WINDRONE: Gesture-Controlled Flight of an AR Drone

Tae-Min Kim, Nikhil Shenoy, Felix Yeung, Shu Xu

Advisers: Roy Yates and Ivan Seskar
Course: 14:332:492 Special Problems/Independent Study
Rutgers University Department of Electrical and Computer Engineering
5/23/2014

Abstract

In this study, we create a new interface to decrease the difficulty of controlling the Parrot AR Drone, a radio controlled flying quadcopter helicopter. Rather than using the default iOS/Android application, we enable a simpler, intuitive, and natural control module: hand gestures. Taking advantage of new technology called the Leap Motion Controller —a device which is able to sense the user's hands and fingers —we create a system for users to send instructions directly to the AR Drone based on the positioning of their hand. Although originally intended for users to be able to more easily control AR Drones, this study may also be used to further study human-computer interaction as well.

Introduction

A Parrot AR Drone (Figure 1) is a radio-controlled flying quadcopter helicopter, which receives flight instructions via a Wi-Fi connection. The user interacts with the drone by sending it instructions from an Android or iOS device. All a user has to do is connect to the Wi-fi hotspot generated by the AR Drone and issue commands via the interface on the mobile device. The AR Drone is capable of executing instructions such as autonomous take-off and landing, movement along the coordinate plane, rotation to its left and right, and hovering in place. With all of this functionality, these drones may be used for many different purposes, such as delivering packages or using its built-in camera to record videos and take pictures. The default interface which the developers of the Parrot AR Drone designed for controlling the quadcopter is an application that can be run on mobile devices that support either iOS or Android. Using this interface, however, proved to be very difficult to control the quadcopter properly when tested.

The Leap Motion Controller (Figure 2) is a small, USB-sized device that is capable of tracking both hands of a user as well as all 10 fingers with great precision and speed. The controller contains LED lights and camera sensors which are constantly scanning a hemi-spherical volume with a diameter of 1 meter and centered on the device, recording information, and sending data to the computer. The controller's software contains listeners which are able to recognize different hand gestures, such as circling, swiping, key tapping, screen tapping, and other movement of the user's hand. Most importantly

for this study, the controller outputs data indicating the position of the hand, including the hand's roll, pitch, and yaw with respect to the device's sensors. With an easy-to-use application programming interface (API), this controller allows developers to receive information about a user's hands and fingers and analyze/manipulate this information as they wish.

In this study, the Leap Motion Controller is integrated with the Parrot AR Drone, creating a new interface for users to control the drone's flight. This newly developed interface strives to provide users with a simpler, more stable, and more natural flight control experience. Our interface allows users to control the AR Drone using basic hand gestures that are interpreted as flight instructions. Stabilization algorithms are also implemented to clearly define the user's hand gestures, allowing the interface to easily interpret the gestures as commands. After receiving input from the user, our interface analyzes and manipulates this information before sending instructions to the AR Drone. Our system block diagram outlines this in Figure 6. In Figure 7, our system flow chart is shown. This flow chart displays the decisions that the system makes based on the inputs it gets from the Leap Motion. Initially, the system checks to see if a hand is present. If it is, then the moving average algorithm will execute. Once that algorithm completes, the Drift Test algorithm executes. These two algorithms are part of the "Stability Algorithm" section of the Block Diagram in Figure 6. Once those two algorithms are completed, then the decision tree is entered so the system can figure out what command to send to the AR Drone. The remaining decision structures in the Flow Chart belong to the "Translate data into instructions" section of the Block Diagram. Once all the decisions have been considered, then the Leap Motion delivers a new set of frames and the Flow Chart goes back to the "Start" step.

Since the AR Drone can move in the same directions as any normal aircraft, it can be described using the same terminology that is used for airplanes and spacecraft. The terminology, called "flight dynamics", allows us to isolate different types of movements and directly pair them with hand gestures provided by the user. We describe the requisite terminology in the next section.

Figure 1: The Parrot AR Drone 2.0

Figure 2: The Leap Motion

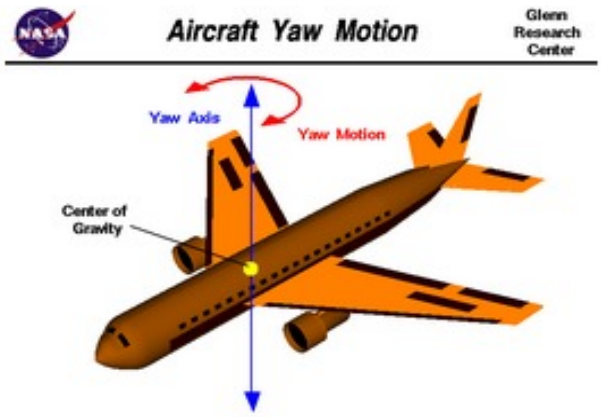


Figure 3: Example of Yaw

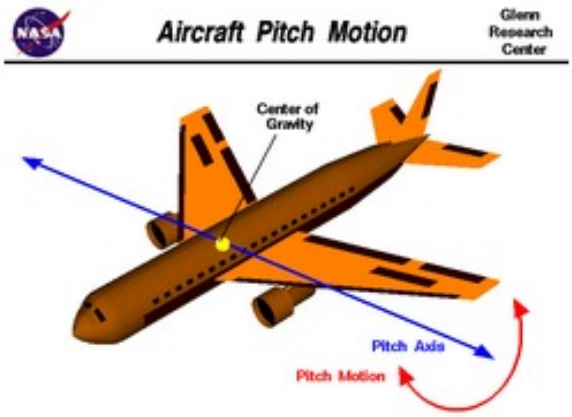


Figure 4: Example of Pitch

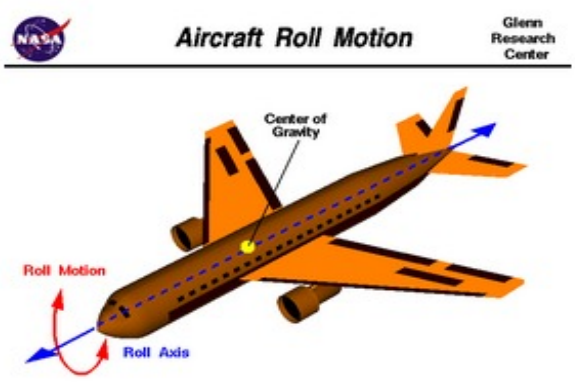


Figure 5: Example of Roll

Introduction to Flight Dynamics

Flight dynamics depend on the angles of rotation in three dimensions around a vehicle's center of mass. These include yaw, pitch, and roll. Note: all references to "clockwise" and "counter-clockwise" are described from the user's perspective of his hand.

- Yaw: left or right rotation with relation to the normal axis of the center of mass of the object.
- Pitch: rotation along the lateral axis of the object. In the example of a plane, the lateral axis is measured from wingtip to wingtip.
- Roll: rotation along the longitudinal axis of the object. In the example of a plane, the longitudinal axis is measured from the plane's head to its tail.

Language Selection

The selection of software environment was based on the performance requirements of the study, ease of implementation, and development speed. Python was chosen as the best language to use because of the availability of toolsets

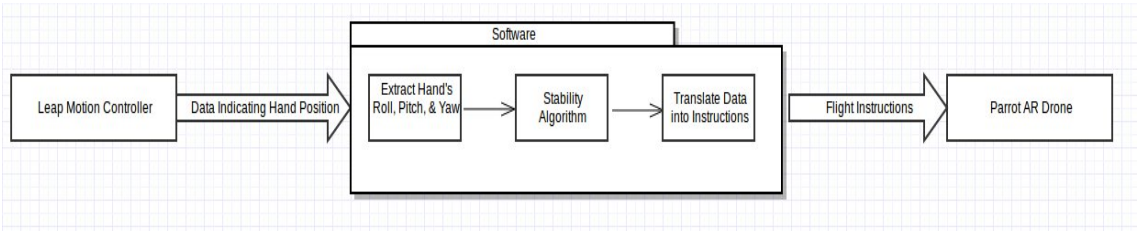


Figure 6: System Block Diagram

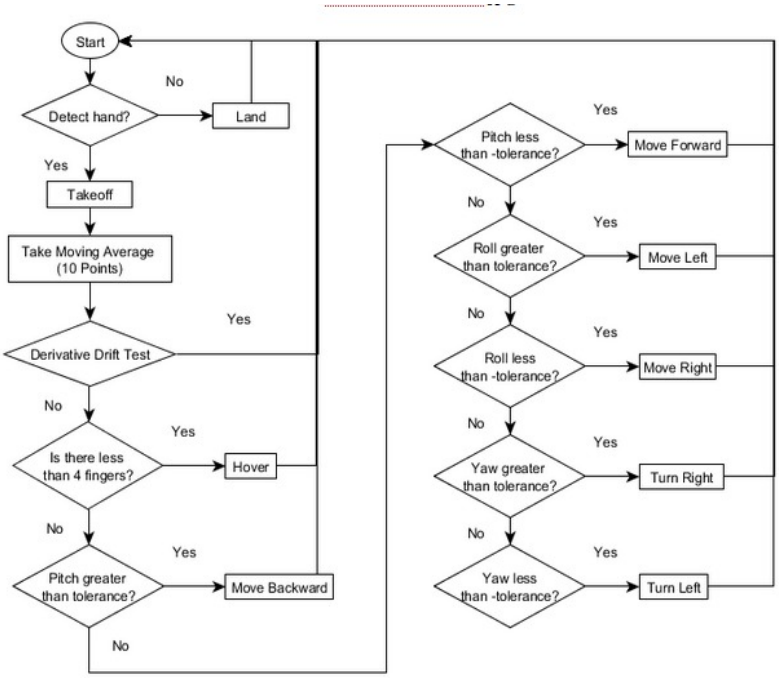


Figure 7: System Flow Chart

as well as open source libraries concerning leap motion data and sending instructions to the drone. The leap motion library packages raw Leap Motion data into a python object called a "frame"[9]. The library that helps send instructions to the drone wraps raw hex instructions into Python objects. We found that Python's execution speed is within an acceptable range of responsiveness, allowing the pilot to maneuver the AR Drone without any observable delays. After investigating a few options such as Java and C++, we decided to implement the Drone interface in Python.

