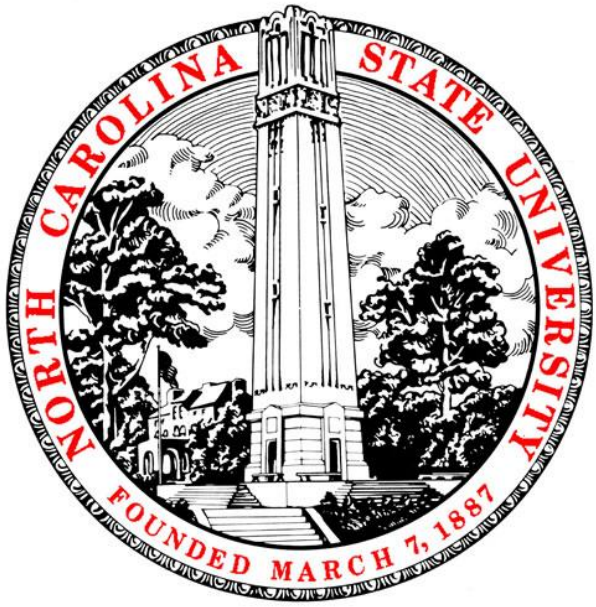
NC STATE UNIVERSITY

Firefighting Drone Challenge

Fall 2014



Control Team 2
Final Report

Tyler Jenkins, Denis Zholob, Ajinkya Khade

1. Introduction

The goal of the firefighting drone challenge is to develop an indoor unmanned aerial vehicle with capabilities to assist firefighting personnel in various environments. Through modular development of an airframe, sensor package, control module, and video feedback, a drone is designed.

A collision avoidance module is a desirable feature for a drone/UAV in high risk environments. Visibility issues, such as smoke, poor lighting, and flames coupled with tight indoor geometries make collision avoidance a necessity.

The team has been tasked with developing such a collision avoidance algorithm, relying on sensor input, a robust airframe, and video feedback developed by other teams. Besides collision avoidance, the team also has a goal of achieving intuitive flight and adaptability to different environments.

This paper has an emphasis on control module design, with overall module integration still to be completed. The methods to solve this interdisciplinary problem are discussed in the next sections.

2. Proposed Controller

The collision avoidance algorithm uses a dynamic proportional control scheme to stop the drone before hitting an object, applying a deceleration based on the drone's velocity and distance from object. By jointly basing the control output on velocity and object distance, the drone should - when correctly tuned - be able to achieve higher speeds in open spaces, as well as greater maneuverability in tight environments. A control scheme, based on standard drone outputs (pitch, roll, and throttle), is developed.

As shown in *figure 2.0.1*, the control scheme separates the drone into components, represented as +x, -x, +y, -y, +z, and -z. Velocity, acceleration, and distances can be resolved into these components. Each component is then controlled independently.

Figure 2.0.1: Drone components

2.1. Calculating Velocity

Velocity calculations are driven by sensor data. The drone's relative velocity to a detected object is calculated using successive bounding box measurements. The velocity is smoothened using a weighted average (*equation 2.1*), with the most recent velocity receiving the greatest weight.

$$\bar{x} = \frac{w_1 x_1 + w_2 x_2 + \cdots + w_n x_n}{w_1 + w_2 + \cdots + w_n}$$

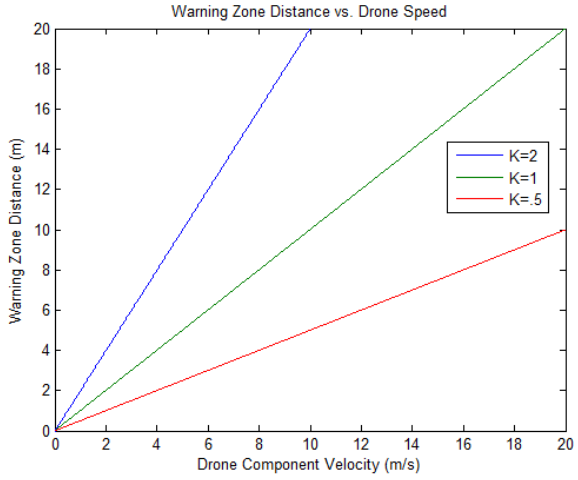
Equation 2.1: Weighted average

When considering using velocity and object distance as control inputs, two questions are presented: "When should the drone slow down?" and "How quickly should the drone slow down?" The team's solution to these questions is discussed next.

2.2. When should the drone slow down? - The Warning Zone

The team has defined a 'Warning Zone' distance in each direction of motion (+/- x, +/-y, +/-z), which linearly increases as speed in that direction increases, as shown in figure 2.2.1. For example, if the drone is quickly moving in only the +x direction, the warning zone will be large in the +x direction, but minimal in all other directions. Each directional warning zone operates independently from the other directional warning zones. The drone will start slowing down whenever an object enters the warning zone.

The essence of the control scheme is that a large component velocity creates a large warning zone, and a small component velocity will create a small warning zone. The collision avoidance will only react when an object enters the warning zone.



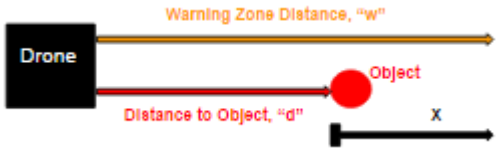
$$w = k * v$$

w: Warning zone distance
k: Tunable parameter
v: Drone velocity, relative to obstruction

Figure 2.2.1: Warning zone scaling

2.3. How quickly should the drone slow down? - The deceleration curve

The drone will decelerate based on an interpolation of object distance and warning zone distance (*figure 2.3.1*). If an object is 50% inside of the warning zone, about 50% of maximum deceleration will be applied. The interpolated percentage is calculated using the following equation:



$$\frac{X}{w} = \frac{w-d}{w}$$

Figure 2.3.1: Object/Warning zone interpolation

To decide how much deceleration to apply (calculated as a percentage of a_{max}, a physical parameter) a second order curve is modeled. The controller will decelerate the drone slowly when objects are further away and more aggressively when objects are closer. The tunable curve can be modeled by using a three point fit, as shown *in figure 2.3.2*. The curve is modeled with *equation 2.3.1*. The user can control the deceleration curve gradient and concavity by setting point (x_3, k).

