5-1-2016

Automating 3D Wireless Measurements with Drones

Ethan Yu
Dartmouth College

Automating 3D Wireless Measurements with Drones

Author: Ethan Yu

Advisor: Xia Zhou

Dartmouth Computer Science Technical Report TR2016-796

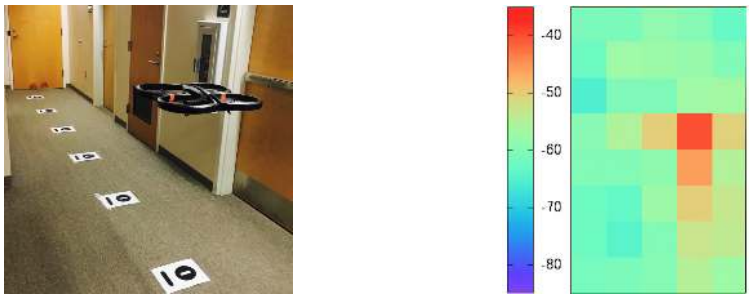


Figure 1: Drone in flight, visualization of wireless signal strength (dBm)

ABSTRACT

Wireless signals and networks are ubiquitous in today's world. Though more reliable than ever, wireless networks still struggle with weak coverage, blind spots, and interference. Having a strong understanding of wireless signal propagation is essential for increasing coverage, optimizing performance, and minimizing interference for wireless networks. Extensive studies have been done on the propagation of wireless signals, and many theoretical models have been made to simulate wireless signal propagation. Unfortunately, models of signal propagation are often not accurate in reality, and real-world signal measurements are required for validation.

Existing methods for collecting wireless measurements involve human researchers walking to each location of interest and manually collecting measurements, which requires large amounts of time and effort, or placing sensors at each location of interest, which is costly. We propose DroneSense: a system for measuring wireless signals using autonomous drones. DroneSense reduces the time and effort required for measurement collection, and is affordable and accessible to all users. This is significant in the field of wireless networking as it provides researchers with an efficient method to quickly analyze wireless coverage and test their wireless propagation models.

1. INTRODUCTION

Wireless signals and networks are ubiquitous today, and accurate control of wireless signal propagation and the resulting wireless network coverage is of great importance in residential and commercial environments. Though more reliable than ever, wireless networks still struggle from problems such as weak signal, blind spots, and interference. Key to solving these problems is the understanding of wireless signal propagation. Studies on wireless signal propagation often rely on theoretical models. Although these models have improved our understanding of wireless signal propagation, they are still limited in accuracy. Because signal propagation models may not necessarily be accurate in reality, we rely on actual signal strength measurements to validate such models.

Existing Methods: One method for collecting highly granular wireless measurements is what we will refer to as *walk sensing*. Researchers must collect measurements at every point of interest by walking to each location. Walk sensing is extremely time consuming, because it requires constant human interaction. This makes it difficult to collect data over a large number of locations (e.g., in a large building), and to collect data multiple times over the same locations (e.g., each time after modifying the signal).

Another method for collecting wireless measurements is to place sensors at every point of interest, and then repeatedly read measurements from those sensors. However, this method is costly because it requires purchasing multiple sensors, which presents a barrier if the number of collection points need to be large.

DroneSense: In this thesis, we propose DroneSense, a system for automatically collecting wireless signal measurements using drones. To meet our objective, we pro-

gram a drone to fly along a preset trajectory and collect measurements along its path. The user inputs a list of coordinates of where to collect measurements, and all drone navigation and signal measurement is handled by the program.

Challenges: The primary technical challenges of a wireless measurement system using drones are accurate navigation and efficient flight. First, for accurate navigation, it is difficult to track the position of the drone because GPS does not operate at this resolution. Without an accurate navigation system, it is impossible to determine the drone's current position and reach the target position for collecting measurements. Second, for efficient flight, because the drone has limited battery capacity, it is imperative that the drone's trajectory to the target is as efficient as possible. If inefficient control commands are sent to the drone, the drone will spend a lot of time adjusting its position before reaching the target location, wasting time and energy.

Methods: DroneSense utilizes a combination of methods to precisely navigate and control the drone. These methods include using an Extended Kalman Filter to estimate the drone's position at all times, using computer vision techniques to detect reference points in the drone's environment to correct the position estimate, and using a Proportional-Integral-Derivative Controller to dynamically control the drone's trajectory to have an efficient flight.

Implementation: DroneSense is accessible to a wide variety of users. We used the Parrot AR Drone 2.0 [18] for our work, as the Parrot AR Drone 2.0 is cheap and can be easily operated. We chose this drone for its low cost, abundance of open-source APIs, and durability while flying indoors. Our code is written in JavaScript, on the Node.js platform, which is compatible with all operating systems. Our software seeks to be as user-friendly as possible, because one of the key advantages we hope to provide with our system is usability. We present the user an easy way to set the drone's trajectory, various options for collecting signal measurements, and a suite of tools for automation and calibration.

Benefits: The primary benefits of this system is that it provides a fully autonomous process for collecting wireless measurements. This reduces the amount of human effort required, and makes the measurement process more efficient. The system is easily scalable to a large number of collection points, and is easily repeated multiple times. Furthermore, it allows for collecting signal strength measurements in 3D space, adding additional data granularity. Our system is also easily accessible to all users because it uses an affordable off-the-shelf drone and a program that is compatible with all platforms.

Key Results: Our system can ensure an accuracy of 20 centimeters from the target location. At this error tolerance, drone flight is efficient, with distance to the target monotonically decreasing with no overshoot or adjustment. Furthermore, measurement at each location for three seconds is sufficient for accurate results.

In the rest of this thesis, we discuss in detail the technical challenges of creating this system, DroneSense's design, the methods we utilize, the evaluation of our results, summarize DroneSense's effectiveness compared to related works, and discuss its impact and how our work can be extended.

2. CHALLENGES

In creating an automated system for drones to collect wireless measurements, we faced challenges in developing precise control over the drone's movement. Two notable challenges are accurate navigation and efficient flight. Accurate navigation means accurately determining the drone's current location relative to the target's location in order to accurately navigate to the target. Efficient flight means reaching the target location quickly without having to make many large adjustments in flight trajectory.

