

Flying, Room-Identifying, Autonomous Robot

Cycle 2 Design Report

Section 4 Group 3*: William Bates
Christopher Newton
Lee Richert
Trip Richert
Kyle Taylor

April 23, 2014

*Auburn University Spring 2014 Senior Design under Dr. Thaddeus Roppel

Executive Summary

This design report describes the design tradeoffs in our attempt to design an extension to the commercially available AR Drone 2.0. This extension, along with the commercial drone, together are described as the flying, room-identifying, autonomous robot (F.R.I.A.R.). The goals for the F.R.I.A.R. project are:

1. establish a two way wireless connection between a ground control station (laptop) and our robot to transmit commands
2. autonomously navigate down a straight hallway maintaining a perpendicular distance to the wall
3. detect obstacles of the robot, and avoid them if necessary
4. use AR Drone 2.0's built in camera to read room numbers from the number plaques of the Shelby Center at Auburn University.

Unfortunately, we ran into many problems in our attempt to accomplish these goals. Our prototype is wirelessly controlled manually from a laptop. It can detect obstacles but cannot navigate autonomously and does not have any on board image processing.

We also developed software to enhance the video stream from the commercial AR drone by removing the barrel distortion. This image enhancement software was not integrated into our prototype.

Parrot, the manufacturer of the AR drone, has licensing restrictions on software using their API to interface with their drone. Our software interfaces with their API, so we are restricted by their software license.

Contents

Executive Summary	i
List of Acronyms and Terms	iii
Acknowledgments	iv
1. Introduction	1
2. System Overview	1
3. Ground Station Communication	3
4. Obstacle Detection	4
4.1. Hardware Design	5
4.2. LV-MaxSonar-EZ0	5
4.3. Printed Circuit Board	5
4.4. Sensor Sampling Software	6
4.5. Design Tradeoffs	6
5. Navigation	7
6. Computer Vision	7
6.1. Distortion Correction	9
6.2. Feature Detection, Description, and Matching	13
6.3. Plaque Classification and Transform of Training Images	14
6.4. Digit Classification	15
6.5. Evaluation of algorithm	15
6.6. Computer Vision Summary	16
7. AR Drone 2.0 License	16
8. Budget	17
9. Schedule	17
10. Conclusion	17
Appendices	18
A. Distortion Model Comparison	19
A.1. Brown Distortion Model	19
A.2. OpenCV Distortion Model	19
B. Detailed Schedule Comparison	19

List of Acronyms and Terms

AR Drone 2.0:	The commercial drone that we are modifying for this project
the Embedded System:	the hardware extension onboard the altered AR Drone 2.0, including the Raspberry Pi processor and its peripherals.
F.R.I.A.R.:	This project: Flying, Room-Identifying, Autonomous Robot
GCS:	Ground Control Station, a stationary personal computer used to control the robot
the robot:	the combination of the AR Drone 2.0 and the Embedded System (essentially everything attached to the flying platform)
Xbee module:	Xbee brand of communication modules. We are using an Xbee module to communicate between the ground station and the embedded Raspberry Pi processor

List of Figures

1. Wireless communication between hardware systems: the GCS is represented by the PC, the Embedded System is represented by the Raspberry Pi processor, and the AR Drone 2.0 is represented by the picture of a quadrotor robot 2
2. Sonar positions on the drone. Green represents the sonar beams, black represents the sonars, and blue is the drone. 4
3. Sonar beam detection characteristics of LV-MaxSonar-EZ0 from [12]. . . 5
4. Printed circuit board layout for sonar sensor power and signal connections. The top layer is in green. The bottom layer is in red. Holes and pins are in blue. 6
5. Image from drone camera with barrel distortion. 9
6. An image created in MATLAB (left) and the same image after barrel distortion (right) 10
7. The original image (left), the distorted image (middle), and the restored image (right) 12
8. Using documented model for OpenCV, attempting to reverse the distortion in the left image produces the result on the right. Many pixels in the corners are not defined. 12
9. Distorted images (top and bottom left) and the corresponding images with distortion removed (top and bottom right) 13
10. The sampling density of I_u as a function of r', c' 14
11. An image of a plaque (left) and the feature points detected (right) 14
12. Matchings between points detected on a training image (right) and the current frame (left). 15
13. Timeline: Projected time worked that was worked is in green. Projected time overruns are in red. Time projected not spent is in white. 23