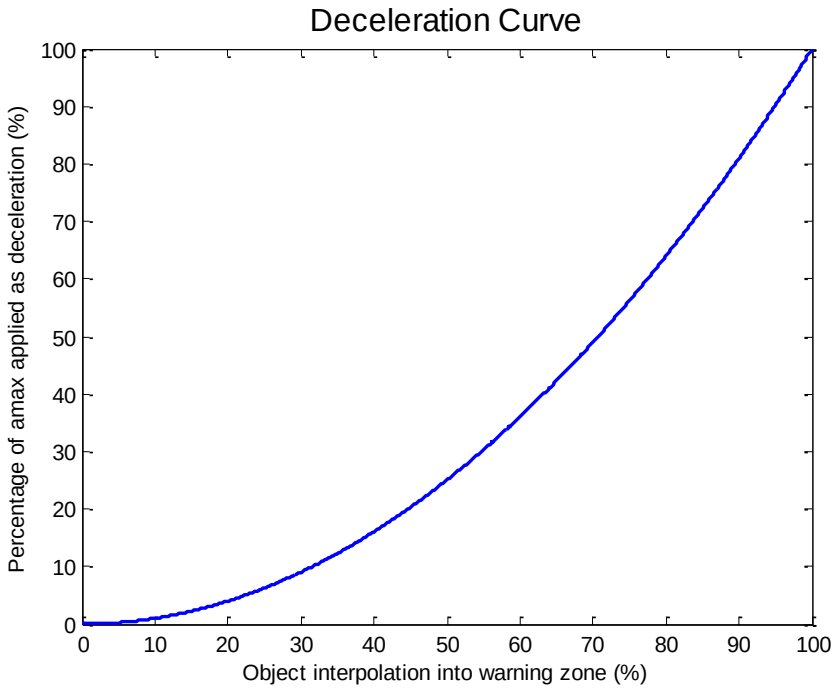


Figure 2.3.2: The Deceleration Curve, k=.25, x_3=.5

$$f(x) = a_0 + a_1(x - x_1) + a_2(x - x_1)(x - x_2),$$
$$f(x_1) = f(0) = 0, f(x_2) = f(1) = 1, f(x_3) = f(.5) = k$$

$$f(x) = \frac{(k - a_{max} * x_3)}{(x_3)(x_3 - 1)}x^2 + \left(a_{max} - \frac{k - a_{max} * x_3}{(x_3)(x_3 - 1)}\right)x$$

Equation 2.3.1: Calculating the deceleration curve

2.4. Worst case scenario - The Danger Zone

The ‘Danger Zone’ is a boundary around the drone that no object should be allowed to cross. This distance is kept constant throughout the flight duration. This serves as a factor of safety, to account for dynamic conditions which can arise during the flight, but can’t be accounted for. In case this zone is breached, the drone quickly needs to be moved away from the obstacle.

To do this, an algorithm similar to the one described in the warning zone is applied. This algorithm kicks into action whenever an obstacle enters the Danger Zone. The location of the drone is then interpolated over the width of the danger zone, in a similar manner to the way it was done in the case of warning zone. The difference being that the danger zone distance is static, as compared to warning zone, which is dynamic. The interpolated point is then used to determine the deceleration that needs to be applied. A deceleration curve, similar to the one in the warning zone, is used to determine this. However the scaling parameters used in this case are slightly different. This also helps to reduce any oscillations of the drone, before it can come to a dead stop.

2.5. Limiting user input - The Caution Zone

The Caution Zone linearly limits user input based on the drone’s distance to an object. The goal of the caution zone is to prevent the user from applying a very large input towards an obstacle in close proximity. This zone extends outwards from the ‘Danger Zone’ to a user defined distance. The simulation showed the best results when the caution zone was stretched to nearly the entire sensor range.

3. Testing and Results

The control scheme has been tested through simulation, with a variety of anticipated situations analyzed. Blender game engine (BGE) was used for the simulation, with details discussed in Appendix A. A basic map of the test environment is shown in *figure 3.0.1*.Test cases and plots of the relevant drone response are presented below.

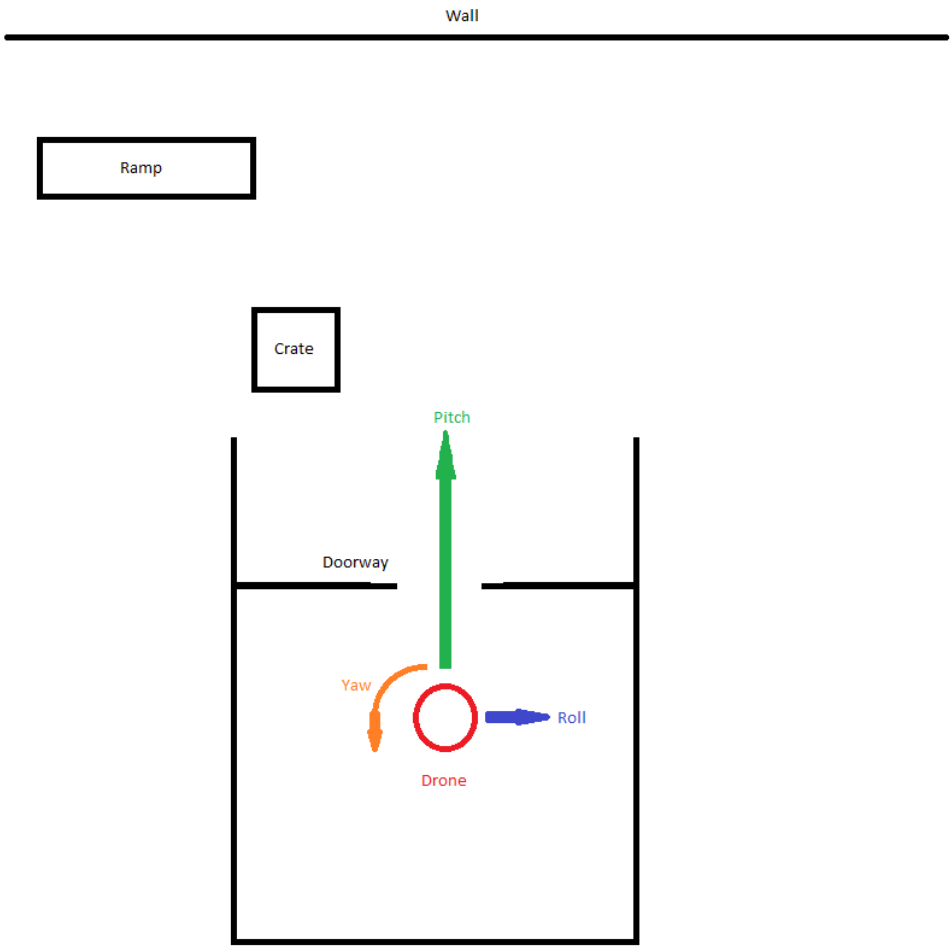


Figure 3.0.1: Blender environment map

All testing was done with parameters (*section 3.7*) shown in *figure 3.0.2*.