Gesture Selection

Six basic gestures were used to create the interface for the Drone. These are the left and right positions, the forward and backwards positions, and the rotate left and rotate right positions. These positions were selected since the user can move his hand in the same manner as the drone can fly. In other words, there is a one-to-one correspondence between our six defined gestures and the flight patterns of the Drone. In addition, the gestures are not difficult to form and are intuitive for the user. Each of the positions consists of holding the hand flat with the fingers held close together, and then rotating or tilting in the six directions to generate our defined states. The gestures for the study are shown and defined below.

- Gesture: a placement of the hand in the space around the Leap Motion. Defined by degrees of rotation of the hand from a standard placement. All gestures recognized by the system should imitate a position similar to that of the hand when placed flat on a table.
- Drone interface: set of mappings of hand gestures to the AR Drone's flight dynamics which is presented to the user.
- Mean (Hover,neutral) position: State defined with the plane of the palm parallel to the ground.The normal vector from the plane of the palm should be pointing towards the and should be perpendicular to the ground. The hand should be held approximately 10 centimeters above the Leap Motion.
- Left position: State defined with the normal vector of the palm rotated counter-clockwise along the longitudinal axis by greater than 4 degrees measured from the mean position.

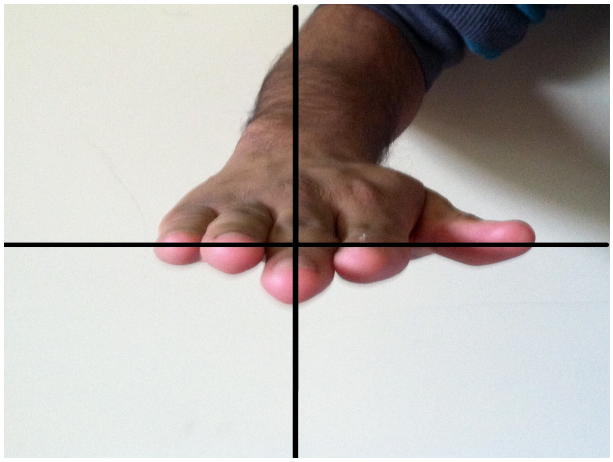


Figure 8: The Mean Position

- Right position: State defined with the normal vector of the palm rotated clockwise along the longitudinal axis by less than -4 degrees measured from the mean position.
- Forward position: State defined with the normal vector of the palm rotated clockwise along the lateral axis by greater than 6 degrees from the mean position.
- Backward position: State defined with the normal vector of the palm rotated counter-clockwise along the lateral axis by less than -6 degrees from the mean position.
- Rotate left position: State defined with the normal vector of the palm rotated counter-clockwise along the normal axis by greater than 12 degrees from the mean position.
- Rotate right position: State defined with the normal vector of the palm rotated counter-clockwise along the normal axis by less than 12 degrees from the mean position.

Overview of Approach

The system in this study consisted of three distinct components; the Leap Motion device, our own processing software, and the interpretation by the

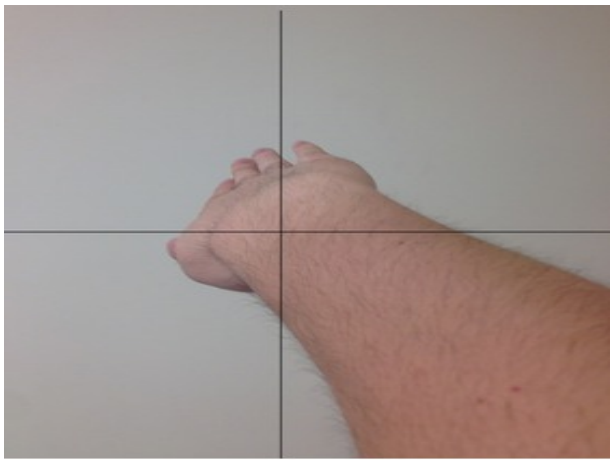


Figure 9: The Left Position

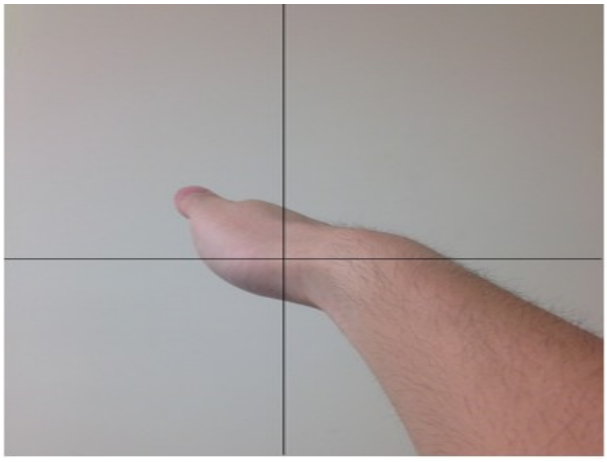


Figure 10: The Right Position

Figure 11: The Forward Position

Figure 12: The Backward Position

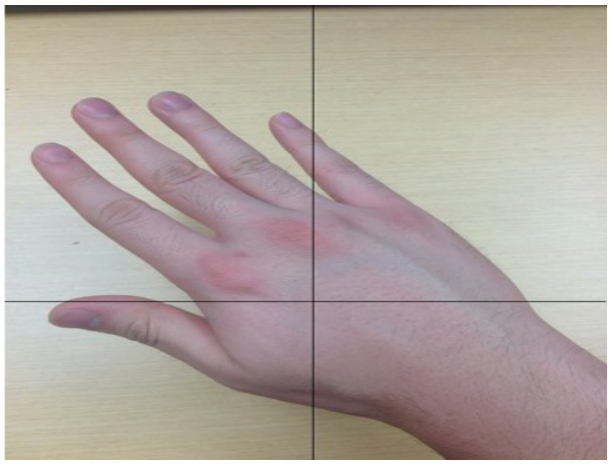


Figure 13: The Rotate Left Position

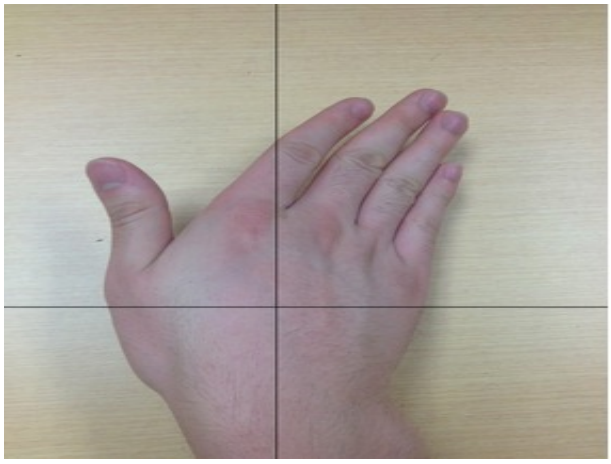


Figure 14: The Rotate Right Position

AR Drone. As the block diagram in Figure 6 shows, the software that we wrote acts as a controller between the Leap Motion and AR Drone devices and uses a series of steps to transform the data generated by the Leap Motion into commands that the AR Drone can execute.

Our software consists of three subsections; isolation of the data about the three flight dynamics of the hand, stabilization of the three sets of data and recognition of different hand states, and a decision structure about the command to be sent to the Drone. The process is started by turning on the Leap Motion and running our software. The Leap Motion begins collecting data about hands in its area of detection and begins creating frames of data. Next, the Leap forwards the frames to the controller software, which unpacks the roll, pitch, and yaw data. The stabilization and recognition algorithms are run on the received data, and then the decision tree in Figure 7 is traversed to decide what control message to send. Finally, the controller software forwards the control message to the AR Drone, which understands what flight pattern is desired, and the drone flies in the appropriate direction. Once the command is interpreted, then the Leap Motion collects a new frame of data. If no hand is detected, the Leap will cause the drone to land and will continue to receive frame data. When the hand is detected, the analysis of its position begins again. This process terminates when the user issues an interrupt signal using Control-C.

