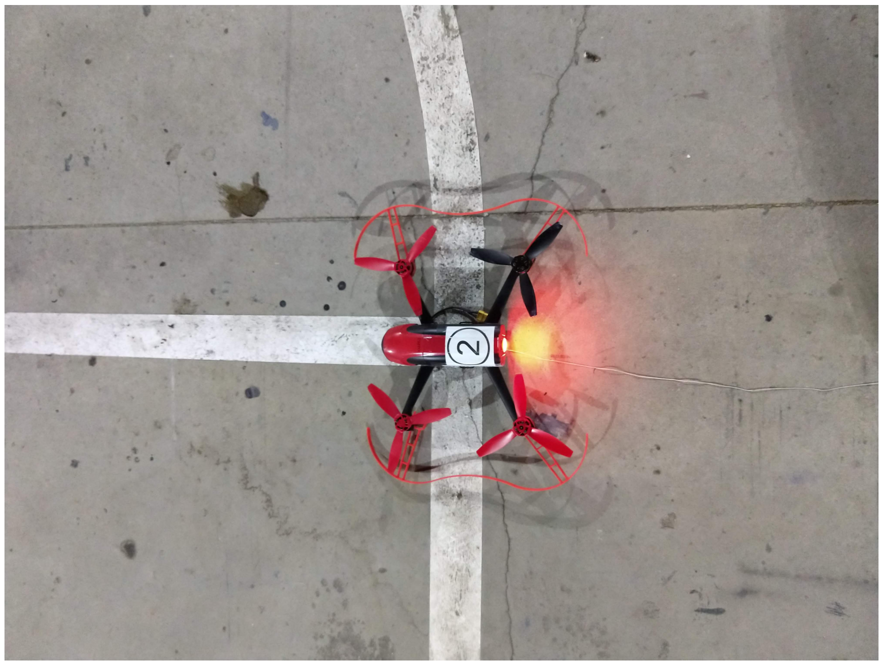
ME134 Final Project: Developing a Drone Platform for Building Observation

Nikhil Gupta, Jimmy Hamilton, Cormac O'Neill

June 2018

GitHub Repository:

`https://github.com/nkgupta1/ME134-Building-Observation`



1 Introduction

Quadcopters and similar drone platforms have rapidly increased in popularity for both consumer and industrial sectors. As a result, the market is full of robust drone platforms with various features integrated into their controllers right off the shelf. Beyond this, many platforms have open-source control software for customization of aftermarket flight controllers. This allows for various hobbyist, industrial, and research projects to be implemented by building off well-developed, open-source platforms.

In the spirit of research building off of pre-existing drone platforms, Professor Shervin Larigani has been working on a way to autonomously analyze the modal frequencies of a building during an earthquake. Since earthquake detection systems only give around 7 seconds of warning, a drone that is stationed at the bottom of building must quickly position itself for video imagery in a frame of reference that is decoupled from the motion of the building. As such, a drone offers a very ideal package to autonomously survey modal frequencies of a building during an earthquake. This would allow for the structural integrity of the building to be analyzed on short notice without influence from the moving world around the desired building. This method allows for the structural integrity of a building to be assessed purely through external sensors, bypassing the need for the costly installation of accelerometers within the structure.

The goal for this project was to have the drone autonomously take off from its base-station upon receiving notice that an earthquake was going to occur. At this point, the drone would navigate to a predefined location to record the motion of the building. For the duration of the earthquake, the drone would hold its position using an internal IMU and GPS signal while recording video. Upon completion of the earthquake, the drone would return to its base-station and land.

1.1 Parrot Bebop 2

One suitable platform for this project is the recreational Bebop 2 drone released by Parrot. This is a light quadcopter with large battery life that can take full HD video with its on-board camera. The Bebop 2's high top speeds of 13.05mph vertically and 37.28mph horizontally mean that it is able to respond quickly to an earthquake notification and move into its observation position. It is a quad-rotor and comes equipped with a suite of sensors: a downward facing camera that watches the ground to help with stabilization; a downward facing ultrasound sensor with a range of 16ft; a pressure sensor; a 3-axis gyroscope, magnetometer and accelerometer; and a GPS receiver. In order to interface with these sensors, we use the Bebop Autonomy ROS driver, developed by M. Monajjemi et al. of Simon Fraser University (`https://github.com/AutonomyLab/bebop_autonomy`). This driver allows us to interact with the Bebop's SDK through ROS, but came with its own limitations.

