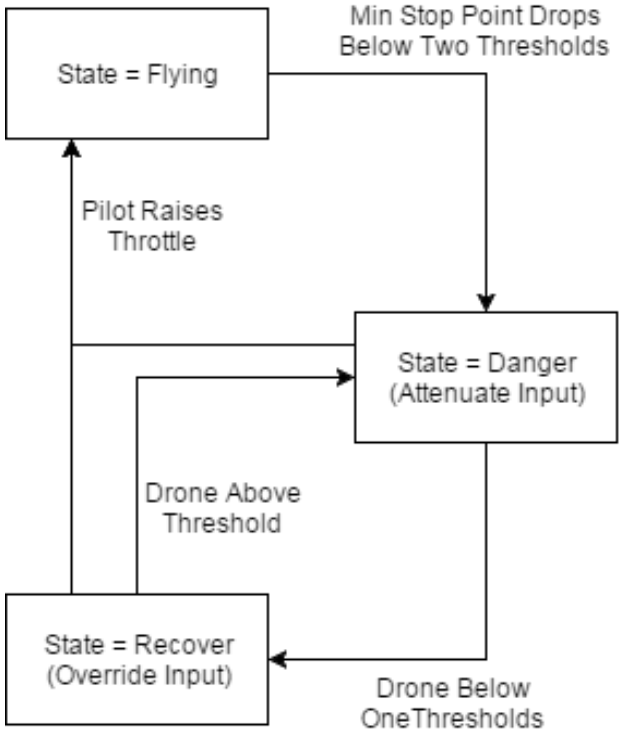

Figure 4 - Software State Machine

The velocity and acceleration are read directly from the ardupilot code and require no additional software to obtain. The current altitude is measured from the rangefinder to eliminate any barometric inaccuracies; this can be obtained from an existing global variable in the system enabled in the mavlink initialization by configuring a range finder.

If the current z axis velocity is detected to be positive (the drone is rising) no additional processing is done and the user retains full control. Similarly if the stopping altitude is above twice the threshold no action is taken. These cases represent the “flying” state. Once the stopping velocity crosses the two threshold altitude the input begins to be attenuated. This is the “Danger” state. As the danger to the drone (lower stopping altitude) is increased the maximum negative vertical velocity of the drone is controlled using an linear attenuation calculated as shown.

```
float alpha = (MIN(current_alt,target_stop_alt) - g2.cp_min_alt) / g2.cp_min_alt;
```

This factor is then multiplied by the pilot_velocity_z_max configuration variable to create a final velocity limit. The target_climb_rate is set to the calculated value forcing the drone to descend at a controlled rate or rise, disallowing high negative z velocity, low altitude conditions.

If at any point the drone detects it has dropped beneath the threshold or is in danger of doing so it immediately attempts to climb until the risk is averted; this is the recover state wherein the user is completely ignored and a safe altitude is attained.

```
target_climb_rate = g.pilot_velocity_z_max*0.1;
```

It is important to note that in all danger cases the target climb rate is compared to the pilot’s desired climb rate and the greater of the two is used. This allows the user to retain as much control as possible and still rise rapidly while maintaining the safety of the vehicle.

Experimental results

Below, listed are the experimental results we have gathered from running simulations of our drone. As can be seen below each of the graphs compares results between two modes. This is denoted by the numbers ‘1’ and ‘2’ on the graph. ‘1’ represents the stabilize mode while ‘2’ represents the crash proof mode. The parameters that are observed are altitude as shown in Fig. X1, servo motor power as shown in Fig. 5, and finally a figure of both graphs to see the relationship between the two parameters as they differ from the two modes.