Data Collection

Categorization

Data is collected from the Leap Motion Controller using Leap Motion's Software Development Kit (SDK). This SDK contains many listeners which receive information from the sensors in frames, which are snapshots of the hand taken at specific intervals. Listeners are software objects that execute certain instructions once an event occurs. For example, a motion-sensor can be thought of as a physical version of a listener. The sensor will continually wait until movement occurs; once the movement does occur, the sensor will light up. Thus, the motion-sensor "listens" for movement. For our study, the Leap Motion was set to generate 300 frames of hand data per second. The SDK provided us with several listeners that collected different types of data, and we modified those listeners to collect the attributes we were looking for. The

listeners most relevant to the system were **on_connect()**, **on_frame()**, and **on_focus_gained()**.

On_connect() could be used to send commands to the drone as soon as the script connected to the Leap Motion. However, this listener was discarded because it would continuously send messages to our script even when frame data is not present. The **on_focus_gained()** listener allows the caller to send commands every time the window where the script is running gains the focus. This listener was also discarded because the stream of commands to the script could be interrupted if the focus on the window is lost. The remaining listener, **on_frame()**, allows the caller to execute commands every time that the Leap Motion receives a frame of data. This listener was chosen because it enables the system to respond only when relevant data has been gathered. In the other two listeners, the execution of commands is dependent on either another factor such as focus, or the maintenance of a connection. The use of these two listeners would slow down the system in two different ways. In the case of **on_connect()**, the system would get bombarded with control messages which may or may not contain frame data. Having the system unpack messages which do not contain any relevant information is a waste of resources, and results in less efficiency. The **on_frame()** listener is inefficient because data is only transmitted as long as the focus on the window is maintained. The user can make any gesture he wants, but those commands will never get sent or processed if the computer's focus on the window is lost. Thus, having the system depend on the focus is unpredictable and inefficient, so we discarded it. Due to the drawbacks of the **on_connect()** and **on_frame()** listeners, the **on_frame()** listener was chosen to receive data from the Leap Motion.

Once the script connected to the Leap Motion, it was able to receive and parse the frame data that was being sent from the device. Each frame is designed to contain information about the roll, pitch, and yaw of the hand in degrees. The degree measurements in each case are taken as the angle measured between the normal vectors of an arbitrary position and the mean position. A sample frame is shown in Figure 15.

Once the frame data is retrieved, we parse each frame for the degree measurements in each of the six positions shown in Figures 8 through 14.

The frame structure shown in Figure 15 was used to collect data about the six positions defined for our system. Each position was held for a period of 10 seconds, and the degree measurements for each flight dynamic were recorded from the frame data. We know that the human hand does not stay absolutely

```
Hand sphere radius: 96.995811 mm, palm position: (115.586, 87.1491, 32.8854)
Hand pitch: -5.981690 degrees, roll: 6.394856 degrees, yaw: 11.320604 degrees
```

Figure 15: Sample frame for the Move Right gesture. Different characteristics of the hand are recorded in this format, and the designer must parse through this frame to use the roll, pitch, and yaw data.

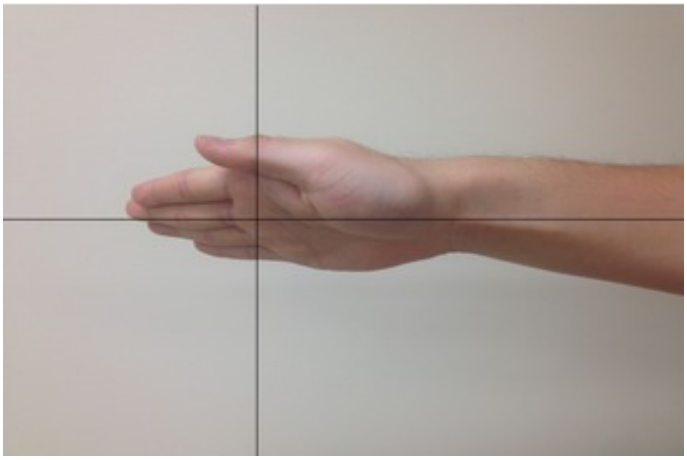


Figure 16: Side view of the Move Right gesture

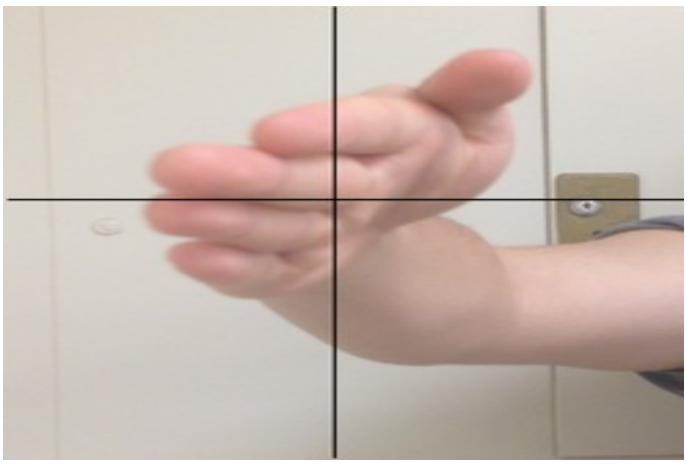


Figure 17: Front view of the Move Right gesture

still when held in a position, so degree measurements were taken over a period of time to observe how much variation the hand produced. A set of frames were generated during the 10 second period of holding a particular position, and this set contained all the degree measurements taken by the Leap Motion over the 10-second time period. Each frame from every set was parsed and the measurements for the intended flight dynamic were placed in an array that is passed to to the Moving Average algorithm. For example, all the roll measurements from the Right position were parsed from the frames because the intended flight dynamic for that position is the roll. Similarly, we parsed the frames for all the pitch measurements from the Forward and Backward positions because the pitch is the intended flight dynamic. We continued this process for the remaining gestures to collect degree measurements for the rest of the states. Once all the degree measurements were correctly parsed from the frames, the data set was passed to the stabilization and recognition algorithms for processing.

Stabilizing Flight

Each state is defined by an offset of specific degrees from the mean position of the hand, depending on the desired state. The mean position is defined with the hand hovering directly over the Leap Motion with the palm facing towards the floor, as shown in Figure 8. Move left and move right, which are shown in Figures 9 and 10, are determined by the offset of roll from the mean position. Move forward and move back, shown in Figures 11 and 12, are determined by the offset of pitch from the mean position. Finally, turn left and turn right are determined by the offset of yaw from the mean position and are demonstrated in Figures 13 and 14. For roll, the degree measurements are made relative to the rightmost part of the hand. Positive degree measurements are measured with the unit circle as a reference. The measurement for pitch is analogous, but the fingertips are used as the reference. Finally, yaw is also measured using the fingertips, with zero degrees being the direction directly in front of the user.

After collecting some degree measurements for each state, we conjectured that the variations in position could be caused by two types of involuntary hand movements: Hand Shaking and Hand Drifting. Hand Shaking occurs because it is impossible for the pilot to keep his hand completely still, so there will always be some degree of "shaking" over the Leap motion. Shaking is characterized by slight rotations of the hand around a particular gesture

4.178198324,4.083621975,4.017895492,4.040072689,4.030165926,

Figure 18: A sample string of comma separated values. This values were used in conjunction with MATLAB to test for normality.

in all three flight dynamics. The rotations are very swift and tend to reverse direction quickly. During these rotations, the hand is moving fast but does not move very far from our desired hand gesture. Hand Drifting occurs when the pilot is focusing so much on the AR Drone that he does not notice that his hand seems to be rotating slightly, resulting in unintentional movements in the drone's flight. A characteristic of drifting is slow rotational movement of the hand that deviates away from the mean position; in other words, the pilot's hand will deviate to an unintentional flight instruction - for example, a pilot's hand might initially be in the "move left" state but gradually move toward the "move right" state without the pilot being aware of this shift. The system must account for this unintentional movement caused by human error and understand that it should not recognize a different state because of it.

To further understand Hand Shaking, we collected sample sets of degree measurements from the Leap Motion for each defined state. 3000 samples were collected from each gesture over a period of 10 seconds for each of the desired positions, and the results were stored in a text file using the comma-separated-value (CSV) format.

We picked the CSV format for our data, because of its ease of import into MATLAB, a high-level language for mathematical analysis, numerical computation, and data visualization. We used the different tools and functions of MATLAB to understand our CSV data, an example of which is shown in Figure 18. Degree measurments from the frames were written into a text file one after another for ease of manipulation in MATLAB. Once all the data was collected, a simple **csvread()** command was used to read the data into a matrix. This matrix was used with MATLAB's statistical tools to test for normality.

The goal in eliminating the "shaking hand" problem was to match the Leap Motion data to a particular probability distribution, and then utilize that distribution to remove any outliers from our data set. Once the outliers

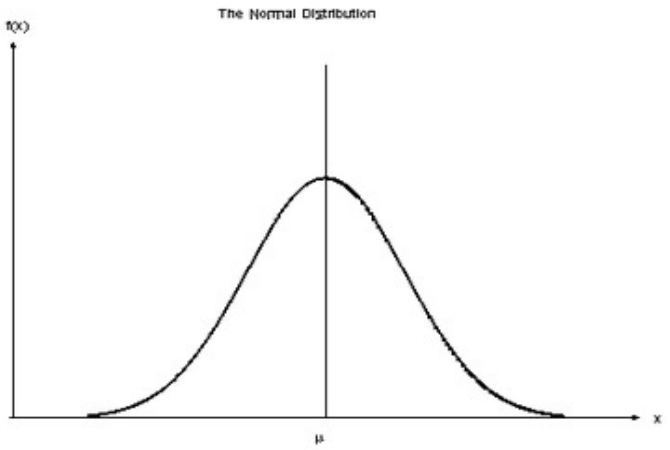


Figure 19: The Normal Distribution

were eliminated, we marked certain points in the distribution within which a state could be recognized. Our initial hypothesis was that the data collected will be normally distributed, such as in Figure 19. The normal distribution plots the frequency with which a specific data point is measured, and it describes a set of data that tends to a central value, and no bias to left or right [3].