These challenges arise because of the characteristics of the drone control API. When communicating with the drone, control commands are sent to the drone as a pair of direction and magnitude, such as "left 1.0" or "up 0.5". The greater the magnitude, the faster the drone will move in that direction. The magnitude is most analogous to motor speed, but does not map directly any physical unit of movement such as velocity or acceleration. This is similar to pressing the gas pedal on a car - the pedal pressure is proportional to the speed of the car but not directly translatable, and to tell its effect without a speedometer iis by observing how the car moves. Since the command magnitude does not translate directly to speed of the drone, we must control the drone precisely by relying on feedback from the environment. Below we explain the implications of these challenges, and briefly discuss our solutions to them.

2.1 Navigation Accuracy

Navigating the drone accurately to the target is one of the challenges we faced. Since GPS does not work well at the resolution of our environment, we must find an alternative way to track the drone's position.

We need to track the drone's position because it is not possible to move the drone an exact distance in an exact direction, so must navigate to the target by making adjustments to its trajectory when the drone goes off course.

Initially, we took a simple approach approach to position tracking, that we will call the *naive approach*: by calibrating the drone to understand how the com-

mand magnitude translates into speed. First test control commands for each direction with various magnitudes, and measure the average time it took the drone to travel 1 meter with those commands. Then, when we want the drone to travel a certain distance, we use the values we calibrated as the magnitude command, and hold that command for an amount of time $t_{distance} = distance * t_{1meter}$ before sending the stop command. This way, we can reach any position by determining the distance the drone needs to travel in each direction, and then applying that command for the calculated amount of time.

This method had several shortcomings. First, calibration for one meter does not translate well to other distances. The drone needs to accelerate from stationary and decelerate when we send the stop command, so there is a fixed acceleration/deceleration period during the drone's flight that does not get multiplied for different distances. Second, it is impossible to get precise calibration, so errors will accumulate over time. Third, the drone does not move perfectly in the direction we specify. For example, when we tell the drone to move to the right, it may actually move to the right and also a little forward. We cannot prevent these errors with this naive method. When we used this method, we were unable to accurately navigate to even the first target location.

Next, we tried using the velocity and rotation sensor data provided by the drone. Over each short period of time, we compute the change in position based on the average velocity and rotation during that period of time. This method worked better than the naive method, allowing us to reach a few targets. However, because the sensor data is noisy and inaccurate, errors still accumulated over time.

This motivates us to find a better solution. In order to successfully navigate the drone, we cannot simply rely on the commands we send it and assume it was followed perfectly. We also cannot rely completely on sensor data, as it may be noisy and inaccurate. Instead, we need to receive feedback from the drone to learn about its position at all times, and account for noise from our sensor readings. Our solution to this challenge is using Reference Point Detection and an Extended Kalman Filter, discussed in sections 4 and 5.

2.2 Flight Efficiency

The other main challenge was ensuring the drone flew in an efficient path. The impact of this problem is not just that the drone will take an indirect path to the target, but more importantly when the drone is near its target, its position will fluctuate wildly as it tries to adjust itself into the exact target position. Since the drone has a limited battery life, it is important to make flights as efficient as possible so that more flight time can be achieved. The efficiency of the flight depends on the magnitude of the control commands we send to the drone, which will determine how quickly the drone reaches its target.

Initially, we used the previously described naive approach to set control command magnitudes. We used a single small magnitude, such as 0.2, and varied the time for which we applied that command. So when the drone is far from the target, we apply the command for a long time, and when it is close to the target, we apply it for a short time, calculated with the equation in the last section.

However, this method had several shortcomings. First, it takes a long time to reach far distances because the magnitude is so low. Because we only use one value for the magnitude, there are instances where a higher magnitude would allow us to reach the target much quicker. Second, our timing is not accurate, which forces us to make many course adjustments. So the drone will constantly undershoot or overshoot the target, and require another adjustment.

Next, we tried keeping the time interval constant but varying the magnitude of the control command. We chose magnitudes proportional to the distance to the target. So when the drone is far from the target, we apply a high magnitude, and when it is close to the target, we apply a low magnitude $m = \alpha * distance$. This solved the first problem of taking a long time to reach the target. However, this resulted in more frequent overshoot, as the drone was unable to stop right over the target because the proportional commands in the latter part of the flight were too high. If we lower the coefficient α, then the drone flies to slowly during the earlier part of the flight. So clearly, a non-linear equation for determining the magnitude is optimal.

The problem of efficient flight arises because we do not have an effective method to determine the magnitude of the command to send to the drone. Using the naive method from above, we typically stick to a small finite set of magnitudes such as 1.0, 0.5, 0.3, 0.1, 0.05, etc. that we arbitrarily chose. And if we use magnitudes proportional to the error, the drone does not decelerate fast enough when it is close to the target, unless the values are very small and the drone moves very slowly. To make the drone's flight more efficient, we must receive feedback about what is the effect of the magnitude we used, and adjust the magnitude on a continuous and non-linear scale in order to find the optimal magnitude. Our solution to this challenge is the Proportional-Integral-Derivative Controller, discussed in section 6.

3. SYSTEM OVERVIEW

DroneSense takes as input a flight path, which is a list of locations to collect measurements. The user also specifies options for collecting measurements, such as

the number of measurements to make at each point and how long to wait between measurements.

DroneSense is divided into two modules: Navigation and Measurement. The Navigation Module is responsible for moving the drone to the desired locations. The Measurement module is responsible for collecting measurements at each desired location.

The Navigation Module is the core of the system, and contains our approach to address each of the challenges listed above. Specifically, to address the problem of navigation accuracy, we use Reference Point Detection and an Extended Kalman Filter; to address the problem of flight efficiency, we use a Proportional-Integral-Derivative Controller. These components of the Navigation Module will be discussed in detail in the following sections.

4. REFERENCE POINT DETECTION

We use reference point detection to help address the challenge of navigation accuracy, by providing position calibrations for the drone. When the drone detects a reference with its camera, we can calculate the reference point's relative position and orientation to the drone in 3D space. We can also predefine each reference point's true position in 3D space so that the program knows the the true position of the detected reference point. Thus, when a marker is detected, we can use the its true position and its relative position and orientation to the drone to calculate the drone's own position and orientation. We place multiple reference points into the environment to so the drone can constantly have a source of position calibration.

The drone has the capability to detect certain types of markers, including the Oriented Roundel marker that we use. Figure 2(a) shows the Oriented Roundel marker, and we see that it is not only very easy to recognize, but also we can easily determine its orientation. Once a marker is detected, we perform calculations to transform its position in the camera image to its position in the drone's coordinate system, and also find its real position. Although the drone has built-in capability to detect these markers, we found that ability lacking, and created our own computer vision marker detection system.