List of Tables

1.	Comparison of Computer Vision Programming Language Options	8
2.	Tradeoffs in Approaches to Inverting Brown's Distortion Equations . . .	11
3.	Normalized Cross-Correlations Between Training Digits	16
4.	Budget	17

Acknowledgments

We would like to acknowledge Dr. Roppel, for his guidance and facilitation of FRIAR; Dr. Murray, for his financial assistance and assistance in setting team goals; Jonathon Taylor, for his technical assistance with the Parrot Drone and Raspberry PI along with a base of C++ code to expand upon; Phillip Ferentinos for assistance with mounting the computing hardware as well as frame improvements; the Electrical Engineering Department, for their support of all of their students; and everyone else who is making this effort possible.

1. Introduction

The Flying, Room-Identifying, Autonomous Robot project is a hardware and software extension to the commercially available AR Drone 2.0. The goals of the project are

1. establish a two way wireless connection between a ground control station (GCS) and our robot to transmit commands
2. autonomously navigate down a straight hallway maintaining a perpendicular distance to the wall
3. detect obstacles of the robot, and avoid them if necessary
4. use AR Drone 2.0's built in camera to read room numbers from the number plaques of the Shelby Center at Auburn University.

We failed on many fronts. Our resulting prototype is uses sonar sensors to detect obstacles, but is manually controlled by a user over a wireless interface. Our room number plaque detection algorithm failed, but we did develop software to remove barrel distortion. This software was not integrated with the embedded system.

Dr. Murray, an associate professor in Auburn University's Department of Industrial and Systems Engineering, has provided our team with a Parrot AR 2.0 drone to use for the project. The previous groups working for Dr. Murray developed software to communicate between the Raspberry Pi and the drone.

PARROT's software license agreement restricts the use of their software exclusively to education and entertainment purposes. The obstacle avoidance capabilities of our robot could be useful for a variety of applications. Obstacle avoidance is a must for any autonomous robot, and owners of a drone for any user controlled game will likely feel much more comfortable about lending the game to a friend if the robot can prevent itself from crashing. The room-number identification feature is extremely useful for navigation in a variety of indoor applications.

We will now describe the organization of the rest of the report. Section 7 describes the PARROT licensing agreement and the effects on our project. Section 2 gives an overview of the entire system. Section 3 describes our progress on developing communication between the robot and a GCS. Section 4 describes the hardware involved in obstacle detection and the software that needs to be developed. Section 5 describes the current state of our navigation software and how we intend to improve it. Section 6 outlines our planned process for detecting and reading plaques from drone-collected images. Section 8 describes the budget of our project, and section 9 describes the project schedule. Section 10 states conclusions of this report. Details extending certain sections of the report are included in the appendices.

2. System Overview

The F.R.I.A.R. has three separate hardware systems: the Ground Control Station (GCS), the Embedded Controller, and the AR Drone 2.0. These three systems are

linked by wireless communication shown in Figure 1. Most of our work is on the Embedded System which manages three key subsystems: Obstacle Detection, Navigation and Obstacle Avoidance, and Computer Vision. The Navigation and Obstacle avoidance and the Computer Vision subsystems do not work.

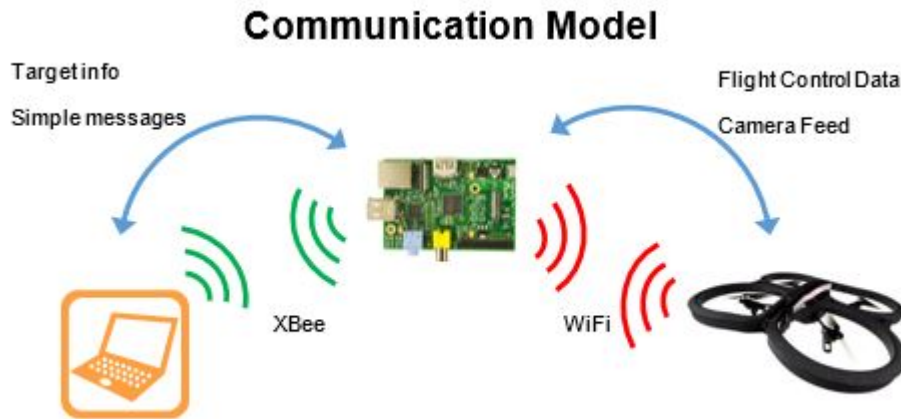


Figure 1: Wireless communication between hardware systems: the GCS is represented by the PC, the Embedded System is represented by the Raspberry Pi processor, and the AR Drone 2.0 is represented by the picture of a quadrotor robot