Using a normal distribution to analyze our data presents several advantages. First, the distribution is characterized by a mean value and a standard deviation from that mean value. The mean value signifies the intended degree of the pilot's hand, and the standard deviation is the offset caused by the shaking of the pilots hand. For example, if the pilot intends to move right, the pilot will move his hand past the tolerance of the hover state towards the roll state. Using the data measured from this new position of the pilot's hand, the drone will move to the right with a velocity which is calculated using the roll degree of the hand. However, due to the shaking of our hand and the noise of the Leap Motion, measurements that deviate slightly from the mean will also be recorded. These variations are frequent and are localized around the mean position, because it is very unlikely that large degrees of variations will be recorded. However, as one moves further from the mean position, the frequency of the variations decrease. This occurs because the probability of a great variation being recorded diminishes as the degree value is increased or decreased. If the hand is supposed to be held at 10 degrees, then the majority of the measurements will fall within a

range of 5 to 15 degrees. In this case, a variation of 100 degrees from the mean position is very unlikely, and the normal distribution can predict this by showing a small number of measurements taken at 100 degree variation from the mean. Thus, the degree of variation from the mean continually decreases as one moves further and further away from the mean value, which is an expected behavior of this distribution.

Second, we can use the standard deviation of the distribution to determine how widely distributed the data set is. The standard deviation quantifies the notion of "hand shaking" because it shows the frequency of variations from the mean position. With this characteristic, we can decide how much of the data we would like to include in our analysis. We can use a specific multiple of the standard deviation to create threshold values between which data can be recognized as significant. For example, we can decide to set the thresholds at half a standard deviation on either side of the mean. This means that any data point that is recorded between those two threshold values, say 40 degrees and 50 degrees, will be recognized as significant. Using the standard deviation, we set certain threshold values for each gesture such that any values that fall within that range will correspond to the correct gesture.

The thresholds are used to mark the maximum variation that a particular state accepts. If the hand is positioned outside of the threshold value for enough time, then the system will not recognize the gesture. Since the gestures are uniquely defined by the threshold values, the hand gesture can be uniquely mapped to a flight pattern. Once the gestures get mapped, a simple decision structure can decide what command gets sent to the AR Drone.

As an initial test, we graphed our data using histograms to see if the shape of the data seemed to close enough to the normal distribution. A histogram is a graphical representation of data which tabulates the frequency of data points over intervals of adjacent rectangles, otherwise known as bins [5]. We created histograms for every position, and found mixed results.

Figures 20-26 show the histograms for the different positions. The basic criteria that we used to find Gaussian shaped distributions were the presence of a single, distinct mean value and a bell-shaped curve that eventually diminished to zero. Using this criteria, we found that some of the histograms closely resembled the normal distribution, while others were only somewhat related. For example, the histogram for the left position appeared to be centered at 31 degrees, but had data points skewed to the left. It did have two tails that diminished to zero at around 28 and 32 degrees, but the sizes of these tails were also affected by the negative skewness. However, the distri-

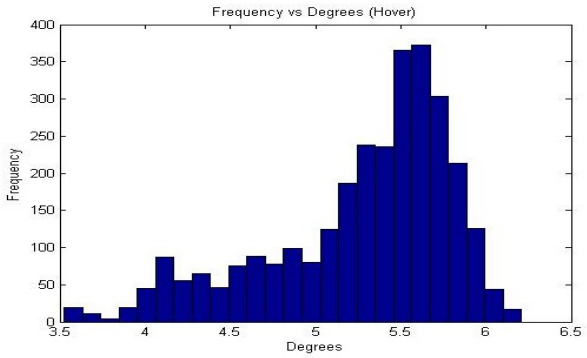


Figure 20: This is the histogram of our hand in hover position. Due to the nature of this state being the easiest to maintain, we feel that this is the most accurate sample for "drifting". It looks to be a Gaussian distribution with a negative amount of skewness, an increased amount of kurtosis, and a pronounced mean at 5.5. The tails diminish the further the sample point is away from the mean value.

bution of the data was similar to the bell-shape of the Gaussian distribution and had a prominent mean, so this set strengthened our hypothesis. On the other hand, the Forward data set showed a distribution very different to the normal distribution. There appeared to be four different peaks in the frequencies, with three of the peaks clustered within a range of four degrees and centered at approximately -108 degrees. These four peaks were unexpected, and their placement in relation to the degree measurements was too erratic. We could not confidently say that the data for this set could be described by a Gaussian distribution. The trend in the histograms showed that each sample set would show either one prominent mean or three to four less prominent means. Due to this trend, we could not confidently conclude that each of the samples could be approximated as a normal distribution. We also concluded that histograms are not an entirely reliable test of normality. An appearance similar to a particular distribution is not enough, since it could be modeled by a function similar to but not exactly the same as that for a bell curve. Unless we are absolutely sure that the data set could be matched to a distribution, we cannot make any conclusions based solely off of a histogram for the data set. Thus, we turned to more rigorous, mathematical tests to confirm the normality of our data.

The data generated by the Leap Motion is filtered by smoothing algo-

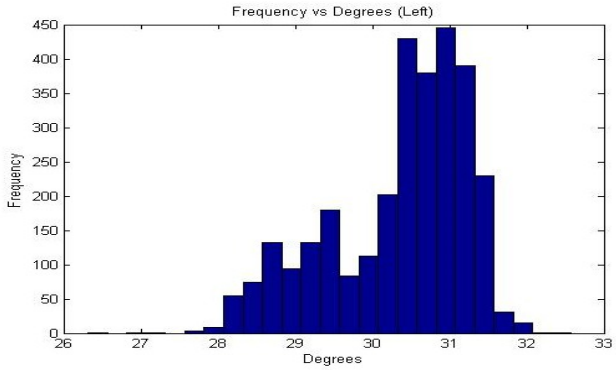


Figure 21: This is the histogram for our hand in the move left state. It is only supposed to take into account the degrees for roll, therefore the data in this histogram is a graphical interpretation of roll in degrees. As can be seen, the data's mean is around 31 degrees, and diminishes on the tails.

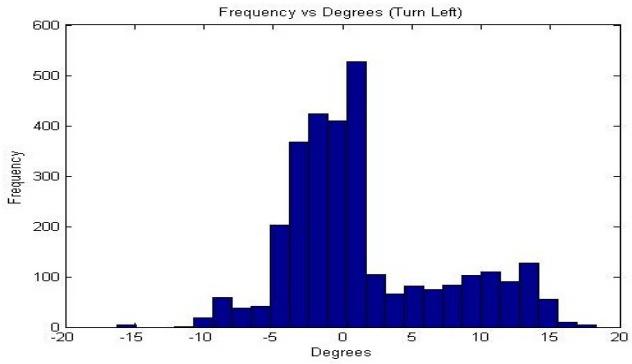


Figure 22: This is the histogram for our hand in the turn left state. It is only supposed to take into account the degrees for yaw, therefore the data in this histogram is a graphical interpretation of yaw in degrees. The data's mean is around 0 degrees, and diminishes on the tails. Movement-wise, yaw is the most difficult for the pilot and is likely to generate shaking noise. From our data it can be seen that the range of error is bigger than the other frames of different positions.

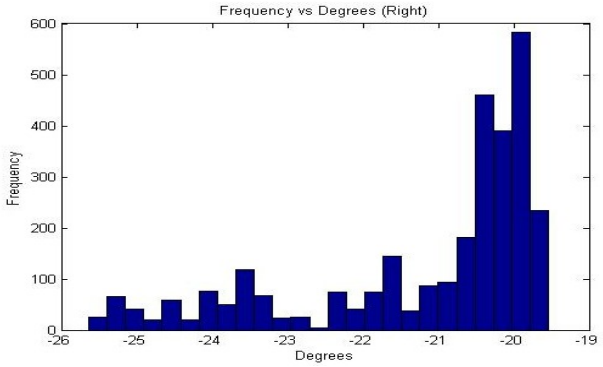


Figure 23: This is the histogram for our hand in the move right state. It is designed to account for the degrees for roll, therefore the data in this histogram is a graphical interpretation of roll in degrees. As can be seen, the data's mean is around -20 degrees, and diminishes on the tails. Our data is skewed towards the right of the sample set. Thus the pilot's hand is shaking towards the right. Since the Leap has picked up degree measurements that are more negative than -26 degrees, we know that the device does not limit values that are less than -21 degrees or so.