Parameter	Value (unit)
Warning Zone Multiplier	1.75 (s)
Maximum Cruising Angle (Pitch/Roll)	.55 (%/100)
Deceleration Limiting Multiplier	.5 (%/100)
Deceleration Curve Scaling Parameter	.25 (%/100)
Danger Distance	.5 (m, from drone center)
Caution Distance	3.5 (m, from end of danger zone)
Dead Zone	.03 (%/100)
Sensor Refresh Rate	15 (Hz)

Figure 3.0.2: Simulation parameters

3.1. Straight into wall

This scenario is to test the response of the drone pitch. The user gives maximum forward pitch input throughout the duration of the test. *Figure 3.1.1* corresponds to the drone pitch, and *figure 3.1.2* corresponds to the drone roll.

Note that much of the drone's physical behavior can easily be matched to the plots and compared with the team's expectations. While *figure 3.1.2* shows minimal drone response from objects picked up by side sensors, *figure 3.1.1* shows interesting behavior. The "door" section shows the drone slowing, but not completely stopping, as the drone approaches the door. The wall response shows the drone slowing completely to a stop. Both results are as anticipated.

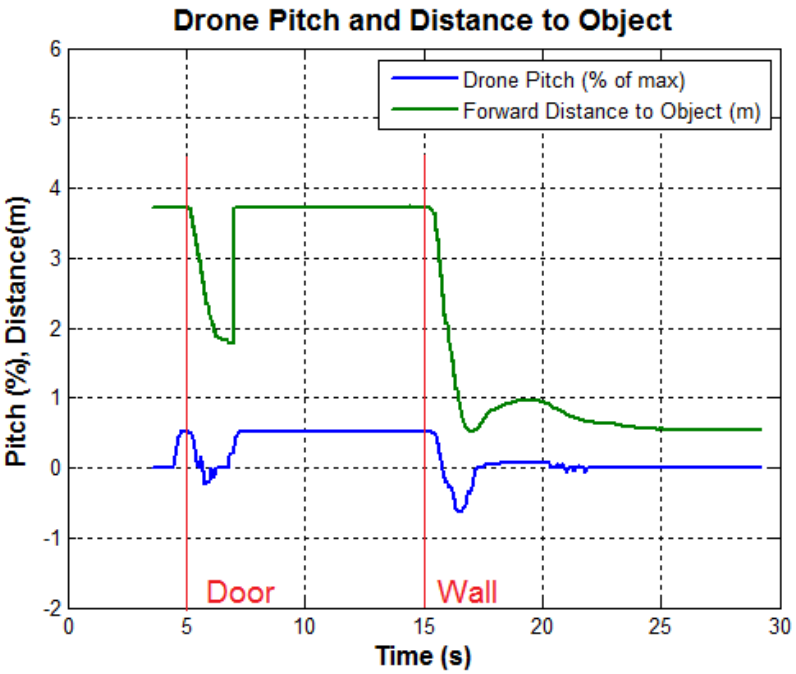


Figure 3.1.1: Forward drone response

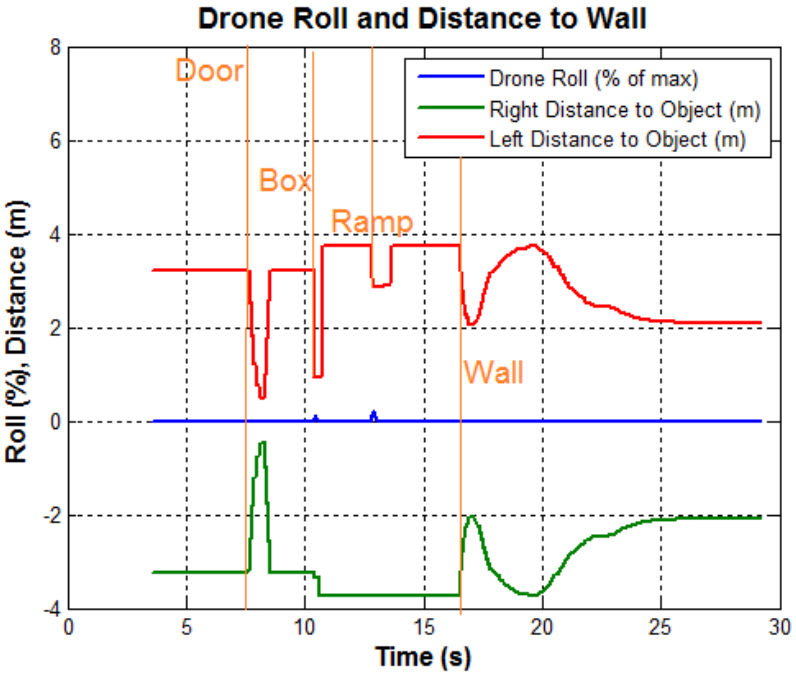


Figure 3.1.2: Side drone response

3.2. Hallway

This scenario tests the drone's response in a hallway. The drone is steered into the hallway at an angle, with full pitch and roll applied. It then "rides the wall" down the hallway.