The Obstacle Detection subsystem consists of five sonar range finders and the software to read and interpret the data provided. The software is implemented in c++ and will run on board the Raspberry Pi. A printed circuit board was designed and manufactured to more reliably power and connect the sonar range finders to the Raspberry Pi.

The Navigation and Obstacle Avoidance subsystem is purely a software construct. The goal was for this system to take input from the obstacle detection subsystem to autonomously pilot the AR Parrot Drone. Unfortunately, we did not get the autonomous navigation to work properly. However, to debug our system, we did design a manual remote control of the AR Drone through the embedded system. This remote control interface works well.

The Computer Vision subsystem consists of the AR Parrot Drone's on board camera as well as software to recognize room numbers on the walls given an image file of the camera's view. Unfortunately, we did not manage to get the feature detection algorithms to detect room number plaques. We also did not manage to access the AR Parrot Drone's video stream.

Communication between these systems is vital to the correct operation of FRIAR. Built into the AR Parrot Drone is a wi-fi hot spot which shall communicate with the Raspberry Pi via the User Datagram Protocol (UDP). Navigation commands are sent over this bridge. The Sonars will communicate with the Raspberry Pi through a pulse width modulated signal. Finally, a ground station communicates with the Raspberry Pi via Xbee hardware. The remote control interface is through this link.

The above configuration was chosen after considering the alternatives and their tradeoffs and strengths. The Raspberry Pi was chosen for its small size and versatility. Although not exceptionally powerful, the Pi is a fully functioning computer the size of credit card and running on little power. The ARM architecture and Linux based operating system (Raspbian) allow the drone to run almost any software and can interface with most devices designed to run on a normal computer, such as wi-fi adapters and other USB devices. The Pi's general purpose input output (GPIO) pins allow various sensors to interface directly with the Pi. A microcontroller was considered in place of the Pi but while it did greatly expand the GPIO capability it would cost more, and add more weight, to interface the microcontroller with wi-fi and XBee than it would with the Pi. Wi-fi and XBee shields would need to be bought for the microcontroller. It would also limit the ability to run the Computer Vision subsystem on the drone itself to what packages and software the microcontroller can run.

The Ground Control Stations needed so that a user can communicate to the drone while in flight. We planned for the GCS is to tell the drone when to start searching and receive any signals from the drone while in flight or to update the drone's goal, but instead it is used for manual control of the drone. I

Custom code was written for the Pi to allow more flexibility with integrating the subsystems. There is open source code, such as JavaDrone and NodeJS, that would do most of what the project needed but it was determined that trying to modify the code for either one would be more difficult. The open source code available was also not meant to run on a Pi and the code may not have run as well on the Pi as on a computer. An example would be the time it takes to execute a system call, while both the Pi and a computer can execute the call the computer is much more powerful and faster than the Pi so runtime may be an issue.

3. Ground Station Communication

In order for the robot to talk back with the Ground Control Station (GCS) two Xbee modules are used. Originally the robot used the Mavlinks protocol to communicate with the GCS. This proved difficult and rather cumbersome for the minimum amount of communication needed between the robot and the GCS. In order to simplify and reduce the number of packets sent, simple ASCII character communication is used. This allows us to test the flight control command code and send simple predetermined messages between the robot and the GCS. Using this method we have successfully used to GCS to remotely control the robot and verify that the flight control codes function as expected. Not using the Xbee with the ZBee protocol is for the same reason as not using Mavlinks, all the features and associated complexity was not needed for the amount and type of communication needed. The Pi communicates with the AR Drone using UDP over an ad-hoc wi-fi connection. This is a feature of the AR Drone and could not be changed. A wi-fi USB adapter was added to the Pi so it could communicate with the AR Drone.