Computer Vision: In order to detect a marker, we use computer vision algorithms [17] to analyze the image detected by the drone's camera. First, we convert the image to grayscale. Next, we run the OpenCV Canny Edge Detection algorithm to produce an image where only the edges pixels are turned on. Next, we dilate the image to increase the size of the edges. Next, we use the OpenCV Contour algorithm and find circles and rounded rectangles in the image based on number of sides for each detected contour. Finally, we process detected shapes to determine if a marker is present in the image. If there exists simultaneously a rounded rectangle whose long side length is equal to the side length of a large circle's bounding rectangle, there is a marker in the image. After a marker is detected, we set the center of the circle as the location of the marker. We draw a line between the center of the circle and the center of the rounded rectangle, and the angle between that line and the X axis is the angle of orientation. We then send the coordinates and the orientation to be converted to 3D coordinates. Figure 2 shows what the computer vision system sees when it is detecting reference points.

Back Projection: The camera has a frame of 640x360 pixels. We need to translate the location of the detected marker from these 640x360 coordinates to 3D coordinates. In other words, we need to translate the marker's 2D position relative to the center of the drone's camera (in pixel units) to the marker's 3D position relative to the center of the drone itself (in meter units). We do so by using the back projection matrix $\mathbf{P}$ that has been calibrated for the drone's bottom camera [4], shown in Equations 1 and 2.

$$\mathbf{P} = \begin{bmatrix} 686.99 & 0 & 329.32 \\ 0 & 688.19 & 159.32 \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

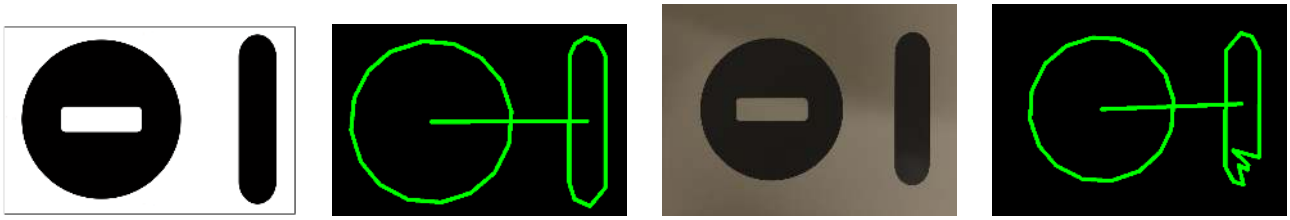
$$\hat{\mathbf{x}} = \mathbf{P}^{-1} * [x * 640/1000, y * 360/1000, 1]^T * h \quad (2)$$

Finally, simple trigonometry allows us to determine the orientation of the marker, shown in Equation 3.

$$\theta = tan^{-1}(\frac{y_{circle} - y_{rectangle}}{x_{circle} - x_{rectangle}}) \quad (3)$$

Thus, we are able to determine the marker's 3D coordinates relative to the drone, as well as the orientation of the drone.

Reference Point Placement: We place markers in the environment 1 meter apart from each other. Fewer markers are needed for the drone to sufficiently navigate, but for more accuracy we try to use as many markers as possible. Because we know that markers are placed at every meter, when a marker is detected, we can assume that marker's true position is the nearest whole-meter coordinate from the drone's current position. If multiple markers are detected, their true positions are the nearest whole-meter coordinates sorted by distance from the drone's current position. For instance, if the drone's current position estimate is (1.7, 2.2), the nearest marker from the drone detected by the camera should have true position (2, 2). We find this assumption works well because the drone's position estimate is easily accurately enough on the 0.5 meter scale. Knowing the marker's true position and its relative position and orientation to the drone allows us to calculate the position and orientation of the drone.



(a) Theoretical Marker (b) Theoretical Marker Vision (c) Real Marker (d) Real Marker Vision

Figure 2: Reference Point Detection

By placing markers at every meter, we make finding the true position of the marker very easy. It is also easy for us to visually inspect the accuracy of the drone, since most collection locations will be a whole-meter intervals so the drone should hover directly over a marker. If we would like to use fewer markers, we can space them out evenly such as one marker every 2 meters, and then tell the program to round its current coordinate to the nearest 2 meter when a marker is detected. If we would like to use irregularly spaced markers, we can indicate the position of the marker on the marker itself with some predefined special symbol, and have a lookup table in the program for the true location of these markers.

5. EXTENDED KALMAN FILTER

We use the Extended Kalman Filter (EKF) [11] to address the challenge of navigation accuracy, by using it to estimate the drone's position at all times. An Extended Kalman Filter is an algorithm that uses a series of measurements over time, containing noise and inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone, by using Bayesian inference and estimating a joint probability distribution over the variables. Estimations are made using a weighted average, with more weight given to measurements with greater certainty. This algorithm has many applications, and is frequently used for guidance and navigation of vehicles.

The Extended Kalman Filter is extremely suitable for our problem because we need to estimate the position of the drone at all times, have many sources of noisy sensor data, and can sometimes make observations of the drone's true position using reference point detection (see section 4). The EKF uses a two step recursive process: predict and update. The predict step uses an position estimate from a previous time as well as sensor data to estimate the current position. In our case, we use a physical model with the last estimate position and the readings from velocity and yaw sensors to estimate the current position. The update step uses the position estimate from the predict step as well as an observation of the position to correct the current position estimate. In our case, we use an observation of the drone's true position when it detects a reference point in the environment to make a more accurate position estimate. The EKF's calculations are based on a weighted average calculated from a computed covariance matrix, or uncertainty of the prediction. A basic linear Kalman Filter uses a linear function for the predict step, while an Extended Kalman filter can use any differentiable function. We require the Extended Kalman filter because our position estimate requires the sine and cosine functions to account for rotation.

Below, we give a brief description of how the Extended Kalman Filter works. We explain the equations and definitions that we used, but do not provide the full derivation. The equations are separated into two sections, for the predict step and update step respectively. Table 1 lists all the variables used.