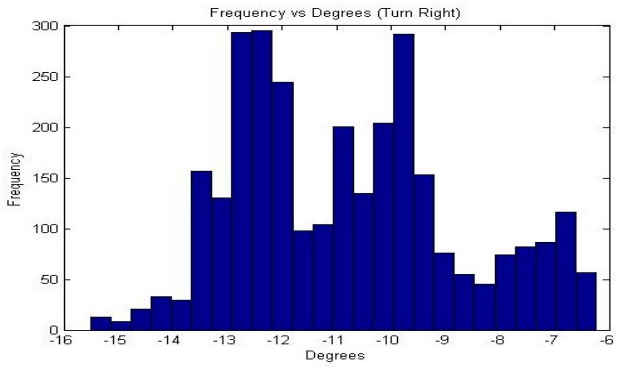


Figure 24: This is the histogram for our hand in the turn right state. It is only supposed to take into account the degrees for roll, therefore the data in this histogram is a graphical interpretation of roll in degrees. As can be seen, the data's mean is around -13 degrees, and diminishes on the tails. Movement-wise, yaw is the most difficult for the pilot and is likely to give us shaking noise. From our data it can be seen that the range of error is bigger than the other frames of different position.

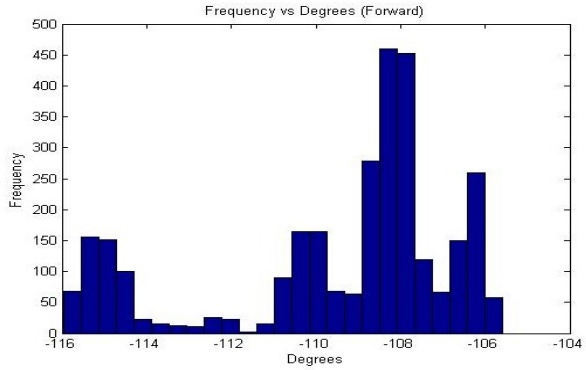


Figure 25: This is the histogram for our hand in the turn right state. It is only supposed to take into account the degrees for roll, therefore the data in this histogram is a graphical interpretation of roll in degrees. As can be seen, the data's mean is around -13 degrees, and diminishes on the tails. Movement-wise, yaw is the most difficult for the pilot and is likely to give us shaking noise. From our data it can be seen that the range of error is bigger than the other frames of different position.

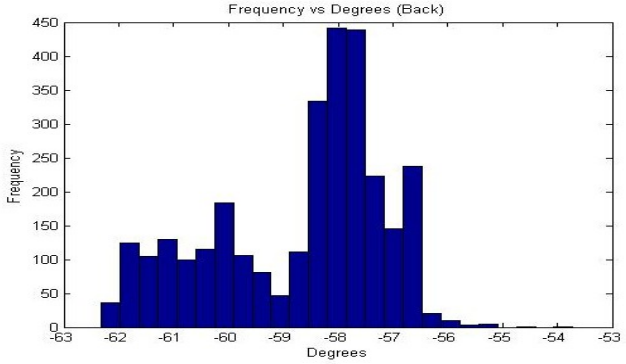


Figure 26: This is the histogram for our hand in the move back state.It is only suppose to take into account the degrees for pitch, therefore the data in this histogram is a graphical interpretation of pitch in degrees. As can be seen, the data's mean is around -58 degrees, and diminishes on the tails. Notice that the data is skewed more towards the more negative degrees, meaning that the hand is shaking towards the move back state more.

rithms are applied internally by the device before getting packaged into frames [11]. So, the data that the script manipulates is pre-processed before being presented to the user. However, this pre-processing does not remove all the noise; it only removes extreme outliers that the device records and does not smooth the signal entirely. Because of this, it is up to the developer to search for and remove unwanted values in the distribution. In addition, histograms like the ones shown in the figures show partially smoothed data by the Leap. Thus, we can manipulate a histogram to determine what kind of gesture is being performed since the pre-processing on the data is not very extensive. The data displayed in the histograms is essentially raw, and it gives us a good depiction of hand at a particular period of time. For example, the gesture in Figure 13 is indicating the Right gesture since a majority of the points are concentrated around the -20 degree point. The feed of data from the Leap Motion gives the developer an advantage because he is allowed to implement whatever filters he likes. The developer can then emphasize whatever data points he wants, be they the central values or the extreme values.

After reading some publications regarding normality testing, we came across four tests that were reputed to be the best. These were the Kolmogorov-Smirnov test, the Lilliefors test, the Anderson-Darling test, and the Shapiro-Wilks test [1]. Although all of these test for normality, each processes the data in a different way.

The Kolmogorov-Smirnov test is a type of empirical distribution test (EDF), and it calculates the largest vertical difference between empirical data and hypothesis data [8]. That means we need a set of data which we know is comparable to our empirical data, but in our case we cannot do that because we don't know the hypothetical data for our hand positions.

The Lilliefors test is similar to the Kolmogorov Smirnov test because it is also a type of EDF. However instead of knowing the hypothetical data, the hypothetical data is calculated from the empirical data. This characteristic makes this test attractive for us because we do not know the hypothetical data for our hand positions [7].

The Anderson-Darling test is also a variation on the Cramer Von Mises (CVM) test, and it gives more weight to the tails of the distribution than does the CVM test. The advantage of this test is that it is supposed to be the most powerful of EDF test [10].

The Shapiro-Wilks test is based on a null hypothesis that the data being tested is no different than data drawn from a normal distribution. An alpha

level, which is a measure of statistical significance, is then set to a value of .05. Once this value is chosen, the test is run and consequently returns a p-value. This p-value is compared to the chosen alpha level to determine whether the null hypothesis should be rejected; if the p-value is greater than the alpha level, then the null hypothesis cannot be rejected. If the p-value is less than the alpha value, then the null hypothesis is rejected [2].

After researching these tests, we decided to use the Shapiro-Wilks test because we found it to be the most powerful. We based this decision on a paper written by Razali & Wah [1] which compares the power of the four formal tests. The power of a formal normal distribution is defined as the probability that the test correctly rejects the null hypothesis when the null hypothesis is false.

Razali and Wah used power comparisons which were obtained through Monte Carlo simulation, the data for which was generated by various sample sizes and alternative normal distributions with varying levels of kurtosis and skewness. Skewness quantifies the symmetric qualities of the distribution, and kurtosis quantifies how sharp the peak of a distribution is.

In our case, the null hypothesis is that the data follows a normal distribution, and the power is the probability that this hypothesis is rejected when our data is not normally distributed.

We show the power comparisons of 2 plots of power for symmetric distribution and 2 plots of power for asymmetric distribution from Razali and Wah's paper. The Beta, Laplace, and Gamma distributions were chosen arbitrarily as examples to illustrate the tests.

Figures 27 and 30 are plots of power for the Symmetric Distributions Beta(2,2) and Laplace(0,1). Figures 29 and 30 are plots of power for the Asymmetric Distributions Gamma(4,5) and Gamma(1,5) distributions.

As is shown by the results of the comparison tests, the Shapiro-Wilks test is the most powerful out of the four tests with the various sample sizes and distributions [1]. For every distribution tested, the Shapiro-Wilks test rose to the maximum power level faster than any of the other tests. Another observation is that the Shapiro-Wilks test was at the maximum power level for several sample sizes smaller than the one being considered. For example, by a sample size of 500, the Shapiro-Wilks test had executed with maximum power for the 300- and 400- sized sets before it for every distribution tested. Other tests were still increasing in power for the 300- and 400- sized sets, and they only reached the maximum power level at a sample size of 500 or greater. From these graphs, we determined that the Shapiro-Wilks test would be the

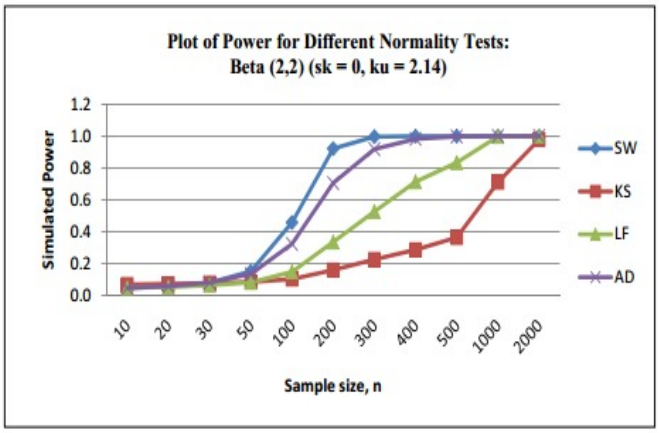


Figure 27: Razali and Wah's power comparison using the Beta(2,2) distribution. "SW" represents the Shapiro-Wilk test, "KS" represents the Kolmogorov-Smirnov test, "LF" represents the Lilliefors test, and "AD" represents the Anderson-Darling test

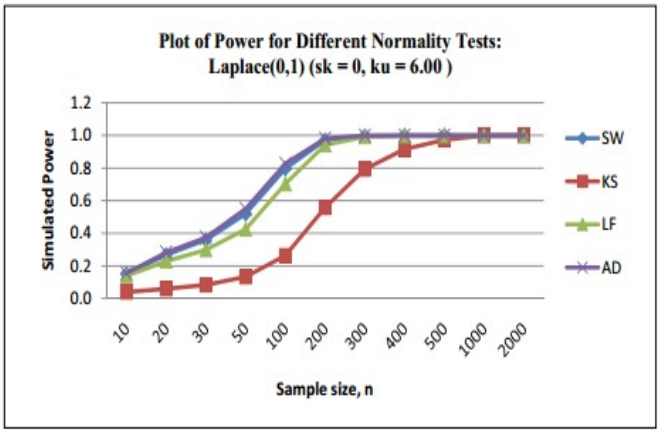


Figure 28: Razali and Wah's power comparison using the Laplace(0,1) distribution. "SW" represents the Shapiro-Wilk test, "KS" represents the Kolmogorov-Smirnov test, "LF" represents the Lilliefors test, and "AD" represents the Anderson-Darling test