4. Obstacle Detection

As mentioned previously, the obstacle detection system will consist of both software and four sonar range finders. The particular range finder that we will use is an LV-MaxSonar-EZ1. This sonar can detect obstacles between 6 and 254 inches away. It produces 20 readings per second and can operate on 3.3 volts. Four of the five range finders are positioned as shown in the Figure 2. The fifth sonar is positioned on top of the drone for ceiling detection.

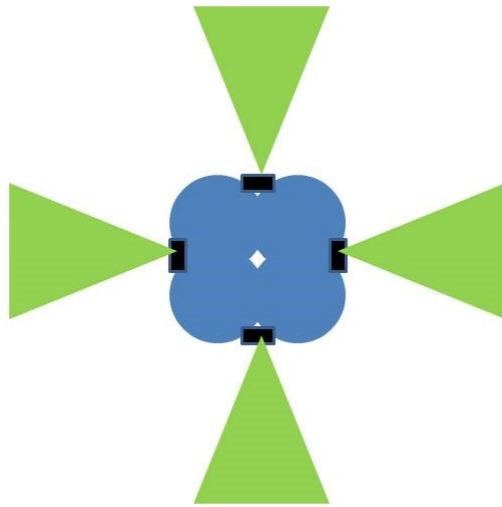


Figure 2: Sonar positions on the drone. Green represents the sonar beams, black represents the sonars, and blue is the drone.

Together, these range finders are able to determine if there is an obstacle immediately left, right, in front, or behind the Drone. In order to interpret this data however it must be connected to the general purpose input output (GPIO) pins on the Raspberry Pi. Over these connections, the sensors will emit a PWM signal with a frequency of 20 Hz. To read this signal, interrupt driven software will be used to interrupt on both rising and falling edges. The rising edge will start a timer and the falling edge will stop the timer. The time difference will then be proportional to the distance the object is away from the sensor. One issue with this technique is that the Raspberry Pi cannot queue more than one interrupt at a time, thus, it seemed that some sort of scheduling technique will be developed to ensure all sensors are read accurately, and none are drowned out by the others interrupt requests. Unfortunately, the only known technique for enabling and disabling interrupts is too computationally expensive to use practically in an algorithm. Fortunately, through counting interrupts over a given period of time, it was determined

that the sensors interrupt evenly enough that the sensors do not drown themselves out.

4.1. Hardware Design

Our main hardware features includes four LV-MaxSonar-EZ0 range finding sensors being read by the Raspberry Pi embedded computer. Each of the four sensors can output data in a variety of formats including PWM. The PWM signal was chosen as the output signal to be used for reasons listed below in the tradeoffs section. To physically connect the four sonar sensors to the Raspberry Pi required 12 wires, or three wires per sonar (positive supply, negative supply, and PWM). The Pi has 17 general purpose pins that can be used for PWM signals, but only two pairs of supply pins. Due to this limitation, as well as previous reports of power delivery errors, it was decided that a printed circuit board would be designed to deliver power to all four sonar sensors.

4.2. LV-MaxSonar-EZ0

This sonar sensor is capable of detecting objects between 6 and 254 inches with a 1 inch resolution. It produces 20 readings per second and is able to output the readings in three formats: PWM, RS232 serial, and analog voltage. It can use either 3.3V or 5V input voltage. The following diagram from the LV-MaxSonar-EZ0s datasheet shows its sonar beam detection characteristics.

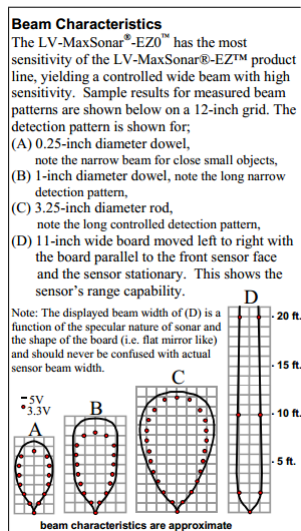


Figure 3: Sonar beam detection characteristics of LV-MaxSonar-EZ0 from [12].

4.3. Printed Circuit Board

For reliability issues involving power distribution, a printed circuit board was designed to facilitate PWM and power connections to each sonar. Because printed circuit boards can be costly, the board was designed to be expandable and redundant. This expandability

mainly comes from the extra pads added to handle up to ten EZ0 sonar sensors on a single board. In addition to this, extra pads were placed on the board to allow for power filtering through the use of bypass capacitors if needed. Drill holes were placed to help mount the board to the Raspberry Pi. Below shows the PCB design, with green as the top layer and red as the bottom layer.

