# Obstacle Detection via Air-Disturbance in Autonomous Quadcopters

Jason Hughes, Damian Lyons

Fordham University, Bronx NY 10458, USA
Robotics and Computer Vision Lab
jhughes50@fordham.edu

**Abstract.** Autonomous drones can detect and avoid walls as the technology stands today but they can only do this with camera, ultrasonic or laser sensors. This paper highlights how data mining classification techniques can be used to predict which side of a drone an object is located from the air-disturbance created by the drone being near such an object. Data was collected from the drone's IMU while it flew near a wall to its immediate left, right and front. The IMU includes gyroscope, accelerometer, roll, pitch and yaw data. Position and barometer data was also collected. The data was then fed to NearestNeighbor, GradientBoosting and RandomForest classifiers.

**Keywords:** UAVs, Quadcopters, Classification, Air-Disturbance, Obstacle Detection

## 1 Introduction

Autonomous flying drones are currently available, meaning the drones can fly to a preprogrammed destination while avoiding objects with no input from a user. The current object avoidance technology comes in the form of a camera, laser, or ultrasonic sensor, this study looks into if the internal sensors can be used to detect such obstacles.

The most prevalent issue that is stopping drones from mass usage in industries is their battery life. Because the drones must be light to fly they must have small batteries, and thus they have a short flight time. The previously mentioned sensors for object avoidance cut down on the drone's battery life substantially. From previous work in the Fordham University Robotics and Computer Vision lab, wind currents can be detected using a classifier [1,2]. The wind makes the drone unstable and this instability can be detected from the data gathered from the drone's internal sensors. The drone also flies unstably when it is close to large objects, like walls. This is because the wall interferes with the airflow created by the rotors of the quadcopter. This interference causes a similar instability as with a wind current.

The idea behind this paper is that the wind current created by a quadcopter interferes with the stability of the drone and that can be used to detect which side of the drone the interfering object is located. This eliminates the need for

the camera, laser or ultrasonic sensors which thus increases the battery life. The onboard computer would have to work no harder, since the data points would be gathered and sent to the home computer that is flying the drone via radio waves, and the home computer would do the calculation and prediction. The prediction can then be sent to the driver program, which can trigger the drone to avoid such an object.

This project looks at well formed objects (walls perpendicular with the ground) because this provides a good surface for the air from the quadcopter's rotors to "bounce" off of. Walls were also used because it is likely what drones will encounter when they fly autonomously in buildings. The quadcopters were flown perpendicular to the walls to collect clear data. While this project is still in its infancy, it was taken on to ultimately eliminate the need for the aforementioned external sensors to increase flight time of autonomous drones.

The drone chosen to collect the data was the crazyflie 2.0 because of its simple internal sensors and ease of programming.[7] The quadcopter has an inertial measurement unit sensor (IMU). This measures the gyroscope in the $(x, y, z)$ planes, acceleration with an accelerometer again in the $(x, y, z)$ planes. From the gyroscope and accelerometer the drone can calculate its roll, pitch and yaw angles for recording. There is also a barometer to measure air pressure. On the bottom of the quadcopter is a Flow Deck that measures the drone's $(x, y, z)$ 3D cartesian coordinates.

To gather the data the drone was flown around a U-shaped wall. Data was collected from the drone every hundredth of a second. The dimensions of the U-shape wall were 1.2 m by 1.2 m by 1.2 m. In total twelve seconds of data was collected from the left and right walls and ten seconds from the front wall, giving a total of 34 seconds of wall data. This is added to one-third of the no wall data. After cleaning the data there was a total of 30 seconds of wall data plus 10 seconds of no wall data chosen at random. This gives a total of 40 seconds of data, and with data collected at every hundredth of a second there were a total of 4,000 examples for training and testing.

## 2 Literature Review

There have been many studies done on ground effect in quadcopters. A ground effect occurs when the drone is flying near the ground and the air that is being pushed down from the rotors hits the ground and pushes back up on the drone making it unstable. This has been studied extensively to make quadcopter landing and takeoff more stable. Alternatively, wind effect in drones has not been studied very much. Drone to drone wind detection was studied in [2]. This project has one drone fly underneath the other and could successfully detect that there was a drone over top using only the internal sensors of the bottom quadcopter and a classifier. Additionally, the researchers in [1] were able to detect wind gusts using only the internal sensors of the drone. They had the drone fly in front of a fan in a different direction and built a classifier to determine if the drone was in a wind gust or not.