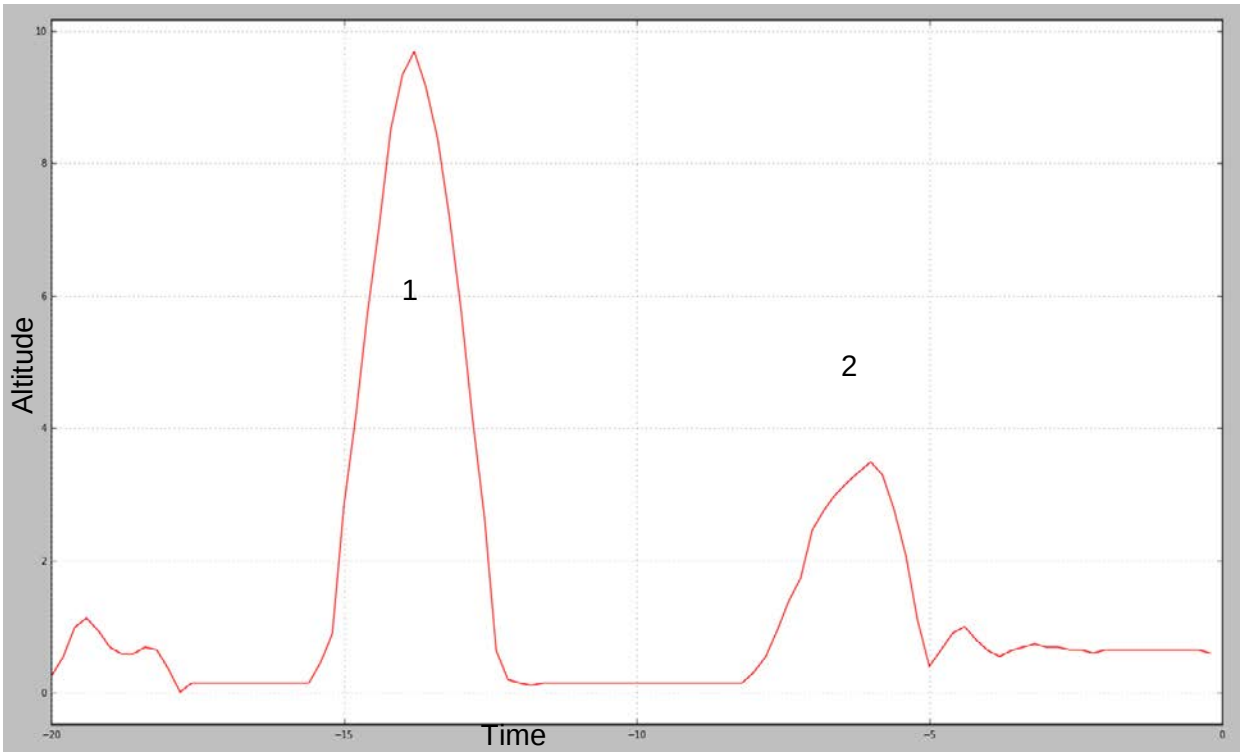

Figure 5 - Altitude vs Time in Stabilize Mode (1) and Crash proof mode (2)