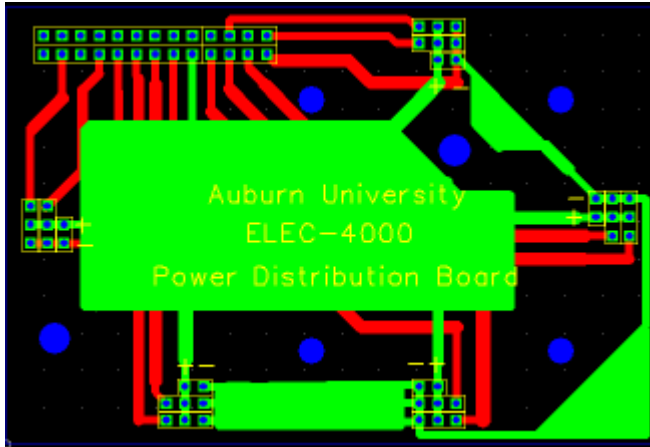


Figure 4: Printed circuit board layout for sonar sensor power and signal connections. The top layer is in green. The bottom layer is in red. Holes and pins are in blue.

4.4. Sensor Sampling Software

The software for sampling each sonar sensor primarily relies on the interrupt driven design principal. When a PWM signal rises or falls, an interrupt service routine (ISR) is entered. If the ISR is a rising edge, it takes the current time and stores it. If the ISR is a falling edge, the current time is subtracted by the time previously stored. The distance that the sonar sensor is directly proportional to the time difference between the two clocked times. Four of these ISRs are implemented to each service one sonar sensor.

4.5. Design Tradeoffs

Since the Raspberry Pi does not support analog input pins, only the RS232 or PWM signals will be feasible to use for sonar data communication. The RS232 serial signal runs at 9600 baud and requires 5 bytes of data per sonar reading, thus requiring a time of approximately 5 milliseconds per reading. For the 700 mhz processor on the Raspberry Pi, this is equivalent to 3.5 million operations, which is much more than the PWM signal sampling would require. Due to these factors, PWM was the chosen method of input.

Another design issue was whether or not to schedule the interrupts from the sonar sensors. The problem with allowing free running interrupts as we did was that the Raspberry Pi does not queue more than one interrupt at a time. In other words, if the code was already in an ISR, and two more interrupts occurred before the code finished,

one of those interrupts would simply be dropped. To solve this, a round-robin scheme was developed to circularly turn on and off interrupts. This was effective to not crowd out any interrupts, but the system call required to enable or disable interrupts was too time costly. In the end, it was decided that it was unlikely that three interrupts would occur at once, and through experimentation, it was discovered that this rarely happened at all.

5. Navigation

In any autonomous or semi-autonomous robot, obstacle avoidance is a must. Our design uses sonars to sense around the robot. These exteroceptive sensors serve two purposes. Primarily, the prevent the robot from colliding into objects. The sonars are also used for navigation. A previous research group developed software to right wall follow using sonar sensors, but they were unable to test it because they did not manage to interface well with their sensors. We analyzed the code to predict what modifications would be needed.

The basic flow of the program begins after initializing variables, the drone begins to search for a wall on the right. If it does not if it finds one, the drone turns until the wall is on the right otherwise, it moves forward until it find an obstacle/wall. When a wall is found, it sets wall located to true so that it does not attempt to go through the wall. Next the drone checks if there is an obstacle on the right, and no obstacle in front. The drone then will proceed forward until an obstacle is found. When an obstacle is detected in front of the drone, it will either turn right or left. We intend to modify this portion of the code so that it hovers for a moment or two before attempting to go around the obstacle. When an obstacle (the wall) is no longer detected on the right, the drone will turn, and hover momentarily. Currently the program instructs the drone to turn until it finds a wall on the right, after stabilizing the drone then proceeds forward.

We predict that the previous group's algorithm is not robust to some circumstances, such as tight corners or recessed doors, so the algorithm will be modified to pass these edge conditions.