These two studies show that the internal sensors can be used to detect wind. Specifically, [2] shows that drones can detect wind currents created by rotors. These papers differ from this project in the respect that they do not predict which direction the wind is coming from. This is pertinent to building a truly autonomous drone that does not rely on outside sensors.

Yoon et. al. and Diaz from NASA's Ames Research Center used Computational Fluid Dynamics (CFD) to model the airflow of a drone's rotors in [5,6]. They show that the velocity of the air underneath the rotor takes the form of a cylinder. The air moves faster within the cylinder in the shape of an hourglass. This is important because it is suspected when a drone is near a wall the air underneath it's rotors is getting pushed outward and then interacting with the wall, causing an instability. Their work was done using the DJI Phantom 3 which is large and powerful. Thai et. al. [4] also did CFD for the DJI Phantom 3. Unlike the others, their work shows one instance of the drone's propeller spinning and shows that the velocity has a helical flow pattern to it. They also show that the airflow from the rotors has the hourglass shape.

Other researchers at McGill University have worked on wall detection without using additional sensors in [3]. This study has drones with rotor guards fly directly at the wall. They are looking at what speed can the quadcopter travel at, hit the wall and still recover and not crash from hitting the wall. The idea behind this study is to have an autonomously flying drone that hits walls, recovers and moves in a different direction after the collision. This project wants to avoid the collision and just use the air disturbance from the wall to fly autonomously. Also, not all drones fly with rotor guards, and without them the drone would just crash. Nonetheless, wall detection is still being worked on in the industry.

## 3 Modeling

In order to understand the air flow of the drone better both mathematical and physical modeling was done to better understand the data was collected and lead to feature selection and feature generation.

### 3.1 Mathematical Modeling

Mathematical modeling was done to simulate the airflow under the rotors of the drone. [9] Used three dimensional Navier-Stokes equations to look at a rotor wake vortex. In forward flight they calculated a spiral vortical wake geometry. The following three-dimensional Navier-Stokes equation for helical flow from [9] was solved.

$$\nabla \cdot \mathbf{u} = 0, \tag{1}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{\nabla p}{\rho} + \nu \cdot \nabla^2 \mathbf{u} + \mathbf{F} \tag{2}$$

The equation was transformed to include the curl expression represented by $\mathbf{w}$ in the following equations.

$$\nabla \cdot \mathbf{u} = 0, \tag{3}$$

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{u} \times \mathbf{w} + \nu \cdot \nabla^2 \mathbf{u} - (\frac{1}{2}\nabla(\mathbf{u}^2) + \frac{\nabla p}{\rho} + \nabla\phi) \tag{4}$$

By expanding the curl expression a system of partial differential equations was created and then solved. The variables in the equations are represented as followed:

- $\mathbf{u}$: air velocity vector containing the radial, angular and tangential velocities
- $\nabla p$: change in pressure from above the rotor to underneath the rotor
- $\rho$: air density coefficient
- $\nu$: kinematic air viscosity
- $\mathbf{F}$: force vector in $x, y, z$ directions
- $-\nabla\phi$: force potential from $\mathbf{F}$

Ershkov explains that the curl field arises from the source of vorticity in the fluid field, which in this case would be the rotor of the drone. $-\frac{\nabla p}{\rho} + \nabla\phi$ is the $x, y, z$ force vector. The system of equations gives the solution as:

$$\mathbf{u} = \exp\left(-\nu \cdot \alpha^2 \cdot t\right) \cdot \mathbf{u}(t_0). \tag{5}$$

where $\mathbf{u}(t_0)$ refers the velocities at the propeller which can be shown in as

$$\mathbf{u}(t_0) = \begin{bmatrix} u_r(t_0) \\ u_t(t_0) \\ u_z(t_0) \end{bmatrix} \tag{6}$$