Variable	Meaning
$\hat{\mathbf{x}}$	Position Estimate Vector
$\mathbf{u}$	Sensor Vector
$\tilde{\mathbf{y}}$	Measurement Residual
$\mathbf{t}$	True Observation Vector
$\mathbf{z}$	Relative Observation Vector
f	Position Transition Function
$\mathbf{f}$	Position Transition Vector
h	Observation Transition Function
$\mathbf{h}$	Observation Transition Vector
$\mathbf{F}$	Position Transition Function Jacobian
$\mathbf{H}$	Observation Transition Function Jacobian
$\mathbf{P}$	Error Covariance Matrix
$\mathbf{Q}$	Process Noise Covariance Matrix
$\mathbf{R}$	Observation Noise Covariance Matrix
$\mathbf{S}$	Residual Covariance Matrix
$\mathbf{K}$	Optimal Kalman Gain

Table 1: Table of Extended Kalman Filter variables

The following equations show how we predict and update the position estimate. We denote the sensor readings as *sensors*, and the camera reference point observations as the *observations*. We use the sensors and

past position estimates to estimate the current position, in other words perform the predict step. We use the observations and past position estimates to correct the model and improve the position estimation, in other words perform the update step.

Predict Step: The predict step utilizes the last position estimate, the velocity readings derived from the accelerometer, the yaw reading derived from the gyroscope, and time interval since last estimation to estimate the current position and orientation. Thus, our position estimate vector $\hat{\mathbf{x}}$ contains the values of x, y, and yaw, and our sensor vector $\mathbf{u}$ contains the values of x velocity and y velocity, yaw, and time interval, shown in Equations 4 and 5.

$$\hat{\mathbf{x}} = [\hat{x}, \hat{y}, \hat{\theta}] \tag{4}$$

$$\mathbf{u} = [v_x, v_y, \theta_u, dt] \tag{5}$$

Using our model based on the laws of physics, we know that the current position should be the past position plus the change in position during the last time interval due to velocity, with a modification for rotation. We call the equations that model this behavior our *transition functions* Our position transition functions f are shown in Equations 6 through 16.

$$\mathbf{f} = [f_x, f_y, f_\theta] \tag{6}$$

$$f_x(\hat{\mathbf{x}}_k, \mathbf{u_k}) = \hat{x}_k + v_x * dt * cos(\hat{\theta}) - v_y * dt * sin(\hat{\theta}) \tag{7}$$

$$f_y(\hat{\mathbf{x}}_k, \mathbf{u_k}) = \hat{y}_k + v_x * dt * sin(\hat{\theta}) + v_y * dt * cos(\hat{\theta}) \tag{8}$$

$$f_\theta(\hat{\mathbf{x}}_k, \mathbf{u_k}) = \theta_u \tag{9}$$

The predict step require a calculation of the Jacobian of our position transition functions $\mathbf{F}$, and also an assumption about the covariance of the predict step process noise $\mathbf{Q}$, which are shown in Equations 10 and 11.

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & -v_x * dt * sin(\hat{\theta}) - v_y * dt * cos(\hat{\theta}) \\ 0 & 1 & v_x * dt * cos(\hat{\theta}) - v_y * dt * sin(\hat{\theta}) \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

$$\mathbf{Q} = \begin{bmatrix} 0.0003 & 0 & 0 \\ 0 & 0.0003 & 0 \\ 0 & 0 & 0.0001 \end{bmatrix} \tag{11}$$

Finally, having all of these variables and equations defined, the predicted position $\hat{\mathbf{x}}$ and prediction covariance $\mathbf{P}$ can be calculated with Equations 12 and 13. The first $\mathbf{P}$ is simply the identity matrix.

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{f}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u_k}) \tag{12}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q} \tag{13}$$

We have now finished the predict step, and successfully calculated the a position estimate.

Update Step: The update step utilizes an observation of the actual position to update, or correct, the model parameters. In our case, when the drone camera detects a reference point, it observes the marker's relative position and rotation, and it knows the marker's true position and rotation. Thus, we can infer the drone's true position based on the discrepancy between the marker's true and relative position. However, although the position estimate derived directly from the reference point detection is likely to be quite accurate, it still may not be perfectly accurate. The update step combines the position estimated in the predict step and the position calculated from the observation, balanced using a weighted average based on uncertainty, to produce a position estimate likely more accurate than both individual calculations.

The observed values will be contained by the true and relative observation vectors $\mathbf{t}$ and $\mathbf{z}$ shown in Equations 14 and 15.

$$\mathbf{t} = [x_t, y_t, \theta_t] \tag{14}$$

$$\mathbf{z} = [x_z, y_z, \theta_z] \tag{15}$$

We use the true marker position and current drone position to calculate what the marker's true position should be relative to the drone, or the marker's theoretical relative position. We call the equations for this calculation our observation transition functions. Our observation transition functions h are shown in Equations 16 through 19.

$$\mathbf{h} = [h_x, h_y, h_\theta] \tag{16}$$

$$h_x(\hat{\mathbf{x}}_k, \mathbf{t}_k) = (x_t - \hat{x}) * cos(\hat{\theta}) + (y_t - \hat{y}) * sin(\hat{\theta}) \tag{17}$$

$$h_y(\hat{\mathbf{x}}_k, \mathbf{t}_k) = -(x_t - \hat{x}) * sin(\hat{\theta}) + (y_t - \hat{y}) * cos(\hat{\theta}) \tag{18}$$

$$h_\theta(\hat{\mathbf{x}}_k, \mathbf{t}_k) = \theta_t - \hat{\theta} \tag{19}$$

Next, we calculate the residual which is the difference between the two relative positions: the relative position of the marker calculated from the camera backprojection, and the theoretical relative position of the marker's true position.

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{h}(\hat{\mathbf{x}}_{k|k-1}, \mathbf{t}_{k|k-1}) \tag{20}$$

The update equations require a calculation of the Jacobian of our observation transition functions, and also an assumption about the covariance of the observation noise, which we calculate and define in Equations 21 and 22.

$$\mathbf{H} = \begin{bmatrix} -cos(\hat{\theta}) & -sin(\hat{\theta}) & (\hat{x} - x_t) * sin(\hat{\theta}) - (\hat{y} - y_t) * cos(\hat{\theta}) \\ sin(\hat{\theta}) & -cos(\hat{\theta}) & (\hat{x} - x_t) * cos(\hat{\theta}) + (\hat{y} - y_t) * sin(\hat{\theta}) \\ 0 & 0 & -1 \end{bmatrix} \tag{21}$$

$$\mathbf{R} = \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0 & 0 & 0.3 \end{bmatrix} \tag{22}$$

Finally, having all of these variables and equations defined, we can calculate the measurement residual covariance and optimal Kalman Gain, which in turn allows us to calculate the updated position and the updated covariance shown in Equations 23 through 26.

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R} \tag{23}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1} \tag{24}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \tag{25}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \tag{26}$$

We have now finished the update step, and successfully updated the predicted position.