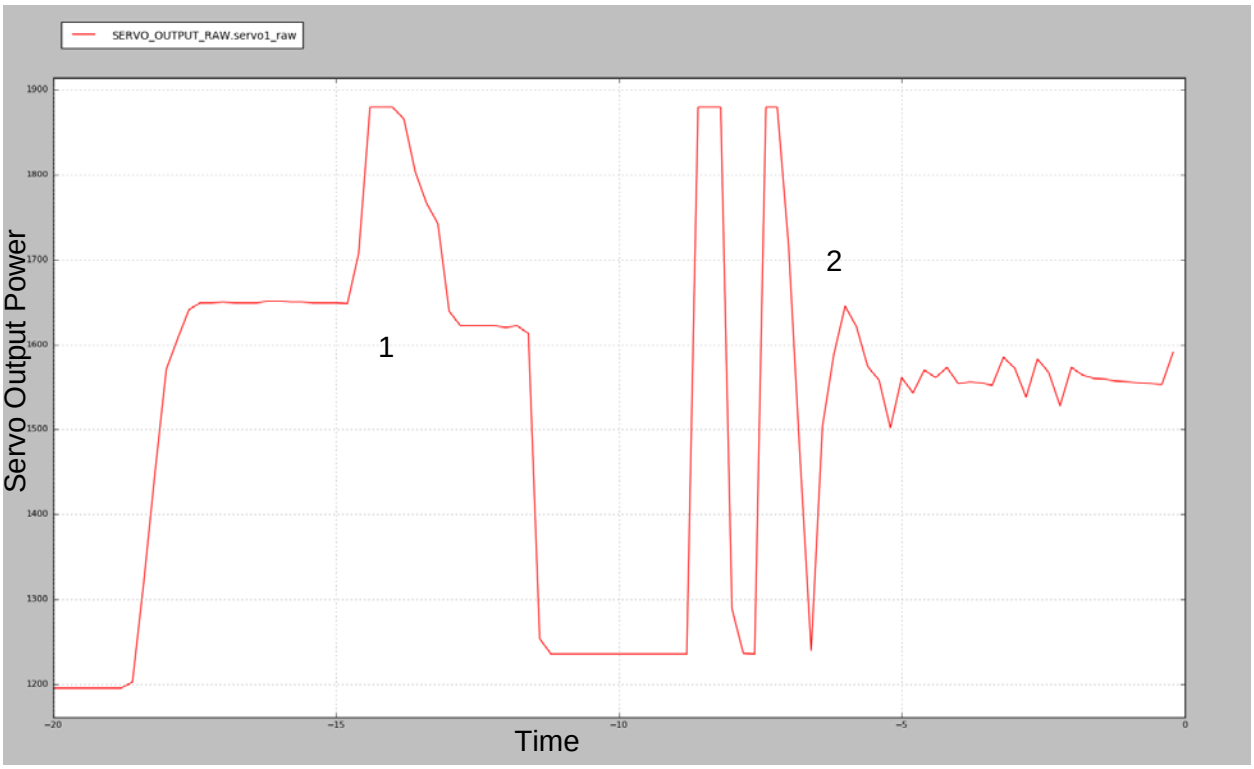

Figure 6 - Servo Output vs Time in stabilize mode(1) and crash proof mode (2)

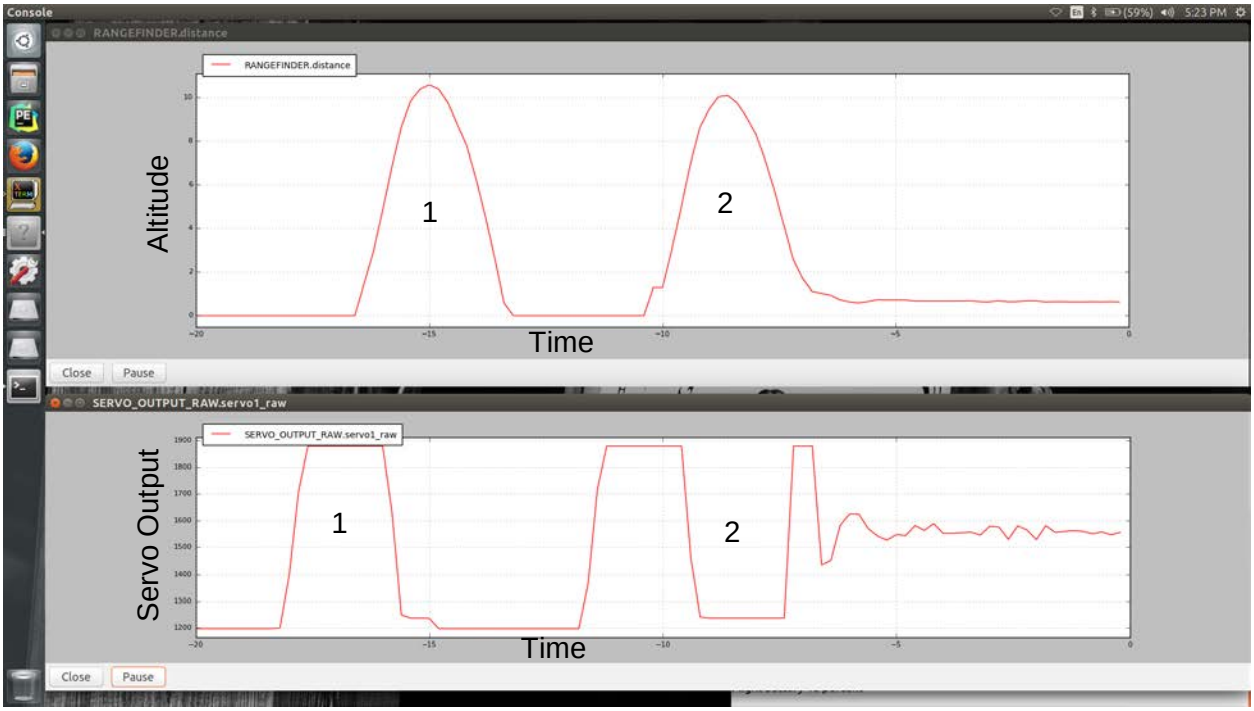

Figure 7 - Side by side relation of Altitude (top) and Servo (bottom) motor output