$u_r, u_t, u_z$ refers to the radial, tangential, and downward velocities at the rotor of the quadcopter. The velocities can be calculated with with the equations:

$$u_r(t_0) = b \cdot \pi n \cdot 2r, \quad u_t(t_0) = r \cdot \omega, \quad u_z(t_0) = \sqrt{\frac{T/A}{2\rho}} \tag{7}$$

where $b$ is a blade swirling factor, $n$ is the rpm, $\omega$ is the induced velocity which equals $u_z$, $T$ is the force of thrust, and $A$ is the area of the rotor disk. The answers in (5) are then used in (3) to solve the equation as time continues, the resulting vectors for $u_r, u_t$ and $u_z$ were plotted as a vector field.

### 3.2 Airflow Testing

In order to validate the mathematics, airflow testing was done by looking at talcum powder in the drone's rotors. This was done to observe the helical airflow of the quadcopter's rotors and to see how the air could possibly interact with a nearby wall.

First, a drone was mounted on a rod with a camera mounted one meter away to video and photograph the airflow. Initially, a fog machine was hung upside down with the nozzle pointing downward at the top of the drone's rotors. After some initial photos and videos it was observed that the drone was not visible through the fog and thus the airflow could not be examined. Alternatively, talcum powder was dropped into the rotors from 0.5 meters above. The rotors were

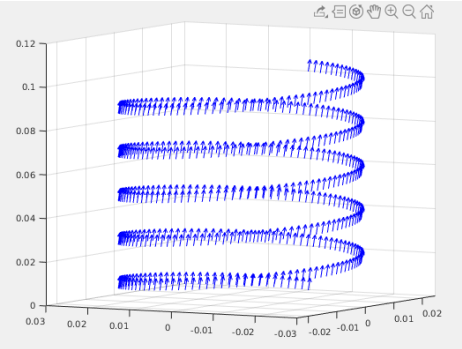



Fig. 1: Velocity Vector Field


(a) (b)

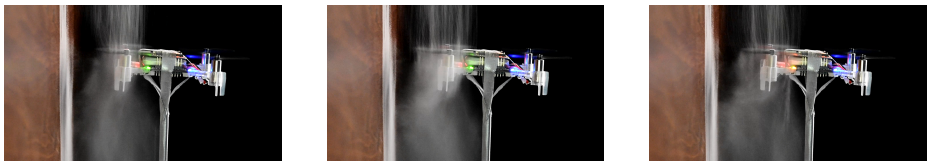
Fig. 2: Experimental setup

(a) (b) (c)

Fig. 3: Sequence of frames from video showing airflow

tested at varied thrust starting at 25% and working up to 80%. The procedure was videoed for later analysis and a snapshot from a video at 80% thrust is shown in figure 3.

The process was repeated but with a wall near the drone to examine how the airflow from the quadcopter interacts with the wall. The drone was placed two rotor diameter-lengths away from the wall, which is about 0.1m.



(a) (b) (c)

Fig. 4: Sequence of frames with wall

## 4 Experimentation

### 4.1 Wall Data Collection

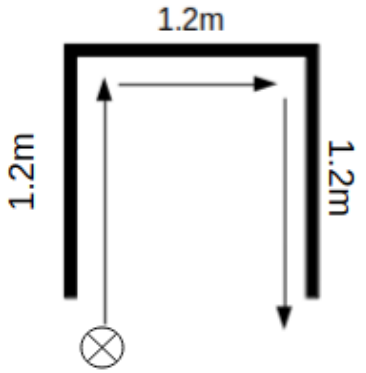
Data needed to be gathered from drones both near and away from walls. The drone used was a BitCraze Crazyflie 2.0 with a flow deck and loco position system (LPS) deck (fig. 2) [7]. The drone measures about 10 cm by 10 cm, and has total weight, including the decks, of 42g. The flow deck uses a laser sensor to determine the height of the quadcopter and the LPS deck gives off a signal to the LPS nodes in each corner of the flying area to determine its Cartesian position $(x, y, z)$.