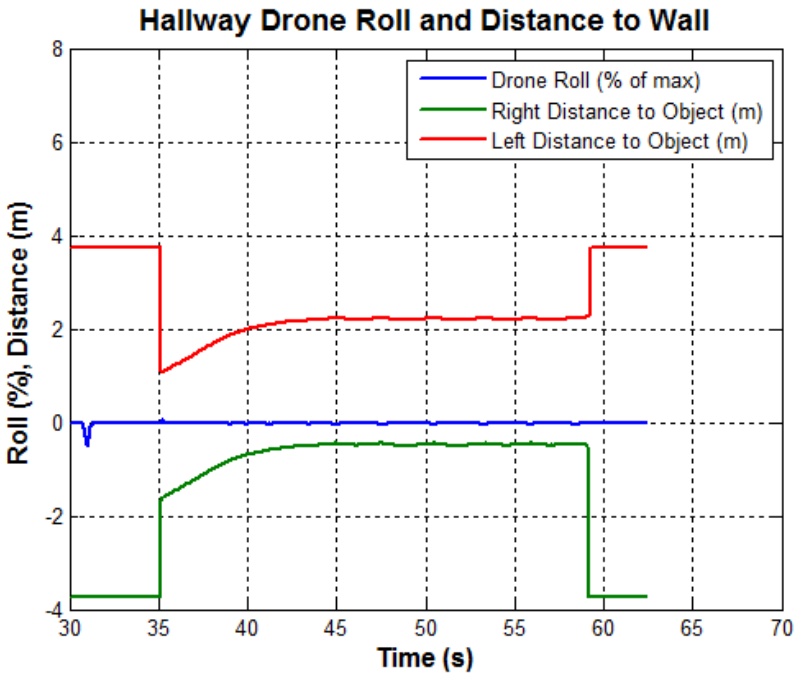


Figure 3.2.1: Drone response in a hallway

Figure 3.2.1 shows the drone response. At 20 seconds, waves begin to appear in the plot. This is representative of the mechanisms driving the drone's motion. It essentially moves forward, is nudged to the left, then moves forward again, and is again nudged to the left, and so on. This takes place very quickly, and produces smooth motion.

3.3. Approaching a wall using multiple user inputs

This scenario represents the user giving full pitch and roll inputs. The drone starts off facing the wall, with no yaw angle, clear of the doorway and other obstructions. Full pitch and roll are applied throughout the entire test.

As shown in *figure 3.3.1*, the drone behaves as expected. When the sensors detect the wall, the drone is decelerated in the forward direction. This deceleration does not affect the drones roll input, and the drone continues to move to the left unobstructed.

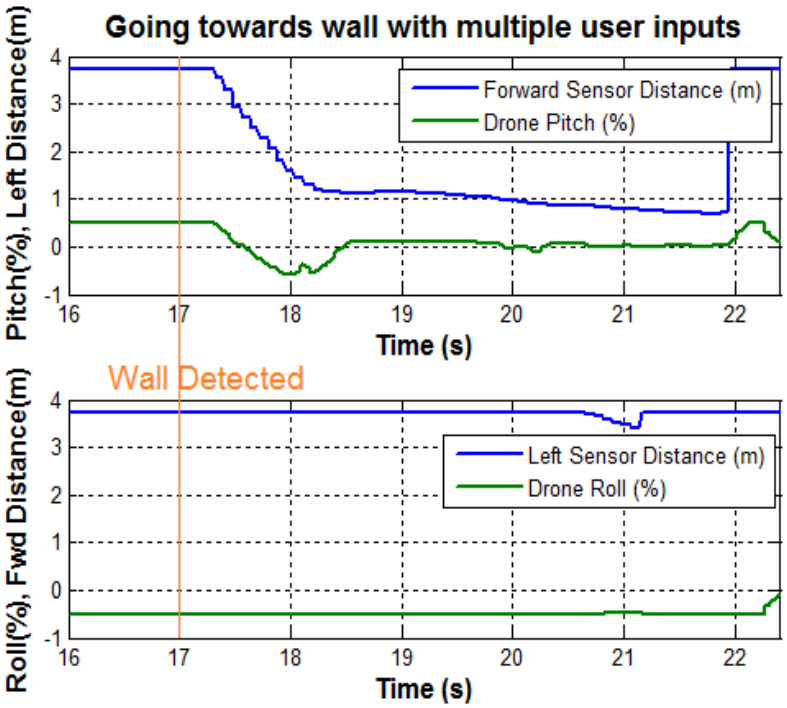


Figure 3.3.1: Multiple user inputs

3.4. Stairs

This scenario represents the drone going up a ramp or stairs. Note that the forward and downward sensors both detect the stairs. The throttle is somewhat intermittent, which is not ideal.

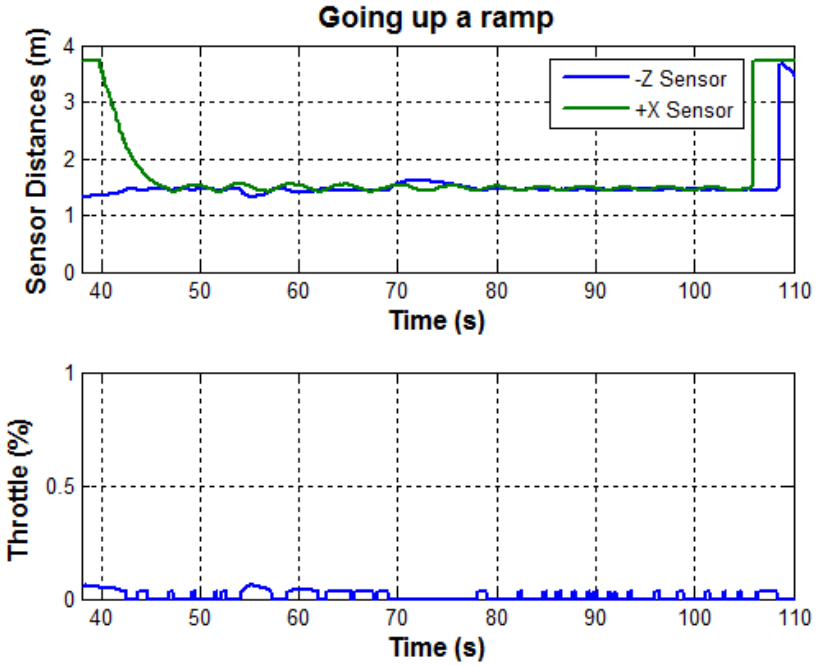


Figure 3.4.1: Going up a ramp

3.5. Greater maximum drone velocity

Test 3.5 is a repeat of 3.1 with a greater maximum drone velocity. The maximum cruising angle was changed from .55 to .65. As the drone had already been tuned to its top speed, this slight increase in speed causes the drone to nearly crash. Oscillations become much larger and the drone becomes less stable.