Analysis

As seen in the results section above obtained from our simulations, we measured the altitude and servo output of our drone in both stabilize mode and crash proof mode. First looking at the Fig. X1 which

shows the altitude as function of time, the first half which is noted by a ‘1’ is when the drone is in stabilize mode while the 2nd half as noted by the ‘2’ on the graph is when the drone is in crash proof mode. When the drone is in stabilize mode, it behaves normally where if the user inputs thrust, the drone files as it is meant to and when no thrust is given the drone falls back to the ground. This can be seen in Fig. X1 where the drone initial starts off the ground i.e. at an altitude of zero and when thrust is inputted the altitude reading goes up to a peak value of 9.5 meters where no more thrust is given. This results in the drone decreasing in altitude and eventually settles back to an altitude of .15 meters. The reason for the altitude settling off at .15 meters is due to sensor location on the drone thus resulting in a small offset. In the second half off the drone where crash proof mode is enabled, the drone lifts as usual and when no more thrust is given to the drone comes back down just like it would in stabilize mode. The difference here is that as the drone get closer to the ground, it automatically increases thrust to stabilize itself and hover over at a certain altitude. In our experiment we set the hover altitude to be .5 meters but in the Fig. 1X it shows to be hovering approximately at an altitude of .65 meters which is once again due to the offset of the sensor. The drone is shown to be stabilizing as can been seen in the second half of the pictures where the tail end has small fluctuations and eventually stabilizes at .65 meters.

The same testing method was used to see the output changes in the servo motor. Once again, the first half of the graph shows the actions of the servo motor while in stabilize mode while the 2nd portion of the graph is a result of being in crash proof mode. In stabilize mode, we can see that if the user inputs thrust then the servo motor behaves accordingly and increases power and when the user inputs no thrust the servo motor power decreases as well. This behavior changes when it’s in crash proof mode as can been seen in Fig. X2. The servo motor behaves similarly to that of stabilize mode when inputted with accommodate for the hovering of the drone. The servo motor power fluctuates in power after the first high some thrust but when no thrust is given the servo motor drops to zero but increases automatically to peak due to fact that the drone is trying to stabilize to a value of .65 meters thus it increases and decreases thrust as needed to maintain its altitude. This relation can be seen in Fig. X3 as shown above, where altitude and power of rotors are directly correlated. As the power is increased in the servo motors, the altitude gradually increases due to user inputting thrust. Looking at the 2nd portion of the graph in Fig. X3, the servo motor power and altitude fluctuates as it tries to stabilize the drone to reach the appropriate threshold altitude. This is due to the fact that rotors are automatically giving more thrust to adjust for the height, and automatically gives zero thrust to overcome the overcompensation of thrust. Eventually it stabilizes to certain a constant thrust value of approximately 50 % power so that the drone maintains a constant altitude. Overall, when activating the crash proof mode the behavior of the drone can be described by the altitude and servo motor power.

Conclusion

Crashing is an inherent risk when it comes to flying a drone, this risk is apparent for both amateurs as well as professional drone flyers. As previously mentioned there are many reasons as to why a crash may occur many of which are unavoidable and unpredictable. These crashes can cause expensive damage to the drone, property, and even interfere with flight plans. There are solutions, but they are not very practical due to bulkiness and weight. To rectify this issue we have created a drone that utilizes the MAXbotix EZ0 ultrasonic sensor to prevent crashes with the ground. The sensor data we collected shows the responsiveness of the sensor. It well overshoots the maximum range and then comes down to the threshold. The results show us that the system is functional, replicable, and has set the framework for a further crash-proof system. We had some issues when first dealing with the flight control module and the system, this is elaborated on in Appendix B.