While the drone is flown by a base computer using a python interface. In python a driver program was created to fly the drone in a U-shape. While the quadcopter is flying it sends data to the computer from its internal sensors which is then stored for later analysis.

The quadcopter was flown around a U-shaped wall within two rotor diameter lengths of the wall. The walls were constructed from particle board 1m tall by 1.2m long.It started with the wall to its left and flew towards the front facing wall in the $x$-direction. When it reached the front facing wall it stopped and moved in the $y$-direction towards the right facing wall. Once the quadcopter reached the right facing wall it moved backwards in the $-x$-direction until it reached the opening again where it landed.

While the quadcopter was flying, data was collected from the onboard sensors which includes an MPU-9250 inertial measurement unit (IMU). The gyroscope sensor has digital-output based on the $x, y, z$ axes angular rates sensors (gyroscopes) with a user-programmable full-scale range of $\pm 250$, $\pm 500$, $\pm 1000$, and

(a) Crazyflie 2.0




(b) Flight Path

Fig. 5

±2000°/sec and integrated 16-bit ADCs. The accelerometer is a digital-output triple-axis component with a programmable full-scale range of ±2g, ±4g, ±8g, and ±16g and integrated 16-bit analog to digital converters (ADCs). In addition to the gyroscope and accelerometer data, stabilizer (roll,pitch,yaw), cartesian position $(x, y, z)$ and barometer data were collected every hundredth of a second [7].The data was stored in separate files based on the wall position (left,front,right).The test was repeated five times and was repeated another time with no walls for a control group.

### 4.2 Classifier Building

In order to predict whether a wall was to the left, right or front of the drone a classifier was built using Pandas and SKlearn packages in Python [8]. The data was collected based on which wall the drone was flying near, i.e. left, front or right. This data was then divided into one-second increments.

The data was first preprocessed manually by cutting out the seconds when the drone was near the corners of the U-shaped wall leading to 30 second of flight data. To test how accurately the wall side can be predicted tests were conducted on each one-second interval, meaning an individual second would be the test set and the model would train on the twenty-nine remaining seconds and ten seconds from the control flight with no walls. The ten control seconds were chosen by selecting the middle-most data from each direction the drone was moving. This was done so the classifier would not be affected by noise from the drone changing direction. The training and testing would be repeated for each individual second to get an accuracy score at each time interval.A K-

NearestNeighbor, RandomForst and GradientBoosting classifier was then run on all the features with four distinct classes, left, front, right, and none.

Next, the same three classifiers were run using the same format as above but with only two classes. The class that was the test class was kept the same and all the data belonging to the three remaining classes was changed to 'other'. This gives the classifier an easier task of distinguishing between two classes rather than four. This leads to the problem of class imbalance. The class that is being tested will have nine seconds left for training and there will be thirty seconds for training for the non-testing class. To resolve this the testing class was duplicated until there were an equal number of examples between the testing class and other classes.

Initially a K-NearestNeighbor classifier was tested, but the results were less than optimal, so the GradientBoosting and RandomForest classifiers were also tested. These classifiers were chosen because of their ensemble capabilities, which is necessary because of the way the distinct features are affected by the walls location relative to the drone. Based on the modeling all features except for gyroscope $x$ and $y$, and position $x$ and $y$ were dropped. The gyroscope was most affected by being close to the wall and keeping the other features only adds training and testing time for similar results.

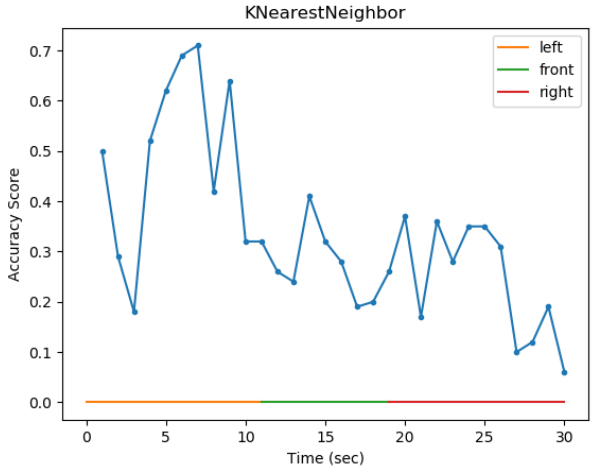
## 5 Results

First, the wall and control data was run through the classifier from the previous study by [Gu] which was used to determine if a Crazyflie drone was in a wind current or not. The classifier performed very well achieving about 99% accuracy. This means that the drone is being affected by its airflow being near the wall.