6. PROPORTIONAL-INTEGRAL-DERIVATIVE CONTROLLER

We use a Proportional-Integral-Derivative (PID) Controller to address the problem of flight efficiency, by calculating appropriate magnitudes for the control commands for the drone. We want to send the drone control command magnitudes that efficiently move the drone to the target quickly, with few large adjustments, and minimizing overshoot. A PID Controller is suitable for our problem because it allows us to dynamically adjust the command magnitude in a non-linear fashion. A PID controller is a feedback mechanism that continuously adjusts the value of a control variable, in our case the control command magnitude, trying to minimize the error value, in our case the distance between the target position and the current position. The controller gets its name because the value of the control variable is a combination of the present error (proportional), total past error (integral), and possible future (derivative) values of the error.

We use a PID Controller for each of the four directions of motion: front or back, left or right, up or down, clockwise or counterclockwise. Each PID determines the magnitude of the control command sent in that direction.

Our PID controller seeks to minimize distance in a certain direction by setting the control command magnitude u, and uses the feedback to continuously adjust the control variable until a steady state is reached. The PID equation is shown in Equation 27.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \tag{27}$$

where

u = control variable

e = error from feedback

K_p = proportional coefficient

K_i = integral coefficient

K_d = derivative coefficient

In this model, the proportional term accounts for the present error. When the current error is larger, the proportional term will take on a larger value, which increases magnitude of the control command, and the drone will move faster. In the short-run, the proportional term suffers from overshoot. This is because when the error is very large and the magnitude is set at very high, because we are working with discrete time, if the controller is not updated quickly enough, it will update the error too late and not set it to a lower value in time. In the long-term, the proportional term suffers from steady-state error, which means the equilibrium position is different from the desired position. This is because as the error gets very small, the proportional term will be insufficient to move the error to zero.

The integral term accounts for the total accumulated values of the error. The integral term accelerates the process to the desired level, and eliminates steady state error. Once the error is low, the proportional term will stop being effective at reducing the error. However, the integral term will become larger on each time step because there is still error remaining, and then the integral term will increase the magnitude of the control command and move the drone to the correct location. The integral term suffers from overshoot, because it will move the drone in the direction opposite of all its past errors, even when the current error is zero.

The derivative term accounts for the future values of the error, based on the rate of change of the error. The derivative term helps prevent overshoot by modulating the magnitude. When the drone is approaching the target faster, the derivative of the error will be a larger negative value, so the derivative term will become larger negative value, and the magnitude will become smaller. By modulating the derivative of the error rather than the proportional error, the drone can move quickly in the beginning and decelerate quickly as it approaches its target.

Table 2 show the effects of the PID coefficients when they are increased [19]. Speed is how quickly the system reaches 90% of the desired level. Overshoot is how much higher the peak level is compared to the desired level. Settle is how quickly the system reaches its steady state

K	Speed	Overshoot	Settle	Error
K_p	↑	↑	–	↓
K_i	↑	↑	↑	↓
K_d	–	↓	↓	–

Table 2: PID Coefficient Effects

PID	K_p	K_i	K_d
X	0.5	0.1	0.35
Y	0.5	0.1	0.35
Z	0.8	0.2	0.25
Yaw	0.8	0.2	0.25

Table 3: PID Controller Coefficients

level. Error is the difference between the desired level and the steady state level.

Table 3 lists the coefficients we use for each PID. For our system, it is imperative that we avoid overshooting the target, because if the drone somehow moves on top of the wrong reference point, all future flight will be wrong. Therefore, we have a slight preference for slower flight than overshoot. We use medium or large values for the proportional coefficient, small values for the integral coefficient, and medium levels for the derivative coefficient. We need a small integral coefficient to eliminate steady state error, and we need a medium derivative term to reduce overshoot. We use larger values for the proportional coefficient of Z and yaw because movement in those directions tend are slower (due to their motor power), so we want to make those adjustments just as fast as X and Y. We also use smaller derivative terms for Z and yaw because their movement is slower and their sensors are more accurate and thus less prone to overshoot.

7. IMPLEMENTATION

DroneSense is designed to provide a easy-to-use, fast, and flexible way of automatically measuring 3D wireless signals. It is written in JavaScript on the Node.js platform. We chose this platform for its quick development process as well as its numerous existing libraries for working with the Parrot AR Drone 2.0.

DroneSense is divided into two modules: Navigation and Measurement. The Navigation module is responsible for moving the drone to the desired locations, estimating its position at all times, and correcting errors in its trajectory. It utilizes an Extended Kalman Filter and reference point detection to navigate the drone, and a Proportional-Integral-Derivative Controller to stabilize the drone's flight.

The Measurement module is responsible for collecting measurements at each desired location, communicating with the drone to tell it when and how to measure signal. It communicates to the drone's operating system via telnet. It waits for the Navigation Module to broadcast that the drone has reached a target location, collects the measurement, and informs the Navigation Module to start moving the drone again.

7.1 Navigation Module

The Navigation module is responsible for guiding the drone to the desired locations. It parses the provided flight path for coordinates to visit, and creates a queue of destinations to visit. Once the system is ready, it sends the takeoff signal to the drone. It processes information from the drone such as readings for velocity, height, and marker detection, and sends control commands to the drone. When it determines the drone has reached the target, it tells the Measurement module to begin measuring. We use a JavaScript implementation of the API for sending commands to and receiving information from the drone provided by the *node-ar-drone* library [15]. The Navigation Module uses an Reference Point Detection and Extended Kalman Filter to estimate the coordinates of the drone, and uses a Proportional-Integral-Derivative Controller to stabilize the flight.

Flight Plan: The Navigation module takes as an input the user's desired flight plan. The flight plan can be a text file where each line is a 3-dimensional coordinate in meters, and the yaw of the drone is assumed to be 0°. We use a coordinate system with the drone's takeoff position as the origin, initially with yaw of 0°. For Z coordinate simplicity, the user specifies a base level height and between-layer height in meters, so a Z coordinate of 0 will correspond to the base level height and a Z coordinate of 1 will correspond to base level height + between-layer height. The program can also handle traversing a 3D grid flight plan without using an explicit text file, by simply having the user specify the dimensions of the grid. The program calculates each step of the flight, and creates a queue of commands for them while adding hover commands between each step (where the Measure Module will collect measurement), takeoff, and land.