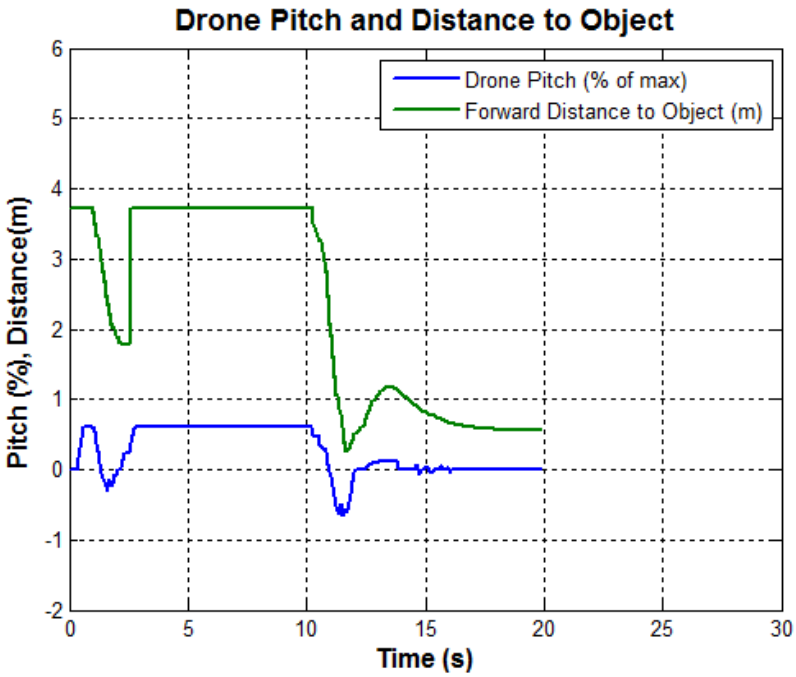


Figure 3.5.1: Straight at wall with greater max velocity

3.6. Lower sensor refresh rate

Figure 3.6.1 shows a repeat of 3.1, with sensor refresh rate changed from 15 Hz to 7.5 Hz. In simulation, it is clear that reducing the measurement frequency leads to more oscillations, and with enough reduction, will cause the drone to crash.

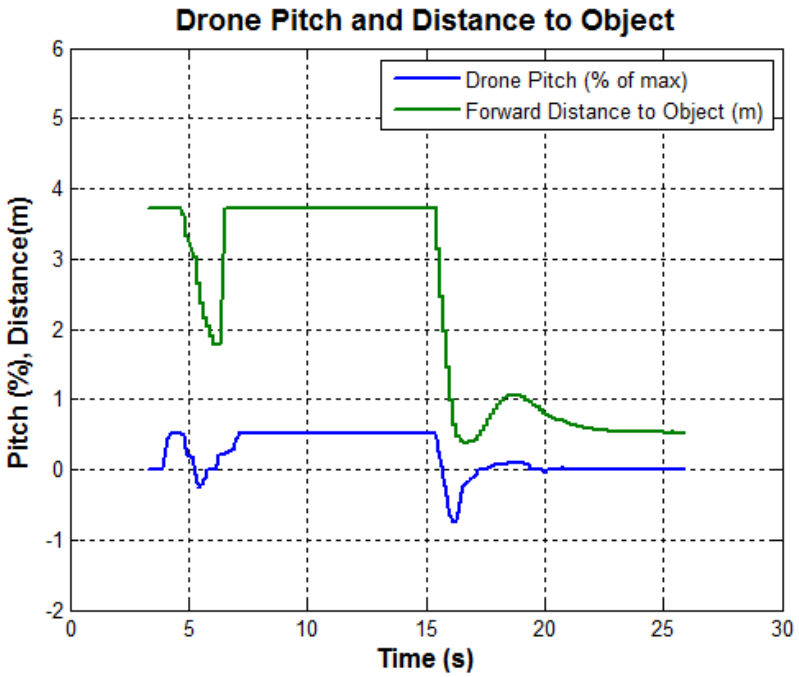


Figure 3.6.1: Straight at wall with decreased sensor refresh rate

3.7. Parameter tuning

Key parameters are qualitatively discussed to give a sense of how to tune the drone. The goal for the parameter tuning section is to convey a sense of realistic parameter ranges and relate parameter changes to their associated design tradeoffs.

3.7.1. Warning Zone Multiplier

The Warning Zone Multiplier linearly scales the warning zone according to the drone's speed. It can be thought of as the warning zone's sensitivity to the drone's speed. Therefore, the Warning Zone Multiplier depends directly on the maximum allowable top speed. A greater top speed will require a larger warning zone multiplier. A greater multiplier has the effect of starting to slow the drone down much earlier in the flight, thus allowing the drone to slow down over a greater distance, and hence being able to stop safely.

Due to limited sensor range, an increase in this parameter has no positive effect on the performance of the drone. This is because the interpolated location of the drone inside the warning zone places it very high on the deceleration curve as soon as an obstacle enters the sensor range. However, the drone has some rotational inertia, and it cannot change its attitude fast enough to match the output of the control scheme.

Hence, the obstacle breaches the danger zone to a greater extent and the drone takes much longer to settle down and come to a dead stop.

3.7.2. Maximum Cruising Angle (Pitch/Roll)

For a given maximum sensor range and drone physical model, there is a maximum speed from which a drone can slow down safely. Another factor contributing to this is the rotational inertia of the drone. In cruising conditions, the control logic limits the pitch/roll of the drone to a maximum allowable value corresponding to this speed. As this parameter is sensitive to the physical drone, the relative behavior of this parameter is summarized as follows:

- Sensor Range - A greater sensor range allows for a greater maximum cruising angle.
- Rotational Inertia - Less inertia, or a more ‘agile’ drone, allows for a greater maximum cruising angle.
- Thrust - A greater value of maximum thrust available in any horizontal direction would allow for a greater maximum cruising angle.