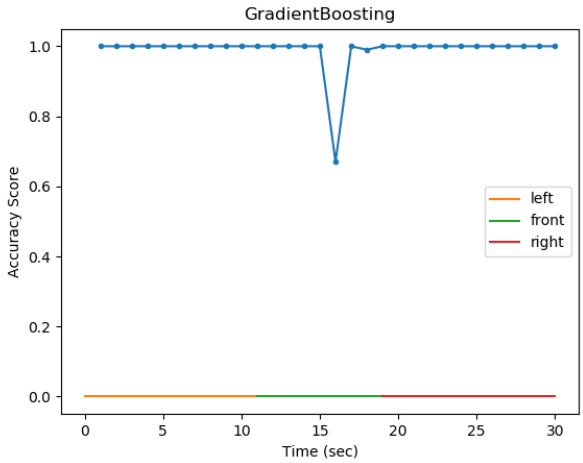
The classifier tested on one second of data and used the other twenty-nine second of data with the drone near the wall and an additional nine seconds of no wall data to train on. This was repeated for the thirty seconds of flight time around the U-shaped wall. An accuracy score was taken at each second to give an overall average of the thirty seconds of testing. This was repeated ten times to get a more accurate average of the full thirty seconds.

When testing with all the features and using four classes, left, front, right, and none, the RandomForest Classifier performed best with 98.21% accuracy followed closely by the GradientBoosting classifier at 97.47%. The K-NearestNEighbor classifier was significantly less accurate at 34.01%. The graphs in figure 6 shows the accuracy score each of the one-second intervals for the three classifiers. To complete a full test of the thirty seconds of flight time, the GradientBoosting was very slow, taking 72.14 seconds. RandomForest took 14.03 seconds and Nearest-Neighbor took just 2.67 seconds.

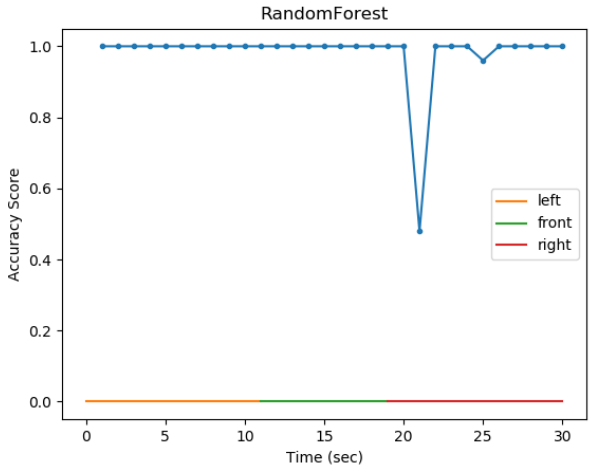
The same test was repeated but this time only the $x$-gyroscope, $y$-gyroscope, $x$-direction, and $y$-direction were used for training and testing. Averaging 10 runs of the classifier showed that the GradientBoosting was slightly more accurate than the RandomForest at 97.17% and 96.91% respectively. NearestNeighbor improved to 51.85%. The testing time dropped significantly to 2.63, 33.20, and




(a)




(b)




(c)

Fig. 6: One-Second Interval Accuracy Scores (4-classes)

8.86 seconds for NearestNeighbor, GradientBoosting and RandomForest respectively.

Next, the classifier was changed to test on a wall side while all the data not belonging to the test class was labeled as other, meaning that the classifier was only testing on two classes. When testing on the four features listed previously, the GradientBoosting was most accurate at 96.65% followed by RandomForst at 94.45% and NearestNeighbor at 77.44%. An example of one test of the full thirty-seconds is shown in figure 7. The times for testing of NearestNeighbor and RandomForest were similar at 2.62 and 9.35 seconds, while GradientBoosting was improved to 10.30 seconds. The test was repeated with all the features and the results were 58.64%, 96.24%, and 96.18% for NearestNeighbor, GradientBoosting and RandomForest respectively. These accuracy scores were an average of ten tests of the classifier (i.e. if you were to average the points on the graphs for ten different runs).