Bebop Autonomy provides no way of directly extracting data from most of the sensors on the drone for determining its location. Rather, the drone's position is published as an Odometry message with its coordinates determined by some combination of the individual sensors. Furthermore, while the Bebop Drone has a controller built into its software in order to navigate autonomously between GPS coordinates and hold its position, the Bebop Autonomy driver does not provide a means to interface with this controller. Rather, navigation commands sent to the drone must be given in terms of a pitch, roll angle or a vertical velocity. The constraints and capabilities of both the Bebop 2 drone and the Bebop Autonomy ROS driver serve as the framework for our project.

In order to pass this information off to Professor Larigani's group, we needed to work in Python and develop a system that they can implement with a gyroscope stabilized camera. As such, we settled on a design that could be implemented for practical use using a Bebop 2 drone with code to autonomously start, come to an observation position by a building, and maintain that position for some desired duration.

2 Methodology

In order to position the drone in a suitable position for the imaging of the target structure, it needs to be able to autonomously take off from its base-station, navigate to a pre-determined location in three-dimensional space, hover in place long enough for enough footage to be collected, and then autonomously navigate home before landing. Since the drone needs to be in position within seven seconds, the base station for the drone would have to be either on top or at the base of a building so it has minimal distance that it needs to travel to get in position. Since the desired measurement position would be near the top of a building, we envision a deployable base on top of a building that the drone can be stationed in, fully charged and ready to move into position as soon as the seismographic trigger is observed. The base would open up, allowing for the drone to launch and move to the desired position out and then below the edge of the building. This seems like the most feasible way to perform the desired building analysis.

For our part of the project, we wrote code to interact with ROS and manage a controller for the Bebop 2. We also tuned the hardware to adequately perform the desired localization task. The Bebop Autonomy driver comes pre-equipped with takeoff and landing topics. The takeoff topic moves the drone from its position on the ground to a position hovering in the air, while the landing topic navigates the drone from a position in the air to a spot landed on the ground. In order to have the drone navigate while in the air, we implemented three independent PID controllers: one each for the x, y and z directions. These controllers treated any new goal destination as a setpoint change and sent velocity commands to the drone in order to make corrections as needed. This PID controller was tuned and then used to allow the drone to navigate both to its observation position and back to its home base once observation is complete.

2.1 PID Controller

To navigate to a desired position, we wrote code that moves the robot to any desired position by simply changing the setpoints within a series of PID controllers. The drone then moves to the desired position based on the PID loops. Three individual PID controllers were implemented in order to provide navigational control for the drone, one for each of the x, y, and z directions. Each of these controllers had their proportional, derivative and integral parameters individually tuned (K_P, K_D, and K_I respectively). The PID loop functions by measuring an error between the drone's desired location and its current location, as indicated by the odometry data extracted via Bebop Autonomy, and then implementing a correction in the form of a velocity command. The corrective velocity command takes a different form for each of the controllers: for the x direction, the correction is a pitch angle expressed as a percentage of the maximum allowed pitch; for the y direction, the correction is a roll angle scaled by the maximum allowed roll; and the correction for the z direction is a factor that is applied to the maximum allowed z velocity. All of these commands therefore take a value in the range $[-1, 1]$. For this project, all of the maximum velocity parameters were left at their default values. In order to implement a PID controller in python, we used the PID code developed by Durmusoglu C. (`http://ivmech.github.io/ivPID/`) with some modifications to allow it to function in concert with Bebop Autonomy.

Within the PID code, the PID controller equation

$$u(t) = K_P e(t) + K_I \int_0^t e(t')dt' + K_D \frac{de(t)}{dt} \tag{1}$$

is discretised such that:

$$u(t) = K_P * error + K_I * integral * dt + K_D \frac{derror}{dt}$$

where:

$$\begin{aligned} error &= \text{Set Point - Current Value} \\ dt &= \text{discrete time interval between measurements} \\ error &= \text{change in error over discrete time interval} \\ integral &= \text{sum of error*dt of all preceding time intervals} \end{aligned}$$

In addition to this, Durmusoglu included a windup guard within the PID code. This term functions as a constraint on the integral term, preventing it from growing beyond a certain magnitude. This is meant to prevent the integral term from growing excessively over time, introducing a bias into the controller rather than correcting for it.