3.7.3. Deceleration Limiting Multiplier

The Deceleration Limiting Multiplier limits the maximum retardation which the control logic applies on the drone once an object breaches the danger zone. Since we are trying to ‘limit’ the retardation applied, this parameter can take values in the range (0, 1). By limiting the deceleration this way, the amount of backlash is reduced, and the drone settles to a stable position sooner. A higher value of this parameter would be better from a safety standpoint, whereas a lower value would be better from a smoothness standpoint. A balance needs to be achieved between the two, based on operating conditions and user preferences. A change in the danger zone distance would affect this parameter in the following way:

- A larger danger zone distance would allow for a lower value, as the drone has a larger distance over which it can stop.
- A smaller danger zone distance would require a larger value, as the drone would require a greater retardation over this smaller distance, in order to prevent a collision.

It is not recommended to try and change this parameter too much for a given danger zone distance.

3.7.4. Deceleration Curve Scaling Parameter

This parameter determines the aggressiveness of the control scheme. It can take values in the range (0, 1). A smaller value means that the control logic increases the applied retardation at a slower rate initially, and at a greater rate as the obstacle gets closer to the drone. The opposite is true for a higher value.

Better behavior has been observed in the lower range of values. Particularly, the time required by the drone to settle down to a stable position, when at the edge of the danger zone, is better with lower values.

3.7.5. Danger Distance

A greater Danger Zone Distance allows the drone to travel at greater speeds, as any possible breach in the danger zone can be handled much more easily. This would also make the drone performance smoother. However, navigating through tight geometries would also get tougher.

3.7.6. Caution Distance

Similar to Danger Zone distance, a greater caution zone distance makes the drone performance smoother. Because the controller limits user input when an object is within the drone's caution zone, an excessive caution zone distance can limit performance.

3.7.7. Dead Zone

The dead zone is a range of output around the neutral state. If the output from the control algorithm lies in this small region, it is set to zero before being passed on to the flight controller. This eliminates minor oscillations of the drone caused because of noise in the sensor input. As this dead zone is made larger, the drone loses its ability to respond to small stimuli. It also becomes tough to navigate in tight geometries, where the control scheme can only allow the drone to accept small inputs. The recommended value for this parameter is about 0.025-0.05 (2.5-5%).

3.7.8. Different Operating Modes

As the discussion about the various tunable parameters has highlighted, the flexibility of the control scheme allows for different modes of operation. The parameter sets can be tuned to achieve varying levels of smoothness or agility. The control scheme can also be tuned to have different pre-set modes for different environments. For example, one mode could be tuned for large open spaces (such as warehouses and industrial areas), where top speed is more desirable than smoothness. Another mode could be set for residential environments, where the drone is required to navigate through intricate spaces, and where it can't achieve high speeds anyway.

4. Practical Implementation

The Beagle Bone Black is being used to execute the control algorithm. It is essentially a tiny computer running Linux, with capabilities similar to desktop machines. It was chosen over similar items (Raspberry Pi and Arduino microcontrollers) because of its superior processing power, and numerous pin headers for inputs and outputs.

In order to process user input and output signals, the BBB must be configured to use Pulse Width Modulation, or PWM. The BBB's Linux OS inherently has unreliable timing for decoding RC style PWM signals, but the onboard Programmable Realtime Units (PRU's) can decode these signals in real time. For information on using the Beagle Bone Black for PWM, see Appendix B.

4.1. Suggested Improvements

- Develop a small layer between the control scheme and flight controller to adjust thrust depending on the drone attitude, so that the altitude is maintained. This would require access to gyroscope readings in order to be able to manipulate thrust fairly accurately.
- The drone could possibly have another set of short range sensors with narrow sensing cones. The only purpose of these sensors would be to decipher if there is a gap wide enough for the drone to be able to pass through, even when the regular set of sensors indicate that there are potential obstacles in that direction. These could prove to be especially handy in situations like navigating through doorways.
- Building in a 'Take-Off Mode' which would increase the thrust gradually till the drone achieves the desired height.

5. Conclusion

The semester goal was to create and simulate a control algorithm to provide collision avoidance for a drone. Design goals of providing intuitive flight and environment adaptability were also established. The team created a control scheme based on these design goals.

Throughout algorithm development, various scenarios, such as doorways and walls, were simulated within Blender Game Engine. As the control scheme was developed, the controller performance was characterized through data logs. Desirable performance criteria (low oscillation, stable, higher top speed) were established and used to continue controller development.

The component based, tunable approach allows for flexibility, where the user can choose desired performance. The dynamic approach allows for higher top speeds in open areas, plus tight maneuverability in constricted space. Our team believes the proposed control scheme shows favorable characteristics and satisfies the presented design goals.

6. References

[1] Shabaz, "Working with the PRU-ICSS/PRUSSv2," 21 May 2013. [Online]. Available: http://www.element14.com/community/community/designcenter/single-board-computers/next-gen_beaglebone/blog/2013/05/22/bbb--working-with-the-pru-icssprussv2.

[2] M. Sichitiu and R. Dutta. [Online]. Available: https://sites.google.com/a/ncsu.edu/firefighting-drone-challenge/home. [Accessed 3 12 2014].

[3] BeagleBoard.org Foundation, "Getting Started with BeagleBone & BeagleBone Black," BeagleBoard.org Foundation, [Online]. Available: http://beagleboard.org/getting-started. [Accessed 3 12 2014].

[4] Blender Foundation, "Blender User Manual," Blender Foundation, [Online]. Available: http://wiki.blender.org/index.php/Doc:2.6/Manual#Game_Engine. [Accessed 3 12 2014].

[5] Blender Cookie, "Blender Interface and Navigation," Blender Cookie, [Online]. Available: http://cgcookie.com/blender/lessons/interface-and-navigation/. [Accessed 3 12 2014].

[6] Blender Cookie, "Learn Blender: Blender Basics," CG Cookie, [Online]. Available: http://cgcookie.com/blender/cgc-courses/blender-basics-introduction-for-beginners/. [Accessed 3 12 2014].

[7] SolarLune, "Game Up!," 23 January 2011. [Online]. Available: http://solarlune-gameup.blogspot.com/2011/01/using-python-in-blender-game-engine.html. [Accessed 3 December 2014].