Extended Kalman Filter: The drone API continuously streams navigation data including velocity, yaw, and altitude (about every 50 milliseconds). We subscribe to this stream, and every time new data is provided, the predict step of the Extended Kalman Filter is computed. We use a Javascript Extended Kalman Filter implementation from the *ardrone-autonomy* library [2] with modifications for altitude and yaw calculation and support for multiple reference points.

Reference Point Detection: The drone API continuously streams images from the bottom camera, which the Navigation Module subscribes to (about every 100 milliseconds). We use a JavaScript wrapper of OpenCV [5] [16] to process images frame-by-frame as they are

provided by the stream. When a marker is detected, we take the drone's current position estimate, round that to the nearest whole coordinates, and take that to be the true position of the reference point. Then, we run the update step of the Extended Kalman Filter.

PID Controller: Each time after the Navigation Program estimates its position, it then needs to determine the appropriate control command to send the drone. We use a JavaScript implementation of a PID Controller provided by the *ardrone-autonomy* library [2], but use our own values for coefficients. Each PID for each direction is called with the amount of distance error in that direction, and the PID returns the control command magnitude in that direction. This control command is applied until the next navigation data is provided.

7.2 Measurement Module

The Measurement module is responsible for communicating with the drone and collecting wireless signal measurements. When it gets the signal that the drone has reached a target location to collect measurement, it sends a message to the drone's operating system to begin measuring. It receives the response, parses it, and saves it to a text file.

We first considered collecting measurements using a separate device attached to the drone. However chose not to pursue this method because of the additional complexity of communicating with two devices and also the potential effects an attached device would have on the drone's flight. Therefore, we use the drone's onboard wireless radio to collect measurements and work within the constraints of that system.

AR Drone 2.0 Wireless System: The Parrot AR Drone 2.0 uses the Atheros AR6000 mobile 802.11bg chipset for wireless networking. Its computer runs the BusyBox operating system [6], a lightweight Unix environment, and has about 10 megabytes of free space. Because of the system constraints, we could not host the control software directly on the drone, but instead must rely on communicating with the drone from the computer program. Directly communicating with the drone requires connecting to the same network as the drone and then using telnet. The drone's wireless driver lacks some important functionality such as monitor mode and accurate noise readings. By default, the drone creates its own access point that computer programs can connect to. However, the drone cannot scan other access points when it is using its own access point. Thus, both the drone and computer must connect to the network we would like to measure, communicate over that network, and collect signals only from its current network. For the drone to connect to secure networks, we install WPA2 drivers on the drone via *curl*. The WPA2 drivers are from the *ardrone-wpa2* library [3].

Measurement Script: We leverage the drone's onboard wireless radio to collect wireless signal measurements. We use telnet to access the drone's command line interface, and place shell scripts for measuring signal on the drone's operating system. The scripts takes as parameters the number of times to collect, and the amount of time to wait between collecting. These scripts use *iwconfig* to get the current network's signal strength. *iwconfig* uses driver meta information to interpret the raw value given by */proc/net/wireless*, displaying the result in units of dBm [10]. We use *iwconfig* because it can be executed quickly in rapid succession and its output is easy to parse.

Measurement Program: The Measurement Program running on the computer communicates with the drone and the Navigation module to collect measurements. It allows the user to specify options such as number of collections to make at each point and amount of time to wait between collecting, as well as where to save the log file. It listens to messages from the Navigation module to see when the drone is in the right position. It creates a telnet connection with the drone via the *node-telnet-client* library [20], and then runs one of the measurement scripts we previously placed on the drone. We chose to use telnet because it is a simple way of executing the measurement scripts we have installed on the drone, and the normal drone control API does not provide a way to run programs on the drone's operating system. We use a single continuous telnet connection with the drone and error catching mechanisms to restart the connection if it is broken. The program listens to the Navigation module to find out when to measure, gets the current coordinates, runs the appropriate measurement script, and writes the result to file. It parses the drone's response, logs it in conjunction with the current position, and tells the Navigation module the drone is ready to move to the next location.

8. EVALUATION

We evaluate the accuracy and efficiency of the drone's navigation, as well as, the accuracy and speed of the drone's measurements. These evaluations can establish the effectiveness of our solution, and to identify areas of improvement for future work.

8.1 Navigation Accuracy

Experimental Setup: Our test setup involves a 3x2 grid of reference points one meter apart from each other. The drone takes off from a corner node and travels to the other five nodes. A string and weight is attached to the bottom of the drone, and while the drone is hovering over a collection location, we mark the projection of the location of the drone on the ground indicated

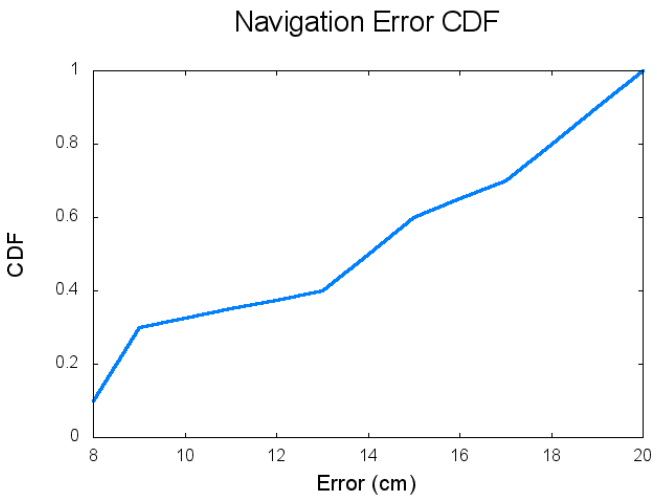


Figure 3: Navigation Error CDF

by the string. Afterwards, we measure the distance between the marked locations and the center of the reference point. The distance between the two points is the navigation error. In the software, we set the accuracy tolerance of the drone to 20 centimeters, which is an acceptable error tolerance and does not cause the drone to spend too long to adjust to reach the target position. We run this experiment 10 times, and find the mean and standard deviation of the error over these 50 data points, and plot the cumulative density function of the results.

Results: The mean error we found was 13 centimeters, and the standard deviation was 4.3 centimeters. Additionally, the error was never more than 20 centimeters, which is important because that is the error tolerance we set in the software. This shows that the drone's navigation is accurate to its specifications. It also means that stricter navigation accuracy standards can be applied, at the expense of navigation time, if greater precision is needed. Figure 3 shows the CDF of the error values.