Appendix A - Bill of Materials and Final Picture

Quantity	Part
3	CCW Propdrive 28-30s
3	CW Propdrive 28-30s
3	CCW 12x4.5 Gemfan Propellers
3	CW 12x4.5 Gemfan Propellers
6	Turnigy Multistar 20A esc
1	Pixhawk flight control system
1	Hitec Optima9 2.4 GHz Receiver
1	PPM - PWM converter
1	FPV Radio Telemetry module.
1	5V Power Converter
1	MAXbotix EZ0 ultrasonic sensor
1	Turnigy 5000mAh
1	Battery Alarm
1	Carbon Fiber Airframe
1	Custom Fiber Board Mounting Platform
1	Aurora 9 Controller

Table 1 – Bill of Materials

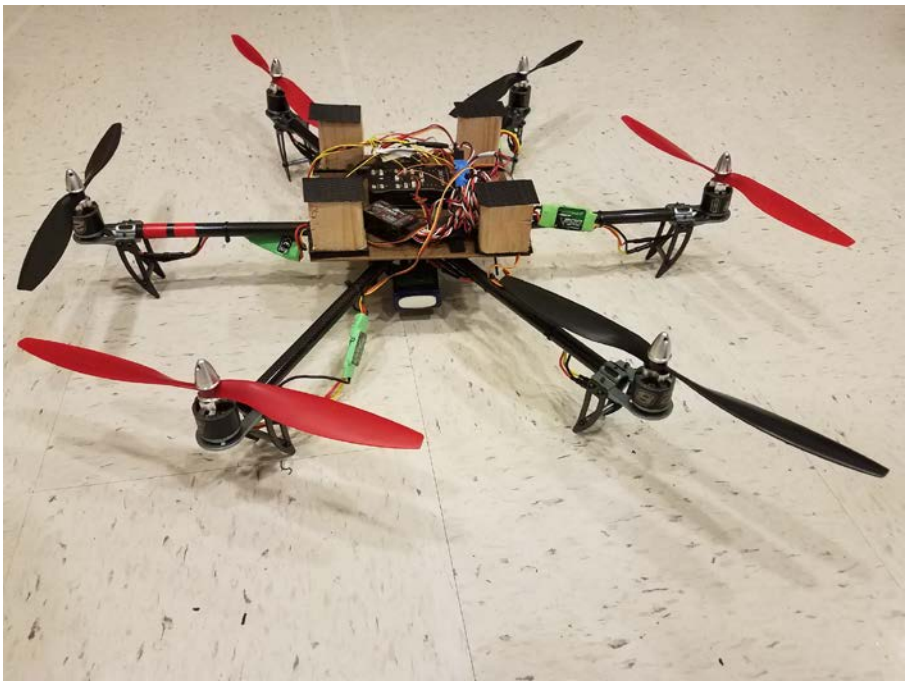

Figure 8 - Complete Drone

Appendix B - Part Selection and Revision

Sensor

Several sensor revisions were made over the course of the project. There were various reasons for the sensor selection and abandonment. The initial decision to use a XCSR04 ultrasonic sensor was driven by availability and proliferation. The sensor is very common in recreational systems and purports to have sufficient resolution and range so as to meet the project requirements. However, once the sensor was attached to the system is was discovered that the system was not performing as advertised and was limited to a range of ~1.1 meters - a threshold much lower than anticipated. This roadblock drove the transition to the MAXbotix EZ0 ultrasonic sensor which functions as expected up to 7 meters with excellent resolution. A laser range finder was originally dismissed as it was not readily compatible with the intended processor (BeagleBone Black).

Flight Control Module

For our flight control module we chose to use a Beaglebone Black with ArduPilot firmware uploaded on it. The Beaglebone Black looked to be promising at first and would run perfectly in the simulations. When implemented on the actual drone the drone would immediately flip causing a propeller to be torn off. We believed this to be user error at first but then after many failed attempts we reached out to an experienced multirotor flier who helped us deduce that the drone would not calibrate properly with the Beaglebone. After this happened we decided to switch to the Pixhawk. As soon as the Pixhawk was flashed with our simulation code we were able to take off, maneuver, and land the drone with ease. We then decided to stay with the Pixhawk.