6. Computer Vision

The goal of the computer vision component of the F.R.I.A.R. project was to develop code to process the video feed of the PARROT drone on the raspberry pi. This code was to detect room number plaques and read room numbers. Constraints on the design include processing power, available memory, and processing time. This task was broken into the following basic steps:

- Distortion correction
- Feature detection
- Feature description

- Feature matching
- Plaque classification
- Transform of training images
- Digit Classification

Distortion correction removes the distortion caused by the fisheye lens of the camera. Feature detection selects a set of locations in the current frame. Feature description extracts descriptors of those locations. Feature matching compares those descriptors to those extracted from feature locations on a training plaque. Homography estimation uses the spatial geometry of the feature matches to estimate a transformation mapping locations in the training plaque to locations in the current frame. The transformation generates a transformed plaque from the current frame which is the same size and orientation as the training plaque. Plaque classification classifies the transformed image as either a plaque or not a plaque. Digit classification reads each image on the plaque.

Programming Language

In development of this code, we had to choose a programming language. Table 1 shows some tradeoffs involved in this decision, where 1-3 are rankings between the different options. C++ was chosen because of its ability to run on the raspberry pi, and for compatibility with code being developed by other members of the team in C++. Its computational speed is also a plus.

Table 1: Comparison of Computer Vision Programming Language Options

	MATLAB	C++	PYTHON
Familiarity with language	Yes	No	No
Able to run on Raspberry Pi	No	Yes	Yes
Development speed	fast	slow	medium
Computational speed	slow	fast	medium
integration with embedded software	need to translate	same language	feasible

Development Environment

We considered three development environments for our computer vision code: Windows, Ubuntu VM, and a Raspberry Pi VM. Windows was already set up and ready to go, but is least like the final intended running environment of the code. This could potentially lead to operating system related bugs upon integration. The raspberry pi VM is the most like the final running environment. However, if memory constraints hindered progress, optimization and development could be necessary to attempt at the same time, which tends to lead towards less readable code and more bugs. The Ubuntu VM offered the best of both worlds, close enough to the final environment to avoid serious bugs upon integration, but avoiding potential memory and speed issues that could hinder the pace of development. For these reasons, we chose to develop on a Ubuntu virtual machine.

6.1. Distortion Correction

General problem and approach

The drone camera uses a fisheye lens, which causes distortion towards the corners of the image. This can be seen in Figure 5, which shows an image captured by the PARROT drone. In any detection process, it is desirable for the representation of the sought object to be independent of its position in the image. The simplest means to way to achieve such a representation is to develop a mapping from pixel locations in an undistorted image representation I_u to the corresponding locations in the distorted image I_d . We



Figure 5: Image from drone camera with barrel distortion.

outlined our distortion correction process as follows:

- Model the distortion
- Generate mapping from undistorted locations to distorted locations
- interpolation (bilinear)

We noted that the only operation of these that is necessary to be computed on the raspberry pi is the interpolation. For code not designed to run on the raspberry pi, the limitations on coding language and development environment are not relevant. We used the languages of MATLAB and C++ for tasks within modeling and mapping.

The Model

From visual inspection of video collected from the PARROT drone, we judged that the distortion could be characterized as barrel distortion. Images depicting barrel distortion are shown in Figure 6. [h] Brown [3] suggests a model that is equipped to handle barrel distortion. This model maps coordinates from the distorted domain to the undistorted domain and is defined in Appendix A.