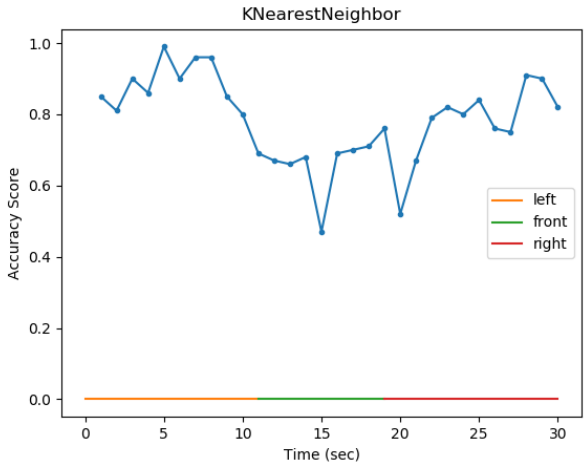
## 6 Discussion

### 6.1 Classifier Accuracy

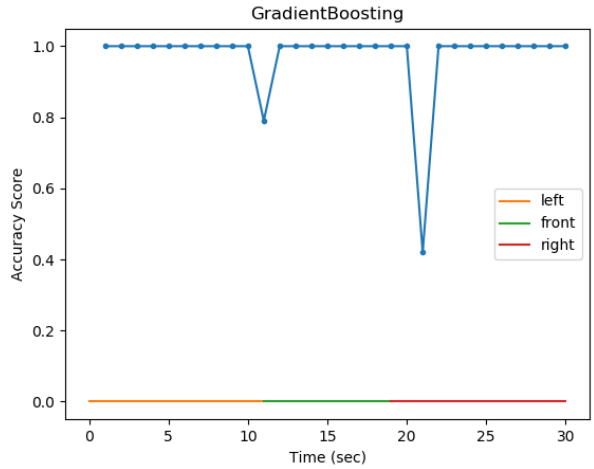
The classifier testing has two distinct results, the first is that a transition to two classes rather than four does not help the classifier, and second, twelve features are not necessary. The classifier performed very well when testing on four features using the RandomForest and GradientBoosting classifiers, showing that these are the only features that need to be used. The GradientBoosing performed almost as well on 4 features as it did on all 12 features and it did it less than half the time. This was the most accurate classifier but it was followed closely by RandomForest. This classifier was about 1.5% less accurate when testing on four classes rather than all twelve which is negligible. It did it in similar times. The only classifier that was affected by switching from four to two classes was the NearestNeighbors classifier, but it never performed accurately enough to be used practically. Overall, both the GradientBoosting and RandomForest classifiers are reasonable ones to use in an onboard classifier system. The speed of the test in real life will likely be a negligible difference and they performed equally well when testing on four classes.

### 6.2 Artificial Ground Effect

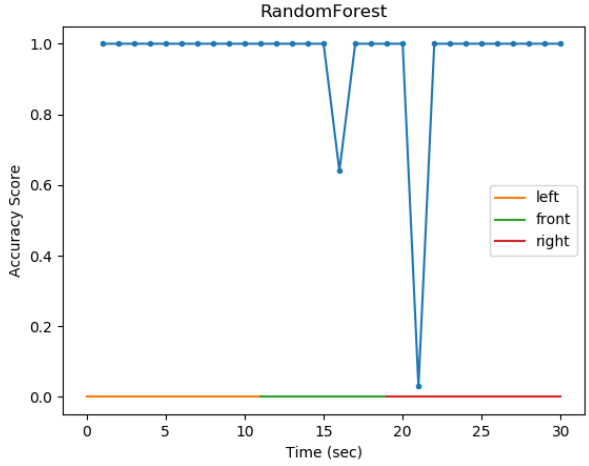
The unsteadiness of the drone occurs because of an artificial ground-effect. It is well studied that when drones are near the ground they become unsteady because the air from their rotors hits the ground and comes back at the quadcopter, thus making it unstable. In this case the air is hitting the wall then dispersing, as air does when it hits an object. This dispersion causes a mass of air underneath the drone acting as the "artificial ground". The proposition is shown in figure 8a and highlighted from the airflow imaging in 8b. Since the "ground" is only on one side of the drone there is likely a difference in gyroscope, accelerometer and stabilizer reading between a left wall, right wall and front wall.