[8] C. Thames, "Tutorials for Blender 3D," [Online]. Available: http://www.tutorialsforblender3d.com/BGE_Python/BGE_Python_Main.html. [Accessed 3 December 2014].

[9] N. Gageik, T. Muller and S. Montenegro, "Obstacle detection and collision avoidance using ultrasonic distance sensors for an autonomous quadrocopter," University of Wurzurg AIT, Wurzburg, 2012.

[10] "CentMesh: The NCSU Centennial Wireless Mesh Project," North Carolina State University, [Online]. Available: http://centmesh.csc.ncsu.edu/.

[11] Blender Foundation, "Download Blender 2.72," [Online]. Available: http://www.blender.org/download/

[12] R. C. Nelson, "BeagleBoardDebian," 30 October 2014. [Online]. Available: http://elinux.org/BeagleBoardDebian.

[13] BeagleBoard Foundation, "BeagleBoard TI-PRU Package," [Online]. Available: https://github.com/beagleboard/am335x_pru_package.

[14] Texas Instruments, "Pin Mux Utility for ARM(R) Microcontrollers," 16 June 2014. [Online].

Available: http://www.ti.com/tool/pinmuxtool.

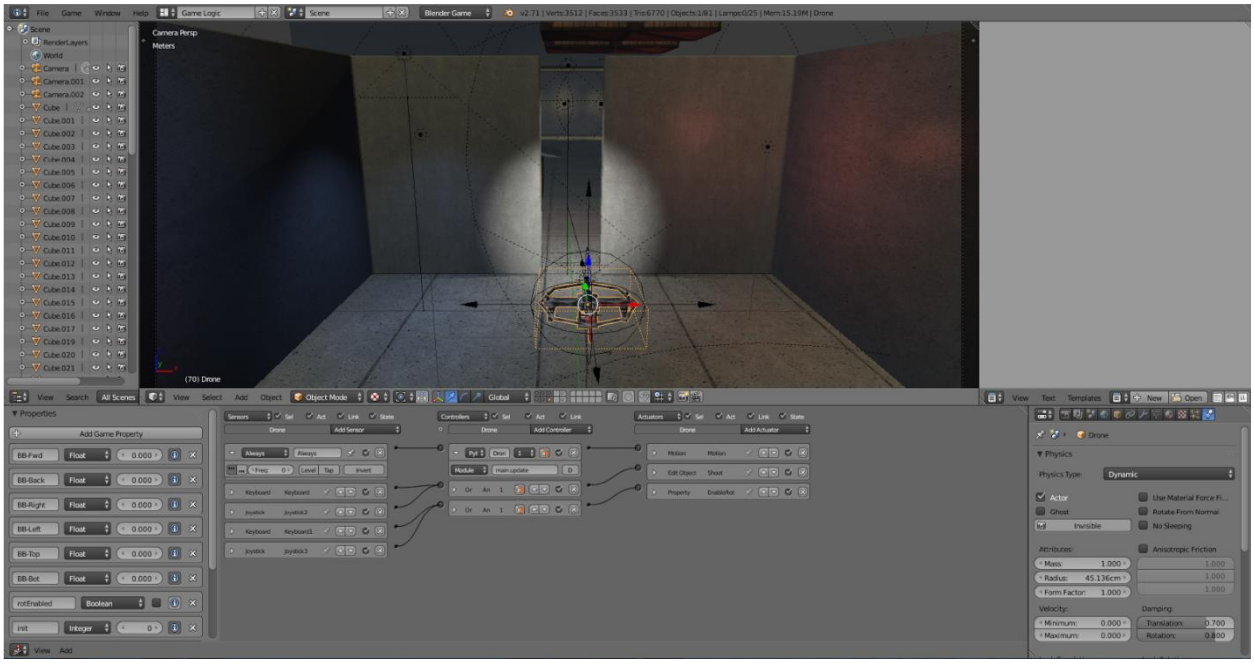
[15] "Beagle-Fly," October 2013. [Online]. Available: https://code.google.com/p/beagle-fly/downloads/list.

[16] Owen, "Wireless Servo Control - PWM with the BBB PRU," Embedded Things, 21 August 2013. [Online]. Available: http://www.embedded-things.com/bbb/wireless-servo-control-part-3-pwm-servo-control-with-the-bbb-pru/.

[17] Elias, "BeagleBone Black GPIO mux for PRU with device tree overlay," 9 June 2013. [Online]. Available: http://hipstercircuits.com/beaglebone-black-gpio-mux-for-pru-with-device-tree-overlay/.

Appendix A - Blender and BGE tutorials

Blender Game Engine



Installation

Blender can be downloaded at [11] for Win, Linux or Mac. For windows, the installation wizard should be followed, with the defaults. Additional instructions can be found at [4], [5], and [6].

Note: Blender is an extensive 3D creation suite, with many features, this simulation leverages the built in game engine. There is a good learning curve to blender that is not required to run the simulation, but is required to construct the environment from scratch, thus it is not discussed. However if one should choose to dig deeper and try to reconstruct the scene from scratch, the tutorial sites referenced are very helpful.

To start the simulation:

1. Open the FirefightingSim.blend file with blender. The file can be opened by a “drag and drop” onto the blender icon.
2. Hover your mouse over the 3D Viewport, shown in *figure A.1*.
3. Press the P Key to Play the game

To end the Simulation

Press the ESC key
Press the R key to restart the simulation

Simulation Keyboard Controls

Up/Down Arrow Keys: Move Forward/Backward
Left/Right Arrow Keys: Move Left/Right
W/S Keys: Move Up/Down

A/D Keys: Rotate (Yaw) CCW/CW
+/- or PgUp/PgDown Keys are used to rotate the camera (your perspective)
½ change cameras (your perspective)

Simulation Game Controller Controls

Right stick: Move Forward/Backward/Left/Right
Left stick: Move Up/Down Yaw CCW/CW
Bottom Triggers: Rotate the camera (your perspective)
Top Triggers: Change cameras (your perspective)

How does it work?

Python Files

The simulation consists of multiple files; some are critical to the simulation while others contain helpful auxiliary methods. The code is written to be as modular as possible, meaning one can comment out a method and remove some functionality.

main.py

Simply instantiates a new Flight Controller class, and calls its update function, to start the controller. This is attached to the drone object inside of the game engine.