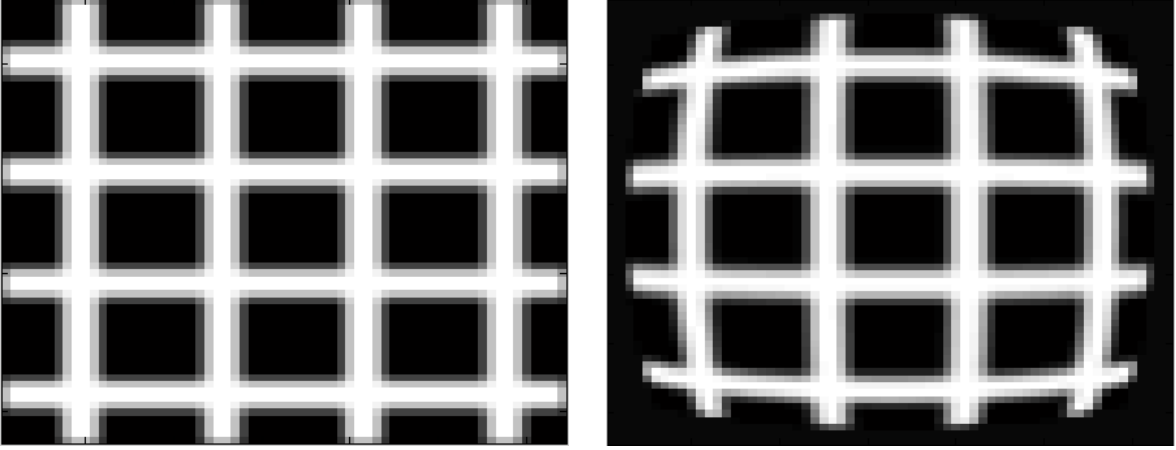


Figure 6: An image created in MATLAB (left) and the same image after barrel distortion (right)

Generating the needed mapping

The Brown distortion equations map coordinates from the distorted image to the undistorted image. The mapping required in the restoration process maps coordinates from the undistorted image to the distorted image. This means some sort of inversion process is required in order to use our chosen model. Brown's distortion equations are nonlinear and multivariate. Finding their inverse is a nontrivial problem. However, mappings are only needed from pixel locations in the undistorted image. If closely enough spaced points from the distorted image are mapped to the undistorted image, an approximate inverse could be computed for the necessary locations.

Of course, we would prefer not to compute the distance between every undistorted pixel and the locations in the undistorted image mapped from locations in the distorted image. One way to avoid this is to map regions of pixels from the distorted image to the undistorted image, and only compute distances for undistorted pixels in the corresponding region in the undistorted image. If we ignored the highest order parameters K_3 and P_3 , we could compute the borders of this region in the undistorted image closed-form using quartic equations, and then from them construct inequalities defining the region. However, quartic equations tend to not have completely stable computations [13], and it is not a simple task to develop a vectorized approach for MATLAB code.

Given these limitations, we opted for a simpler approach. Instead of computing the region of the undistorted image exactly, we instead calculate a larger region that encompasses it. We determined the size of the larger region by mapping the corners of a rectangular region in one quadrant of the distorted image to their corresponding locations in the undistorted image (the sides of this rectangle are parallel to the axis). Since within quadrant undistorted vertical and horizontal lines map to monotonic functions, the rectangular region in the undistorted image containing all four mapped points will encompass all the locations mapped from the corresponding region in the distorted image. This approach imposes no limitations on the parameters of the model. The

tradeoffs between these different approaches are summarized in Table 2. We decided to use approach C.

Table 2: Tradeoffs in Approaches to Inverting Brown’s Distortion Equations

Approach		Advantages	Disadvantages
A	Compare pixel locations in I_u to all point locations mapped from I_d	Simplest to code	Requires highest computational time memory usage
		Independent of model	Unable to distinguish whether a pixel location of I_u maps outside of I_d for some pixels, so cropping is necessary.
B	Use quartic equations to produce inequalities defining regions of I_u corresponding to rectangular regions of I_d	Only requires comparison of pixel locations of I_u to mapped point locations from the corresponding region of I_d	Requires simplified distortion model
		Able to identify all pixel locations in I_u that map outside of I_d	Difficult to code
			Computationally unstable
C	Compute larger rectangular regions of I_u encompassing the region corresponding to rectangular regions of I_d	Does not require a simplification of Brown’s distortion model	Unable to distinguish whether a pixel location of I_u maps outside of I_d for some pixels, so cropping is necessary.
		Requires significantly less processing time and memory than approach A	
		Simpler to code than approach B	

Testing Mapping Procedure

We implemented MATLAB code to produce a distorted image from an undistorted image, given a set of distortion parameters. With a distorted image of known distortion parameters, we tested our restoration process on synthetic images (one test is shown in Figure 6). In this manner, we were able to test this process independently from parameter estimation. In our tests, the restored images were visually indistinguishable from the originals.

Getting the Model Parameters

To calculate the model parameters, we used open source code available in the sample code of the OpenCV library. Our camera calibration process is explained in the user manual.