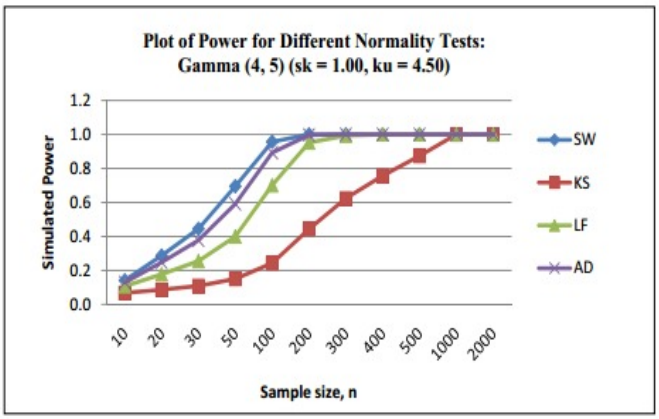


Figure 29: Razali and Wah's power comparison using the Gamma(4,5) distribution. "SW" represents the Shapiro-Wilk test, "KS" represents the Kolmogorov-Smirnov test, "LF" represents the Lilliefors test, and "AD" represents the Anderson-Darling test

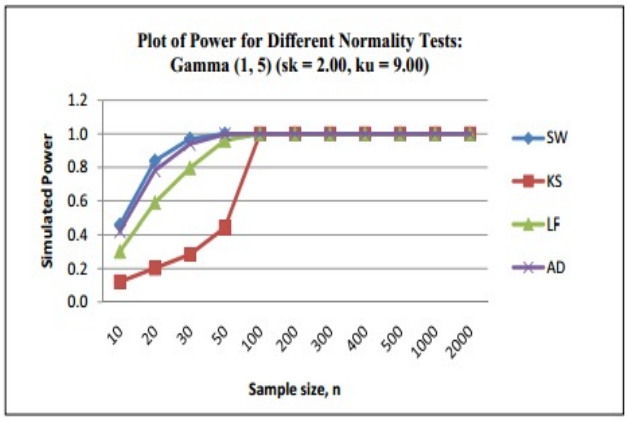


Figure 30: Razali and Wah's power comparison using the Gamma(1,5) distribution. "SW" represents the Shapiro-Wilk test, "KS" represents the Kolmogorov-Smirnov test, "LF" represents the Lilliefors test, and "AD" represents the Anderson-Darling test

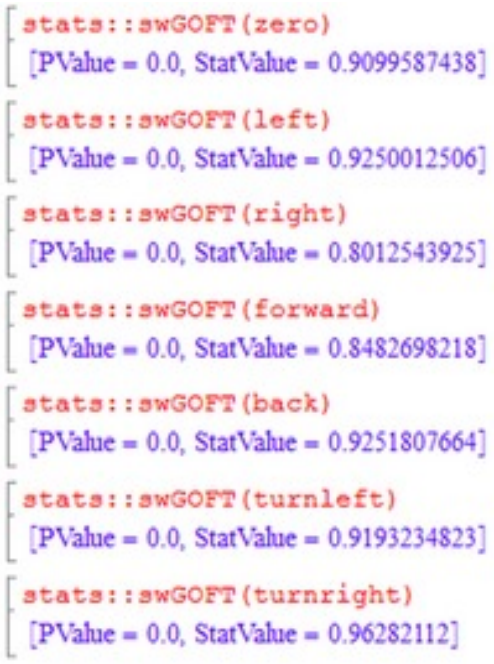


Figure 31: Results from MATLAB's Shapiro-Wilks test

most powerful test to use because it can give us accurate normality readings for small and large sample sizes. Although the Shapiro-Wilks test is accurate for a wide range of sample sizes, we decided to use the test with a sample size of 3000. This decision was based on the rate at which the Leap Motion generates data along with the desire to ensure that the test operated at the maximum power.

Once we determined that the Shapiro-Wilks test was the ideal test of choice for our study, we used it to test our data using MATLAB as seen in figure 31. However, none of the outputs that we received returned a positive result for any of our data sets. Therefore, our hypothesis was unfortunately rejected as the p-value for every sample came out to be 0.0, which is less than 0.05 [2].

We knew that rejection of the null hypothesis was possible, but we did not expect the test to reject the null hypothesis for every data set. This seemed strange to us, so we decided to run a general distribution test in MATLAB to find out what distribution the data corresponded to. The test matched the data sets to the General Extreme Value distribution. This distribution is

characterized by a set of data which focuses on the minimum and maximum values, or the "rarities." It arises as "limiting distributions for maximums or minimums (extreme values) of a sample of independent, identically distributed random variables, as the sample size increases" [12]. Since we did get a response back from this test, we decided to use one more method of confirmation: a quantile-quantile plot. Quantiles are points taken at regular intervals from the cumulative distribution function of a random variable, a function whose value is the probability that a corresponding continuous random variable has a value less than or equal to the argument of the function [4]. Thus, the QQ (Quantile-Quantile) plot compares the quantiles of our data with the quantiles of the normal distribution The comparison tells us that the closer the data points are to the line y = x, the more likely it is that the data set came from a normal distribution.

Figure 32(a) shows the QQ Plot for the Left Position. As is shown by the plot, the center of the data follows the y=x line. However, the data set deviates from the line towards the tails of the plot, which indicates that there was a greater deviation from the normal distribution for those points. We saw similar behavior in the QQ plots of the other positions, but some of those plots visibly deviated more at the tail values than did the plot for the Left position. For example, the Forward position plot exhibited a great deal of variation on both the left and right tails of the distribution, much more so than the Left position plot, and a small of amount of variation around the center of the distribution. The Forward position plot contained the most deviation from the y=x line. From these plots, we confirmed that our distribution is that of the general extreme value distribution, because the tails deviated from the line thus it focuses on the minimum and maximums of our data.

Since the general extreme value distribution emphasizes the minimums and maximums of the data, we focused on including those variations. To take the tails into account, we decided to include data that fell within two standard deviations of the mean. This guaranteed that the behavior at the tail end of the distribution would be recorded and accounted for.

Using the left test, we can see that the data follows normal distribution in the middle, but does not at the minimum and maximum. The QQ Plots for the different positions: Hover, Move Left, Move Right, Turn Left, Turn Right, Move Forward, and Move Back can be seen below.

Armed with the knowledge that our data follows the general extreme value distribution, we determined threshold values with which the pilot's

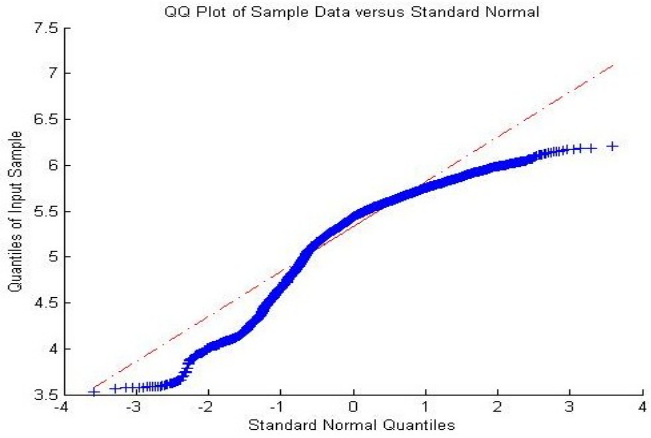
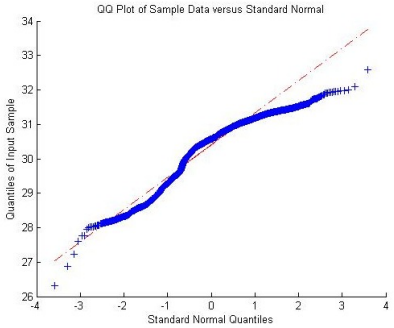
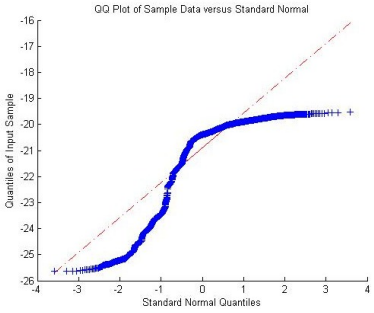


Figure 32: QQ Plot for the Hover gesture.

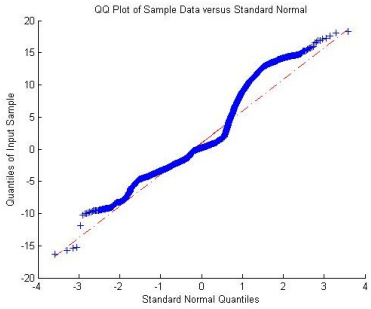


(a) QQ Plot for the Move Left gesture.