2.2 Tuning

Once the PID controller was set up, we needed to tune the values to get the desired functionality. We first started by setting just a proportional term. This acts similar to a spring constant, driving you to the desired position. We started with a random K_P value and then scaled it up or down based on response. Initially we started out with too large of a value and the quadcopter responded to errors in its position by oscillating out of control. Scaling this term, we eventually got an oscillation around the desired position that decayed over time. The oscillation was not centered around the desired position, however, so we then added K_I until the oscillations centered around the desired position. Finally, we added in a damping term K_D in an effort to critically damp the drone.

This tuning step was critical to the actual implementation of the project because we needed the drone to get into position quickly. If it was not critically damped, then getting to position would take a long time as the oscillations around the desired position would take time to level off. As such, quick and snappy control to command the drone into position was achieved such that we could accurately reach a desired position shortly after commanding the setpoint changes. Tuning for many hours obtained the response as seen in the following plots with the final parameters found in the repository.

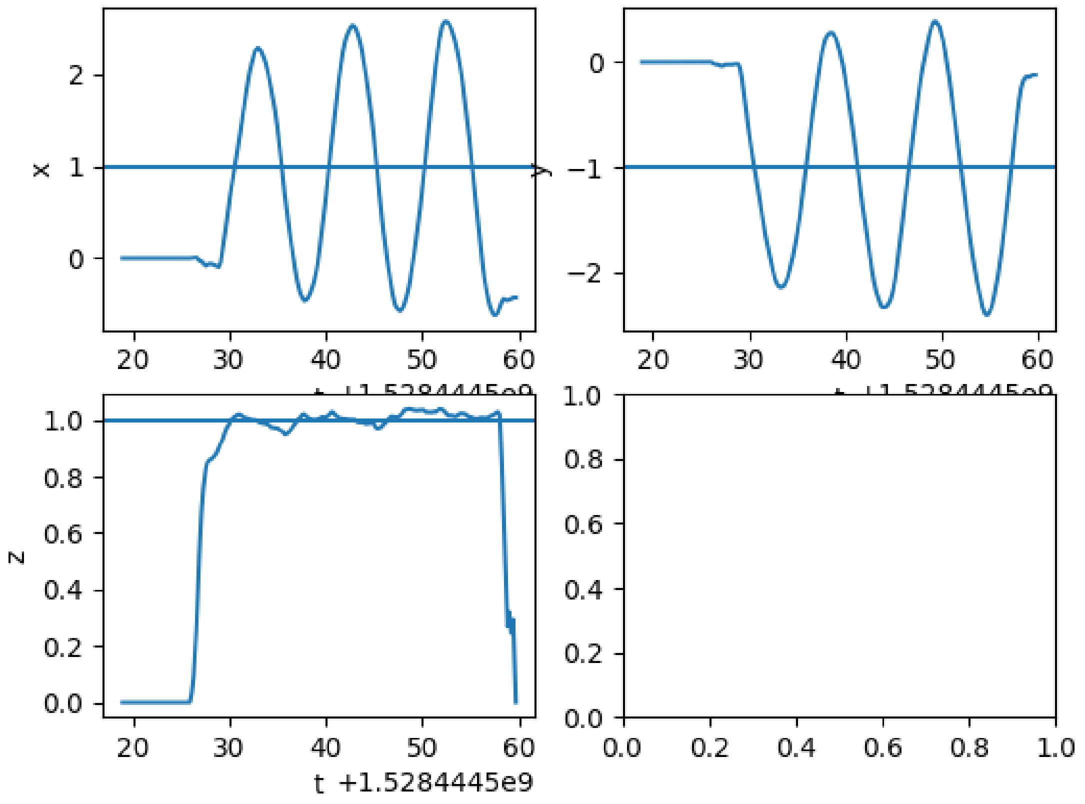


Figure 1: Plots of x, y, and z positions over time, showing how the oscillations increase in amplitude around the x and y goals. This is an indication that the K_P parameter was set too high, causing the drone to go out of control.