8.2 Flight Efficiency

Experimental Setup: To evaluate flight efficiency, we have the drone takeoff from one reference point and travel to another reference point 1 meter away, and hover there before landing. We will now work with the drone's estimated coordinates, because it is possible to get multiple position estimates every second. We plot the distance between the drone and the target as estimated by the drone during this flight. We compare the plot of the error when the system uses the current PID Controller implementation, with the plot of the error when the system uses a naive controller. For the naive controller, we a hybrid of the methods we described in the flight efficiency challenge in section 2. This controller sets the magnitude proportional to the error and then rounds the value to the nearest 0.05. This is a reasonable naive implementation, because the magnitude is proportional to the current error and also drawn from a finite set of commands.

Results: Figure 4 shows the error plots for the naive controller and PID Controller. The point where the drone determines it has reached its target is marked. We see that the PID controller not only helped the drone reach the target faster, but stopped the drone successfully once the target was reached. For the inefficient controller, overshoot is a problem because after the target is reached, the drone does not quickly stop itself, overshoots, and has to adjust back to the target. If all controls commands are decreased in magnitude, then the overshoot problem is reduced, but then it will take much longer to reach the target. Therefore, the PID Controller is a superior controller, and significantly increases the efficiency of the flight.

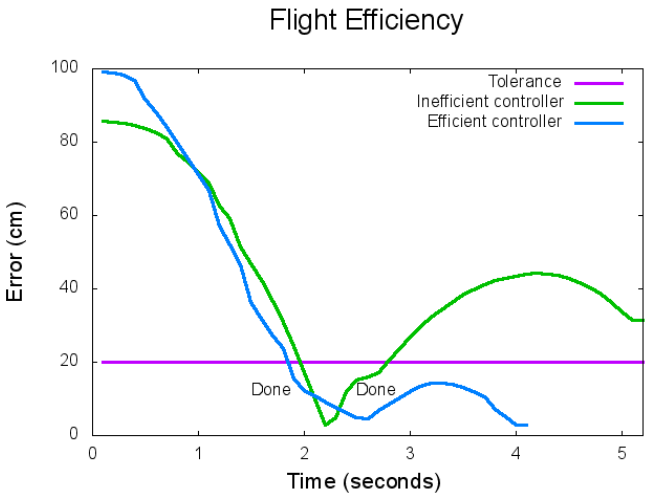


Figure 4: Efficient and inefficient controller comparison

8.3 Measurement Accuracy

Experimental Setup: Although the drone attempts to hover in place when it is collecting measurements, it nevertheless makes small movements in order to maintain its hover. To evaluate how the drone's measurement accuracy is influenced by its motion, we have the drone hover for 60 seconds. Then, we plot signal strength measured and the velocity of the drone. We then calculate the correlation between the two data sets to see the influence of the drone movement on the signal strength measured.

Results: We found that the signal strength measured by the drone is inversely correlated with the velocity of

the drone. When the velocity of the drone is higher, the value of the signal strength in negative dBm is higher, meaning the signal is weaker. Figure 5 shows the signal strength measured by the drone and the velocity of the drone. We see that signal strength measured by the drone in one location can fluctuate as much as 10dBm. This issue is reduced by taking the average of the measurements over a longer period of time, which we evaluate in the next subsection.

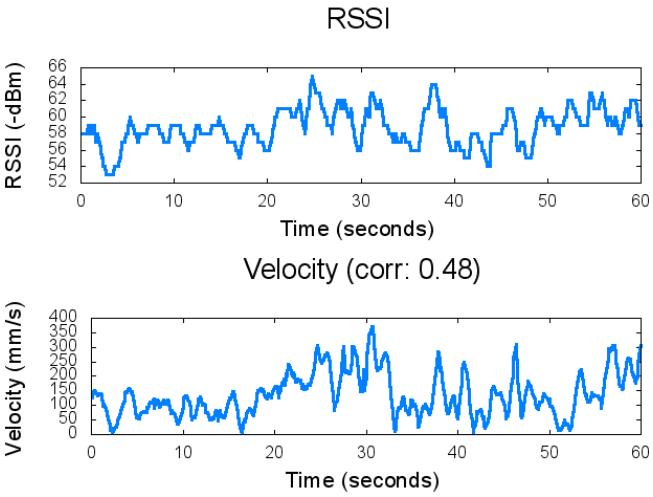


Figure 5: Correlation between signal strength and drone velocity

8.4 Measurement Time

Experimental Setup: In order to determine an adequate amount of time to collect measurements so that the effect of noise is negligible, we test collecting measurements at two locations for various amounts of time. These two locations have significantly different signal strength, and we test collecting measurements for 1, 3, 10, and 30 seconds. Then we calculate the mean and standard deviation of signal strength collected at each location for each duration and see at which duration the results start to converge.

Results: We found that 3 seconds is a sufficient length of time for the drone to collect measurements at each location. In Table 4, we list the mean and standard deviation of measurements with different duration at multiple locations.

Location	Duration (second)	Mean (dBm)	Standard Deviation (dBm)
a	1	-73.00	0.8165
	3	-74.33	1.18
	10	-74.35	2.34
	30	-73.78	2.15
b	1	-52.00	1.49
	3	-46.9	1.42
	10	-46.64	3.64
	30	-47.50	3.50

Table 4: Measurement Duration, Mean, and Standard Deviation

9. RELATED WORK

Indoor Walk Sensing Products: There are many commercial products that provide all-in-one solutions to facilitate the indoor wireless measurement process, such as the Agilent E6474A [1] and the Nemo Walker Air [14]. These products provide a set of equipment to carry in a backpack while walking around an indoor space, and are designed for comfort and portability. They are used to analyze network coverage and evaluate network performance, often performing activities such as web browsing, voice calling, text messaging, and video streaming while the user walks around. They require a floor plan to interpolate signal strength between measurement points, and rely on GPS for position estimation. One benefit of these products is being able to test various real world scenarios rather than simply measuring signal strength. The key advantage of our system is that it is automated and does not require human labor once set up, and does not require GPS.

Propagation Modeling: Studies on propagation modeling require actual signal collection for validation. Thus, research on wireless propagation modeling and our work go hand in hand, because as each model is developed, it needs to be tested for accuracy. For instance, a study based on propagation modeling, WiPrint, tries to shape the wireless coverage in the room [7]. One limitation of propagation models is that they are not fully accurate, which is why our system is complementary to these efforts.

Wireless Measurements : There have been extensive crowd sourcing efforts for wireless data. Two examples are CRAWDAD [12] and the Seattle WiFi Map Project [13]. These efforts provide valuable resources for the research community, but serve a slightly different purpose. Our system differs in that it is more suitable for real-time measurement at the current research location, rather than past data at the predetermined locations.