(a)




(b)




(c)

Fig. 7: One-Second Interval Accuracy Scores (2-classes)

(a) (b)

Fig. 8: Artificial Ground Effect

### 6.3 Lab Variances

A limitation of this project was the amount of data. Only one test was completed with the drone flying around the U-shaped wall. The lab was shut down due to the COVID-19 pandemic before more tests could be completed. Ideally, there would be between five to ten tests of the drone flying near the walls. This would eliminate the need to duplicate the testing class when testing with two classes because the majority class could be undersampled if there was more data. It can also be noted that the very low accuracy can most likely be attributed to the limited amount of data. As the study stands right now data is limited so all of the data needed to be used for training and thus the testing class had to be duplicated.

Upon further investigation, the way the data was collected meant the position was unique for each wall. When the wall was to the left the drone was increasing in $x$-position, when the wall was to the front the $y$-position was increasing and when the wall was to the right the $x$-position was decreasing. When only the position data was fed to the classifier it preformed equally as well as it did when the gyroscope data was also included. When testing on just the gyroscope data the classifier preformed at about 27% accuracy. However, there is something to say about the direction the quadcopter is moving and its relation to objects. The ability to use pitch and roll angles to get a better sense of where the wall is is being worked on currently.

### 6.4 Future Work

Future work for this project includes rerunning these experiments with more data to get more accurate results. After that, data will be collected with drones at different angles relative to the wall and using a regression classifier to predict what that angle is or it could calculated from the raw roll and pitch data from the drone using Rodriguez Angles. From this angle the drone would be able to

calculate and turn, or do a feature transformation within a classifier to detect which side the wall is on. Future work will also include feature generation from the mathematical model in this paper for wall detection. Lastly, a driver can be built implementing a classifier to predict if the drone is near a wall while it is flying. This study highlights the early steps to reaching full autonomy in quadcopters without using camera or ultrasonic sensors to avoid objects.

## References

1. S. Gu, M. Lin, T.-H. Nguyen and D. Lyons, "Wind gust detection using physical sensors in quadcopters," in arXiv:1906.09371 [cs.RO].
2. Q. Zhou, J. Hughes, and D. Lyons, "Drone Proximity Detection Via Air Disturbance Analysis," SPIE Defense + Commercial Sensing 2020 Digital Forum
3. F. Chui, G. Dicker, I. Sharf, " Dynamics of a Quadrotor Undergoing Impact with a Wall" Int. Conf. Unmanned Aircraft Sys, Arlington VA, June 2016
4. A. Thai, R. Jain, S. Grace, "CFD Validation of Small Quadrotor Performance using CREATE-AV Helios" Vertical Flight Society 75th Annual Forum & Technology Display, Philadelphia, Pennsylvania, May 13–16, 2019.
5. S. Yoon, H. Lee, T. Pulliam, "Computational Analysis of Multi-Rotor Flows" NASA Ames Research Center, Moffett Field, California 94035.
6. P. Diaz, S. Yoon, "High-Fidelity Computational Aerodynamics of Multi-Rotor Unmanned Aerial Vehicles" AIAA SciTech Forum 2018, Kissimmee, Florida
7. Bitcraze Crazyflie 2.0, www.bitcraze.io/products/old-products/crazyflie-2-0/, Accessed May 5, 2020.
8. Scikit-Learn, https://scikit-learn.org/stable/, Accessed April 20, 2020.
9. S. Ershkov, "Non-Stationary Helical Flows for Incompressible 3D Navier-Stokes Equations", Applied Mathematics and Computation 274 (2016) 611-614.