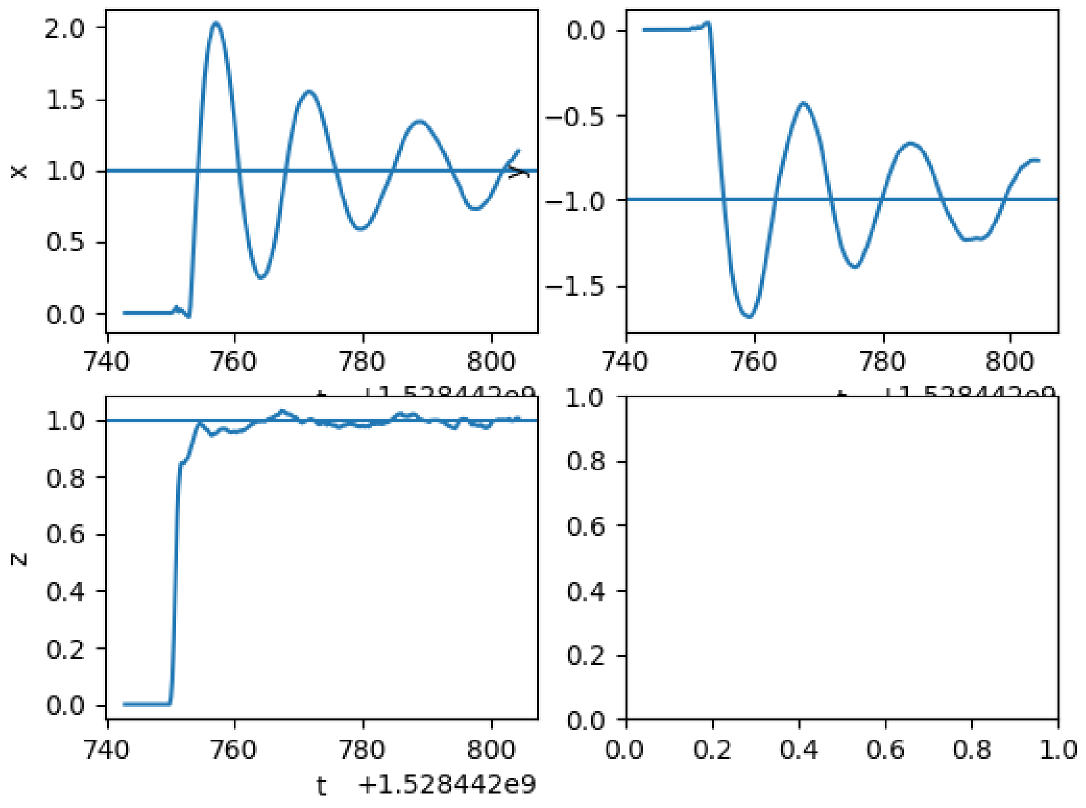


Figure 2: Plots of x, y, and z positions over time, with oscillation decreasing in amplitude around the x and y goals. This is an indication that the K_P parameter was set appropriately, allowing for the drone to approach its desired goal. However, the oscillations imply that the movement to the goal is inefficient.