Drone Navigation: There have been extensive efforts in developing reliable navigation systems for drones. There are expensive drones on the market today that already include highly advanced navigation systems. Though these navigation systems may be commoditized in the future, there is still much value today in studying and developing drone navigation systems, especially with

low-cost drones accessible to everyone. Researchers at the Technical University of Munich have published several papers on visual navigation of drones [8] [9]. Their methods rely on visual tracking of objects to determine the drone's position and guide the drone to its target. They have implemented these methods in the open source project *tum_ardrone* [21], which is only compatible with Linux. While many of these methods are more advanced than the Reference Point Detection and Extended Kalman Filter we use, these researchers were trying to create a general purpose navigation system, rather a specific system for collecting wireless measurements. These systems emphasize adaptability to any environment, and its visual methods try to track the movement of any object in the environment rather than predefined reference points. So for our problem, our solution currently leads to greater accuracy, although these more complex solutions could be adapted to our purpose and result in even greater accuracy.

10. CONCLUSION AND FUTURE WORK

This thesis presents DroneSense, a system to automate collecting 3D wireless signal measurements using drones. Our system is able to reduce the amount of effort needed to collect measurements, and can be easily adapted to a variety of environments. To achieve this, we address challenges of accurate drone navigation and efficient flight trajectory, using methods of Reference Point Detection, Extended Kalman Filter, and Proportional-Integral-Derivative Controller. Our system uses the Parrot AR Drone 2.0 for its hardware, and JavaScript on the Node.js platform for its software. Below, we list some possible areas for future work.

Navigation Accuracy: The current system works well for collecting measurements at whole-meter intervals, with accurate navigation and efficient flight with an error tolerance of 20 centimeters. We hope to improve the accuracy of the system further to allow for sub-meter intervals. This can be achieved through improving the drone's position estimation system, upgrading the model of the drone to one with more accurate velocity readings, setting better parameters for the PID controller, placing additional reference points in the environment. This will help achieve greater resolutions in measurement collection and provide additional valuable data where whole meter intervals is too large.

Measuring Other Signals: The current system provides a way to measure wireless signal strength. However, there may be other items of interest that researchers would like to measure. We hope to adapt the system to collect these other types of signals. We could explore using an external attachment on the drone rather than rely on the drone's on-board software and hardware.

Reducing or Eliminating Reference Points: One major limitation of the current system is the need for placing reference points throughout the environment. We hope to explore methods to reduce or eliminate the need for reference points while maintaining accurate position estimates. One such method could be placing cameras in the room and using them to determine the drone's location at all times.

Handling Obstacles: The current system relies on the user to input a flight path that does not collide with any obstacles. Additionally, if the drone flies over a sizable object with height, it will increase its own height because it will use the object as the reference point for the ground. If the drone is able to avoid obstacles, it may be possible to develop a measurement option that does not require the user to input a flight plan-the drone will simply collect measurements everywhere in an environment.

Multiple Drones: Since our system is automated and does not require human supervision, there is no reason why the computer program could not be controlling multiple drones at once. We hope to extend the functionality of our system to work with multiple drones in order to speed up the measurement collection process. One simple approach is to simply run the current program simultaneously in multiple separate processes and have the drones cover different parts of the room. A more complicated approach will have the multiple drones in mind, and coordinate their paths in an algorithmically efficient way.

11. REFERENCES

[1] Agilent indoor wireless measurement system. `http://literature.cdn.keysight.com/litweb/pdf/5968-8691E.pdf`. Accessed May 25, 2016.

[2] ardrone-autonomy. `https://github.com/eschnou/ardrone-autonomy`. Accessed May 25, 2016.

[3] ardrone-wpa2. `https://github.com/daraosn/ardrone-wpa2`. Accessed May 25, 2016.

[4] AR Drone camera backproject. `https://github.com/tum-vision/ardrone_autonomy/blob/master/calibrations`. Accessed May 25, 2016.

[5] Bradski, G. OpenCV. *Dr. Dobb's Journal of Software Tools* (2000).

[6] BusyBox. `https://www.busybox.net/`. Accessed May 25, 2016.

[7] Chan, J., Zheng, C., and Zhou, X. 3d printing your wireless coverage. In *Proceedings of the 2nd International Workshop on Hot Topics in Wireless* (2015), ACM, pp. 1–5.

[8] Engel, J., Sturm, J., and Cremers, D. Camera-based navigation of a low-cost

quadrocopter. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on* (2012), IEEE, pp. 2815–2821.

[9] Engel, J., Sturm, J., and Cremers, D. Scale-aware navigation of a low-cost quadrocopter with a monocular camera. *Robotics and Autonomous Systems 62*, 11 (2014), 1646–1656.

[10] iwconfig. `http://linuxcommand.org/man_pages/iwconfig8.html`. Accessed May 25, 2016.

[11] Kalman, R. E. A new approach to linear filtering and prediction problems. *Journal of basic Engineering 82*, 1 (1960), 35–45.

[12] Kotz, D., and Henderson, T. Crawdad: A community resource for archiving wireless data at dartmouth. *Pervasive Computing, IEEE 4*, 4 (2005), 12–14.

[13] Marwick, A. Seattle WiFi Map Project. *Students of COM300, Fall* (2004).

[14] Nemo walker air indoor benchmarking wireless networks. `http://www.anite.com/sites/default/files/Nemo%20Walker%20Air%20Brochure%20Mar2016.pdf`. Accessed May 25, 2016.

[15] node-ar-drone. `https://github.com/felixge/node-ar-drone`. Accessed May 25, 2016.

[16] node-opencv. `https://github.com/peterbraden/node-opencv`. Accessed May 25, 2016.

[17] OpenCV Tutorials: *imgproc* module - Image Processing. `http://docs.opencv.org/2.4/doc/tutorials/imgproc/table_of_content_imgproc/table_of_content_imgproc.html`. Accessed May 25, 2016.

[18] Parrot. Parrot AR Drone 2.0. `http://ardrone2.parrot.com`. Accessed May 25, 2016.

[19] PID Controller Tuning: A Short Tutorial. `http://saba.kntu.ac.ir/eecd/pcl/download/PIDtutorial.pdf`. Accessed May 25, 2016.

[20] node-telnet. `https://github.com/mkozjak/node-telnet-client`. Accessed May 25, 2016.

[21] TUM Computer Vision Group. tum_ardrone. `http://wiki.ros.org/tum\_ardrone`. Accessed May 25, 2016.