FlightController.py

This is a very basic simulation of a flight controller, taking in inputs to move the drone and translates that into a Force and Torque on a drone object inside the game engine.

Note: this is not a fully realistic simulation, as the forces are applied on the object center rather taking into account any rotor blades and a resulting force/torque on the drone resulting from the thrust change on the motors. In order to achieve a more realistic simulation, actual flight controller code would have to be used. Due to time constraints this was not implemented but, one version of such controller code that can possibly be used can be found at [10] (sim_multicopter.py).

DroneLogic.py

This is the brain of the drone, receives input from sensors and user input. It contains the actual control scheme. Processes the data using an avoidance algorithm and sends modified inputs out. This function is responsible from keeping the drone from crashing into any obstacles that the sensors see.

Input.py

This contains the functions to get user input to the BGE; this can be in a way of keyboard keys or an attached game controller.

Sensors.py

This simulates sensors on the drone.

Input.py

This contains the functions to get user input

Camera.py

Adds height control to the main (rear) camera (or whatever camera the script is attached to)

VectorFnx.py

Contain functions for various vector operations. (Dot product, add, subtract, etc...)

Formatter.py

This functions to format numbers nicely, rounding decimals to a specific digit, and printing arrays neatly to be easily used in data analysis.

DataLogger.py

This is responsible for writing data to a log file for later review.

DataPlotter.m

Using the log files generated while playing the game, analysis can be done with tools like MATLAB. For a quick plot, change directories in MATLAB to the data log location. To use the built in GUI, right-click on the file and select “Import Data”.

Connection to BGE

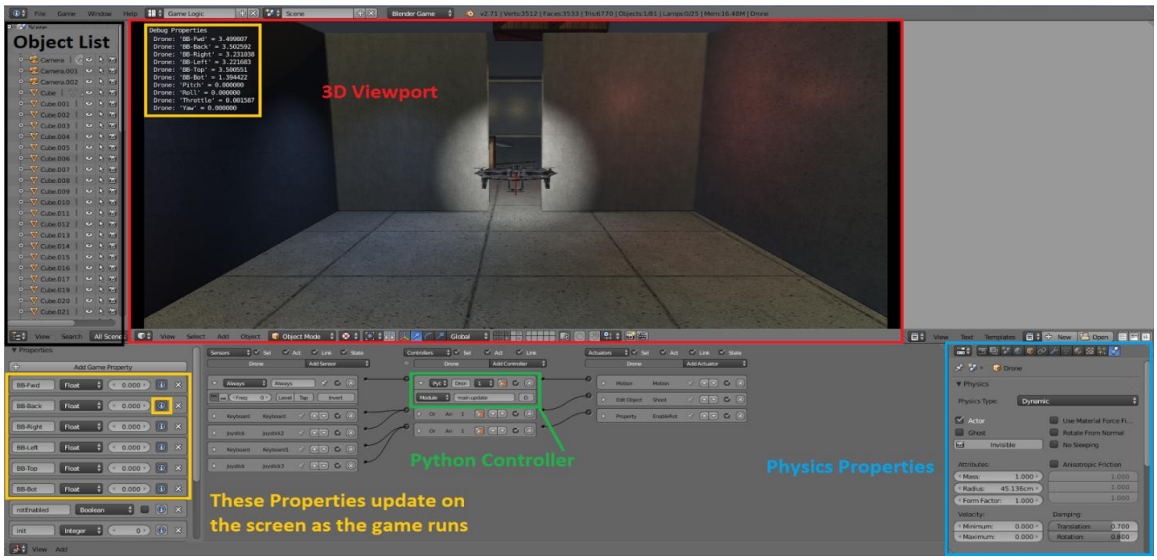


Figure A.1: BGE Interface

The game environment scene contains multiple objects, most are various obstacles. The drone is one object. The drone has an “always” sensor that is connected to a python controller. The always sensor acts like an infinite while loop that keeps calling the update function in the main file. This updates the flight controller, which propagates more updates into the sensors and input.

The drone object also has physical properties, like mass, collision radius (used by the BGE to stop the object, when it collides with other objects like the floor), and damping values. These are located in the Physics Properties of the drone.

Appendix B – Pulse Width Modulation

Appendix B details the steps taken by the team to set up the BBB for RC style PWM signals. Basic details on getting started are first presented, followed by a brief PWM discussion, and finally the onboard Programmable Realtime Units (PRU's) are discussed.

Getting Started with Beagle Bone Black

Many helpful links offer advice on getting started with the BBB, the most useful introduction being beagleboard.org [3].

Install the OS

Beagle Bone Black comes preloaded with an Angstrom image. Our group elected to upgrade to Debian 7.5, 5-14-2014. The image is loaded to a microSD card and flashed to the onboard eMMc. Details can be found at [3], while the latest Debian image can be found at [12].

PWM

The team is in the process of enabling PWM inputs and outputs. In order to accomplish this, we must use the two onboard "Programmable Real-time Units", or PRU's. As their name implies, the PRU's operate in real time, executing instructions every 5ns. The Linux OS inherently has unreliable timing for decoding RC style PWM signals, but the PRU's can decode these signals in real time.

RC style PWM signals

RC style PWM signals have a frequency of 50 Hz (period = 20 ms). A pulse of 1 ms (duty cycle = 5%) corresponds to a low input, and a pulse of 2 ms (duty cycle = 10%) corresponds to a high input. A pulse of 1.5 ms is neutral.

Four PWM inputs can then be mapped to a range corresponding to a roll (-100 to 100), pitch (-100 to 100), throttle (0 to 100), and yaw (-100 to 100). The PWM inputs and outputs will need to be appropriately mapped to the python controller.

Using the PRU's to decode RC PWM signals

This seems to be the most difficult and tedious part of the PWM. First, a process of enabling the PRU's, enabling the device tree, and decompiling/editing/recompiling device tree overlays must be completed. Details can be found at [15], [16], and [17]. These also reference the Texas Instruments examples found at [12].

After that is completed, a real-time assembly program must be written that communicates with a higher level C++ program through shared memory. An example was adapted from [15] and has been included in the repository. The PWM signals were not entirely completed, but that explains the starting point.