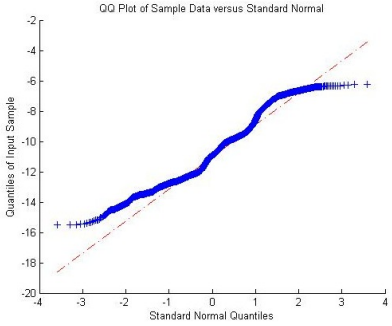


(b) QQ Plot for the Move Right gesture.

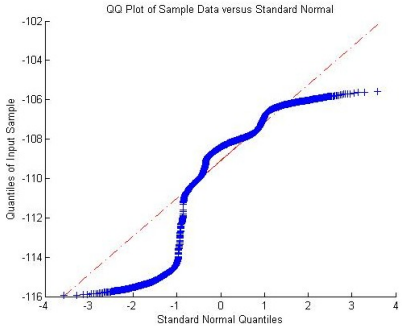
hand can shake and still be considered the correct gesture. These thresholds serve as boundary values for each state; any degree measurements recorded outside of these values will not be recognized as part of the gesture. We decided to use the value of two standard deviations on either side of the mean as our threshold values because these values would include 95% of the data points. By doing this, we make sure that certain degrees of tilt can be tolerated, but not so much that the gesture becomes too extreme for the user. For example, the move left state could be positioned at 45 degrees from the neutral position, and angles in the range of 43 to 47 degrees will be recognized if the standard deviation is 2 degrees. The value of the deviation for each gesture is tabulated in the following table.



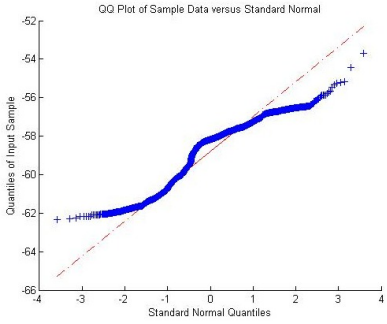
(a) QQ Plot for the Turn Left gesture.



(b) QQ Plot for the Turn Right gesture.



(a) QQ Plot for the Forward gesture.



(b) QQ Plot for the Backward gesture.

Gesture	Value of Two Standard Deviations (degrees)
Hover	1.0853
Move Left	1.7864
Move Right	3.2460
Turn Left	11.5063
Turn Right	4.0744
Move Forward	5.7164
Move Backward	3.1171

We can group these gestures to rotational movements with Hover, Move Left, and Move Right as Roll, Turn Left and Turn Right, as Yaw, and Move Forward and Move Back as Pitch. To find the maximum value for tolerance, we compared the deviation values for each flight dynamic. We took the greater of the two values and rounded it to the next greatest integer. For example, the pair of gestures corresponding to roll are Move Left and Right.

The threshold value for these states are 1.7864 degrees and 3.246 degrees respectively. Since 3.246 is greater than 1.7864, we round 3.246 degrees to 4 degrees and set that as the tolerance for the two gestures corresponding to roll. Similarly, we found tolerances of 6 degrees and 12 degrees for the pitch and yaw gestures respectively. By setting these tolerances, we define concrete states which include even the maximum and minimum values deviating from the desired position. The tolerances and their effects on the theoretical degree value are shown in the following table.

Gesture	Tolerance (degress)	Ideal (degrees)	Positive Threshold (degrees)	Negative Threshold (degrees)
Hover	±2	0	+2	-2
Move Left	±4	+45	+49	+41
Move Right	±4	-45	-41	-49
Turn Left	±12	-45	-33	-57
Turn Right	±12	+45	+57	+33
Move Forward	±6	+45	+51	+39
Move Backward	±6	-45	-39	-51

Hand Drifting

We noticed that Hand Drifting is caused by slight unintentional movements over time when the pilot is focused on the drone. This means that drifting occurs when there is a small and sustained rotational movement of the hand. The difference between Hand Drifting and Hand Shaking is the type of movement involved. In hand shaking, the hand "vibrates" around the position it is expected to be in — the thresholds previously mentioned are used to determine maximum ranges within which the hand can vibrate. With Hand Drifting, the hand may steadily and unintentionally be rotating towards another state, as mentioned previously in the Stabilizing Flight subsection. Therefore, if the rate of change of the hand is constant and non-zero between two frames of data, the drifting of the hand must be taken into account. Since the Leap Motion can extract degree measurements from each frame, the measurement of the rate of change is simply:

$$\text{Rotational Speed of Hand} = \frac{\text{Current Position} - \text{Previous Position}}{\text{Elapsed Time}}$$

The two frames used to calculate the rotational speed are the current frame and the previous frame. The goal of this algorithm is to allow the software to recognize which hand movements are deliberate and which hand movements are non-deliberate, allowing it to safely discard signals from non-deliberate movements. Non-deliberate hand drifting has a rotational speed that is much smaller than the rotational speed of deliberate hand movement. To correct for this we eliminate data that has a rotational speed that is outside of range of rotational speeds for deliberate hand motions. The minimum and maximum rotational speed of deliberate hand motions are calculated below and control signals will only send commands if the rotational speed of the hand is within the range eliminating undesirable signals.

Flight Dynamic	Maximum Rotational Speed (degrees/second)	Minimum Rotational Speed (degrees/second)
Pitch	63.4	22.3
Yaw	19.4	8.3
Roll	139.1	112.4

This allows a tighter threshold range as it does not need to account for non-deliberate hand movements as much as a result the controls become more responsive.

Lastly, we decided to smooth our data and filter any unexpected spikes in our data through a moving average filter. The reason for this is that the speed at which our hand moves is in the timeframe of seconds, and any sudden spikes which may occur in one of the 300 frames per second would indicate unwanted noise. This algorithm collects a set of points, averages them together, and generates a mean value for that set of points. This mean value is considered to be a representative for the ten initial points. The algorithm calculates another representative point based on the next set of ten numbers starting from the second value in the initial set. The number of points we use to calculate the moving average is called the period length. The period length determines the lag behind the degrees we are calculating, which is why it is so important. There is no "right" value for the length of the priod; the selection of a value is essentially a compromise between the accuracy of the data and the speed of the computation. Accuracy, which is how close the result is to the general trend, is traded for speed, which is how fast a set of averaged points can be generated. Using more points

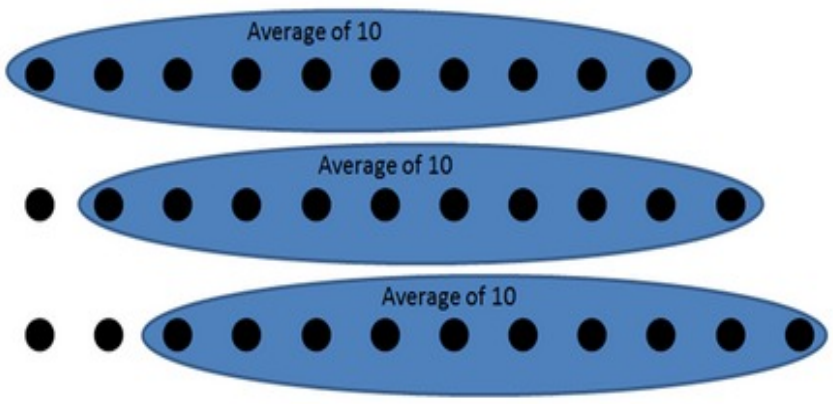


Figure 33: Graphical depiction of the moving average.

will make the algorithm more accurate, however the process will take longer to compute the averaged points [13]. This is because for every point in the data set, the algorithm must average a larger number of points. Conversely, taking fewer points will generate less accurate data points, but the algorithm will compute the averaged points faster since there are less points to average per data point in the set. We opted to take the moving average across 10 points because the time taken to complete the algorithm is 1/30 of a second. We found this to be the best compromise between speed and accuracy. It smooths and filters our data with enough accuracy in a reasonable amount of time. In addition, taking more points would mean that quicker movements would not be interpreted as such. This is because the outliers in the data set would be averaged to a value that would make the change in direction less pronounced. Thus, the algorithm continually smooths the data as it is being generated by the leap motion. By doing this, we eliminate unexpected variations in the data and make the underlying pattern more visible. Figure 33 provides a visual representation of how the moving average works, and figure 34 provides a mathematical expression for the moving average.

Through the use of threshold values, the rotational speed of the hand, and moving average, we manage to achieve much more stable flight. Their combination indivdiually reduces certain types of undesirable signals. The threshold values filter out small quick hand movements where the rotational speed calcuated is not as accurate, and the rotational speed filter elimiates most of the drifting. When the effects are combined, they minimize the effects

$$s_i = (1/n) \sum_{j=i}^{i+n-1} a_j$$

Figure 34: Mathematical expression for the moving average. n represents the period length for the average, and a_j represents the jth data point. s_i represents the resulting averaged value from the algorithm [14].

of two major unintentional signals individually to produce a signal that the user intended to make. The moving average helps reduce variations in the signals that go outside of the accepted paramenters, allowing the drone to update more frequently and be more responsive. Also, the controls have also become much more fine-tuned due to the fact that the signals are now much clearer to the drone after our filtering and smoothing algorithms.