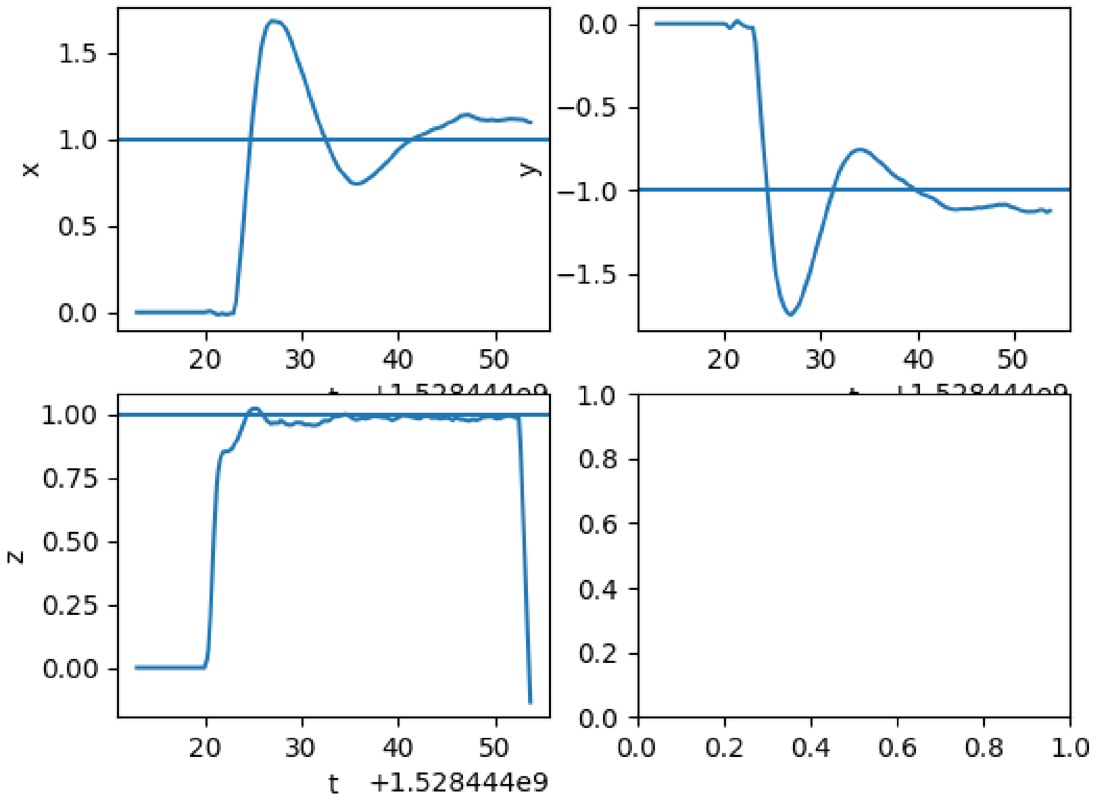


Figure 3: Plots of x, y, and z positions over time, with decaying oscillations. There is a visible bias in the final position of the drone in both the x and y directions. This indicates that a greater K_I parameter is needed.

3 Results

By the conclusion of this experiment, we were able to the build a system to achieve the full cycle of taking off, navigating to a position, holding position, navigating back to base-station, and landing. For the purposes of this scenario, the drone holds the goal position for a fixed period of time but the code could easily be adapted to return to home on some internal stimulus (like battery level or storage space) or external stimulus (like the earthquake being over). On this stimulus, the drone would set the home station as its goal position. Since the drone has such a large battery life though, it would be able to hover in position for more than enough time to document the motion during an earthquake and even following aftershocks.

The results presented here are the results from a single run and were repeatable.