Communication with the AR Drone

The drone receives information through its own built-in Wi-Fi access point, where software may send instructions to. Using Python, we created a network socket in order to communicate with the drone through this Wi-fi access point. The Leap Motion controller is made to capture 300 frames every second, or one frame approximately every 3.33 milliseconds. Under ideal conditions, this would translate into an instruction being sent to the drone every 3.33 milliseconds as well. We choose to send an instruction to the drone every time a frame is processed rather than processing multiple frames and then sending an instruction every 33 or 333 milliseconds in order to maximize the stability of the flight and minimize delays. As will be explained later, by sending instructions as often as possible we implement a safety measure to ensure that a good connection is being maintained between the application and the drone. This allows the pilot to have maximum control over the drone - if instructions were sent every 300 milliseconds rather than every 3 milliseconds, or even 30 milliseconds rather than 3 milliseconds, unnecessary delays would be noticed. Essentially, the response time decreases as we send instructions to the drone more rapidly. We found 3.33 milliseconds to be the fastest interval we can use to capture new data from the drone. We decided to utilize User Datagram Protocol (UDP) rather than Transmission Control Protocol (TCP). The main difference between UDP and TCP is their

speed and reliability- while TCP goes through procedures to ensure that data packets sent are delivered through its hand-shaking process, it sends data at a slower rate than UDP, which does not care as much about reliability as it does speed. For this reason, we decided to utilize UDP. Since we send 300 frames every second and human hands tend to not vary greatly within a second, making sure that every instruction is correctly received is not so important. Rather, it is more important that instructions are sent with as little delay as possible so that the pilot does not recognize a significant or noticeable amount of delay between his instruction and the drone's flight path. As a measure to prevent loss of control over the Drone, we included a safeguard that commands the Drone to automatically land and shut down if no recognizable instructions have been received for 300 milliseconds. We put drone instructions inside of a callback method, which makes the program send instructions every time a frame is captured by the Leap Motion Controller. However, an occasional issue occurs when the Leap Motion controller may drop frames on resource-limited computers, or the application might miss frames if the listener takes too long to process a single frame, as it can not grab the next frame until the current one is finished processing. Unfortunately, frames are being processed and instructions are being sent with such high frequency that there is no definitive way to know when single frames are being dropped. However, this very high frequency is also fortunate because it means that even if single frames are being dropped, the next instruction will be sent to the drone before any significant or noticeable damage is dealt. As long as the latest instruction sent to the drone is being executed, the instructions sent to the drone during a transition to the new state are safely discarded as they are no longer relevant. If the drone fails to receive instructions for 300 milliseconds (approximately 100 frames), it is evident that disconnection is either due to network problems or a major fault in the controller/application.

Results

The overall aim of this project was to create a simple interface for mapping intuitive hand gestures to the flight patterns of the AR Drone. We sought to do this by analyzing the position data that the Leap Motion generated by observing hand movements. In specific, we wanted to monitor the variation in degree from the mean position of the hand, which is when the user has his

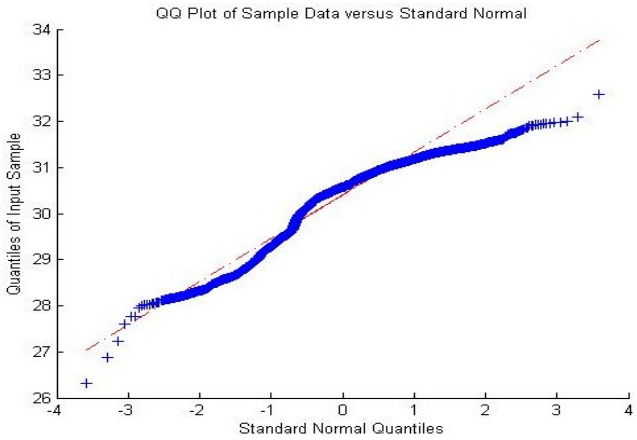


Figure 35: QQ Plot for the Move Left gesture.

palm facing towards the floor and positioned directly over the Leap Motion. Our initial histograms showed us a distribution of data which seemed to match the normal distribution, so we decided to verify that assumption. We ran through the Shapiro-Wilks and found the previously mentioned General Extreme Value distributions were the best match for our data sets.

These tests provided us with a standard deviation for each position, which we then used to define the states for the hand. For example, when we moved into the "turn right" state, we found that the standard deviation from the mean of 45o was 1.8176 degrees. Using our proposed rule of including all data within two standard deviations of the mean as part of the state, we decided that the user's hand could vary by 3.6352 degrees in the counterclockwise or clockwise direction from the mean position. Similarly, we used the standard deviations to define regions within which the system would recognize that the hand was in a particular state.

The combination of the defined states and the flight-stabilizing algorithm resulted in precise control of the AR Drone. We tested each of the six states (both positive and negative orientations for yaw, pitch, and roll), and the Drone responded to each gesture with ease. We also tried slightly more complicated movements, such as navigating a figure-eight. While this proved to be more difficult than movement in general directions, we were still able to complete the flight path with less difficulty than with the default mobile interface. Based on the responsiveness of the Drone and the analysis of our data, we believe that we have achieved our objective of creating an intuitive,

gesture-driven interface for the AR Drone.

Future Work

There are several different improvements that can be made to the project. First, the transitions between the different states of movement can be improved. In our tests, we found that the Drone would not respond instantaneously to a change in the hand movement. This delay could be due to the time that the Leap Motion takes to detect and register hand movements through the frames, or it could be due to a definition of hand states that was not as precise. Also, the inertia of the drone could be a factor of the delay. When the drone moves in a certain direction, it has significant amount of momentum. The motors must fight against this momentum whenever the user wants to change the direction of the drone, which takes a small but significant amount of time. In addition, the time taken by the motors to counteract the momentum can vary depending on how fast the drone is going. As a result, the user may not get an instantenous response from the drone when it tries to change direction. Regardless, the response of the Drone to quick changes in directions is an area for further work.

In addition to response time, the precision of control over the Drone should be worked on. In our figure-eight test, we found that the maneuvers did not provide enough control to navigate the path. Control was especially difficult when traversing the sharp turns. Our current system did not account for extremely precise movements of the Drone, and we feel that this is definitely an area for future work.

Finally, integrating the project with Google Glass and utilizing the Drone's video camera is the most interesting avenue for development. The goal would be to have the user stream video from the Drone's camera into the Google Glass and then manipulate the Drone based on the video feed. The Leap Motion would also be worn by the user so that he is not completely bound to one location by a laptop. This opens up exciting new possibilities for control, and the Drone could be developed to perform certain tasks, such as picking up objects in places that the user cannot reach. Although this is an ambitious plan, there could be immense rewards.

Contribution Breakdown

	Tae-Min Kim	Felix Yeung	Nikhil Shenoy	Shu Xu
Report	25%	25%	25%	25%
Stability Testing		50%	50%	
Stability Implementation	50%			50%
Drone Instructions		50%	50%	
Leap Motion	50%			50%
Leap/Drone Integration	25%	25%	25%	25%

References

[1] Razali, Nornadiah Mohd, and Yap Bee Wah. "Power Comparisons of Shapiro-Wilks, Kolmogorov-Smirnov, Lilliefors, and Anderson-Darling Test." *Journal of Statistical Modeling and Analytics* 21-33 2.1 (2011): 21-33. Print.

[2] Shapiro, S.S.; Wilk, M.B. (1965). "An analysis of variance test for normality (complete samples)". *Biometrika* **52** (3-4): 591-611. doi:10.1093/biomet/52.3-4.591. JSTOR 2333709. MR 205384. p. 593.

[3] Hazewinkel, Michael, ed. (2001), "Normal distribution", *Encyclopedia of Mathematics*, Springer, ISBN 978-1-55608-010-4

[4] Scott, David. "Quantile-Quantile (q-q) Plots." *Online Statistics Education: A Free Resource for Introductory Statistics*. National Science Foundation's Division of Undergraduate Eduation, n.d. Web. 03 May 2014.

[5] David, Lane M. "Histograms." *Histograms*. National Science Foundation's Division of Undergraduate Education, n.d. Web. 01 May 2014.

[6] Arshad M., Rasool, M.T. And Ahmad, M.I. (2003). *Anderson Darling and Modified Anderson Darling Tests for Generalied Pareto Distribution*. Pakistan journal of Applied Sciences 3(2),pp.85-88.

[7] Lilliefors, H.W. (1967). *On the Kolmogorov-Smirnov Test for Normality with Mean and Variance Unknown*. Journal of American Statistical Association, Vol. 62, No. 18, pp. 399-402

[8] Conover, W.J. (1999). *Practical Nonparametric Statistics*. Third Edition, John Wiley and Sons, Inc. New York. pp. 428-443 (6.1)

[9] "Frame." *Frame - Leap Motion Python SDK V1.2 Documentation.* Leap Motion, n.d. Web. 20 May 2014

[10] "Cramer-von Mises criterion." *Wikipedia: the Free Encyclopedia.* N.p., n.d. Web. 20 May 2014. http://en.wikipedia.org/wiki/Cram%C3%A9r%E2%80%93von_Mises

[11] "Getting Frame Data." www.developer.leapmotion.com. Leap Motion, n.d. Web. 20 May 2014

[12] "The Extreme Value Distribution." The University of Alabama in Huntsville, n.d.Web. 19 May 2014

[13] "Moving Average Period." *Macroption RSS*.N.p., n.d. Web. 27 May 2014

[14] Weisstein, Eric W. "Moving Average." From *MathWorld* – A Wolfram Web Resource. https:mathworld.wolfram.com/MovingAverage.html