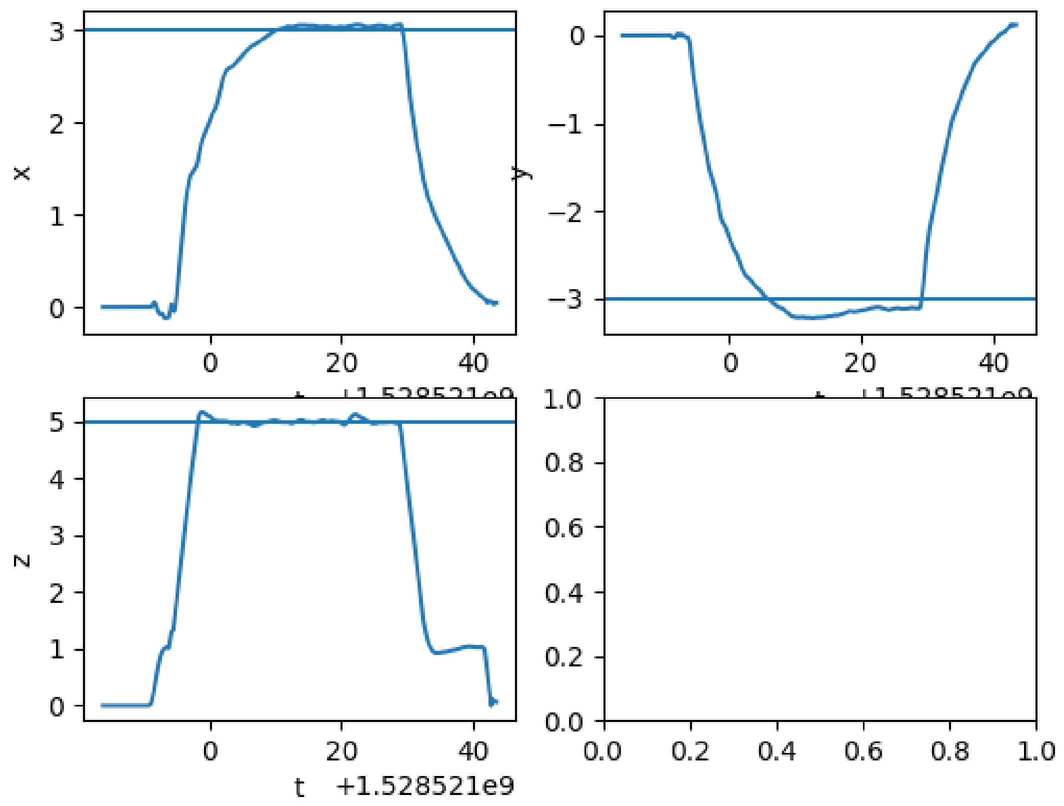


Figure 4: Plots of x, y, and z position with the horizontal line as the initial goal. Footage from the drone is available at `https://youtu.be/XGPKE_0b0aY`. Footage from an observer is available at `https://youtu.be/k2P7-vj2Sh4`.

The drone achieves the goal position and holds it for about 20 seconds. We see that the drone is able to achieve the goal after about 15 seconds of giving control to the PID controller. This time to achieve a goal of 15 seconds was relatively consistent independent of the distance to the goal on different runs. While this is longer than the 7 seconds of warning typical in an earthquake, the parameters could be further tuned to increase response and make the drone response even snappier. In particular, K_P would have to be increased to create a faster response and then the other parameters tuned accordingly to give a stable approach to the goal. In the drone and observer footage, we are able to see that the drone has significant drift even once it believes it has achieved its goal position. This is due to inaccuracies in the odometry measurement that build up over time and could be resolved by using a better IMU. As such, the drone is able to achieve and holds its position up to the odometry measurement.

In terms of visually detecting the modal frequencies, the slight movement we are seeing once we get in position would not necessarily make image processing for modal frequencies impossible. The region of interest in the building would just need to be adjusted by the offset from home as detected by the drone. Using a more precise form of localization, the registered position would let you offset your measurement location from the image to get where the actual building lies. Alternatively, data analysis of the image stack could averaged out the motion over time. This works because the drift occurs in a single direction at a smooth velocity. For instance, a wind could start blowing the drone off course, but the motion could be offset based on the average movement over time. As such, the building motion could be decoupled from any drone drift either through hardware or data analysis techniques.

4 Conclusion

After extensive tuning of the PID parameters, we were able to get the drone to navigate to the goal and hold the its position for the desired period of time. One important note regarding actual implementation is that the drone's odometry measurement encountered significant drift. In particular, the odometry would drift about 1.5 meters for 30 seconds of operation. The direction of this drift would change when the drone was re-calibrated or the trim changed. While this drift is significant for this experiment, it could easily be resolved by buying a better IMU unit or using supplemental data sources to augment the odometry measurement. In particular, we could use GPS, RFID localization, etc. to augment the measurement, which would prevent massive amounts of drift and more accurately return us to home. A different approach would be to use a computer vision approach to maintain some distance from the building until the landing sequence is initialized. A computer vision approach could also be used to land in the proper location by using a QR code to detect a landing platform.

Beyond the further work that could be done on both the hardware and software side for more robust performance, this project has many applications to structure testing around the world. Beyond just buildings, modal analysis of bridges could be performed by having a drone stationed under a bridge in a small box, similar to the kind described earlier on the top of a building. It is well known that the nation has significant structural deficiency issues, particularly with bridges. Stationing cheap drone stations under them waiting for an earthquake would allow for better knowledge of the bridges resonance and need for repair. As such, this project functions to build a framework for the development of a low-cost earthquake surveillance system that could target key structures at risk of collapse.