Figure 7: The original image (left), the distorted image (middle), and the restored image (right)

Testing the Model Parameters

To test the model parameters, we used the code described in the Generating the Needed Mapping section to attempt to remove the distortion on images captured on the camera of the PARROT drone. The first attempt is shown in Figure 8. As can be seen, the distortion worsened. The operation on the image appears to be the inverse of the desired one.

Updating the model to match the observed behavior of the OpenCV output, we see improved results as shown in Figure 9. This observed OpenCV model is described in Appendix A. This model maps coordinates locations in the undistorted image to coordinate locations in the distorted image, so the inversion process described in the section Generating the Needed Mapping is no longer necessary.

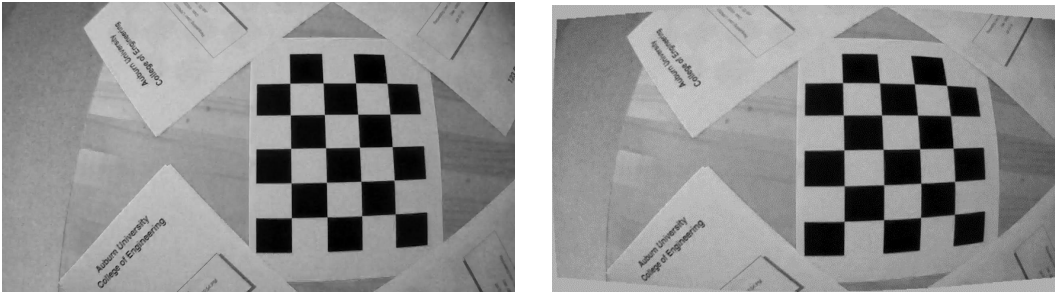


Figure 8: Using documented model for OpenCV, attempting to reverse the distortion in the left image produces the result on the right. Many pixels in the corners are not defined.

In this distortion removal process, we are altering the sampling density (samples per pixel). This leaves the potential that aliasing could occur. Assuming uniform sampling density in the distorted image, if r, c are the coordinates of the distorted image, and r', c' are the coordinates of the undistorted image, then sampling density S can be calculated



Figure 9: Distorted images (top and bottom left) and the corresponding images with distortion removed (top and bottom right)

as in equation 1.

$$S = \left| \frac{\delta r \delta c}{\delta r' \delta c'} \right| \quad (1)$$

We calculated the sampling density for our distortion parameters. The resulting sampling density image is shown in Figure 10. As can be seen, there is no risk of substantial aliasing. Looking at row and column partial derivatives separately, row and column respective sampling densities are 1.0014 samples/pixel and 1.0008 samples/pixel respectively.

6.2. Feature Detection, Description, and Matching

Assessing the plaque-likeness of every single pixel location would be very computationally expensive. Feature detection is the selection of regions in the image that are likely informative and insensitive to perspective and illumination changes. We decided to use FAST for feature detection. The feature detection algorithm is built into OpenCV and is described in [4] and [5]. We use the BRIEF feature description algorithm will be used to describe those locations [10]. Neither FAST nor BRIEF is robust to scale changes, so we constructed an image pyramid (a stack of images that have been smoothed and down-sampled to various degrees) and ran FAST on each pyramid level. Each BRIEF descriptor was extracted from the same pyramid level that the feature point was detected.

For each detected feature in the drone camera image, the training descriptor of minimum Hamming distance is considered its match. These matches were then filtered

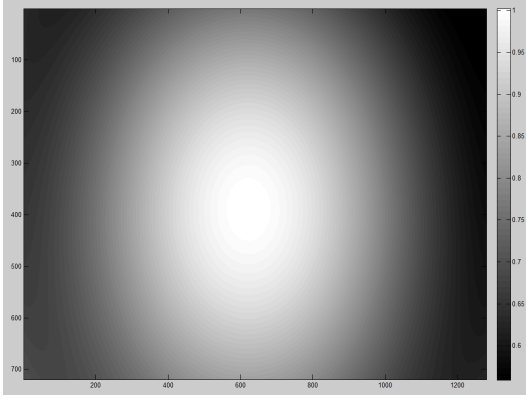


Figure 10: The sampling density of I_u as a function of r', c' .

through thresholding. Example screenshots of feature detection and feature matching are shown in Figure 11 and Figure 12 respectively.

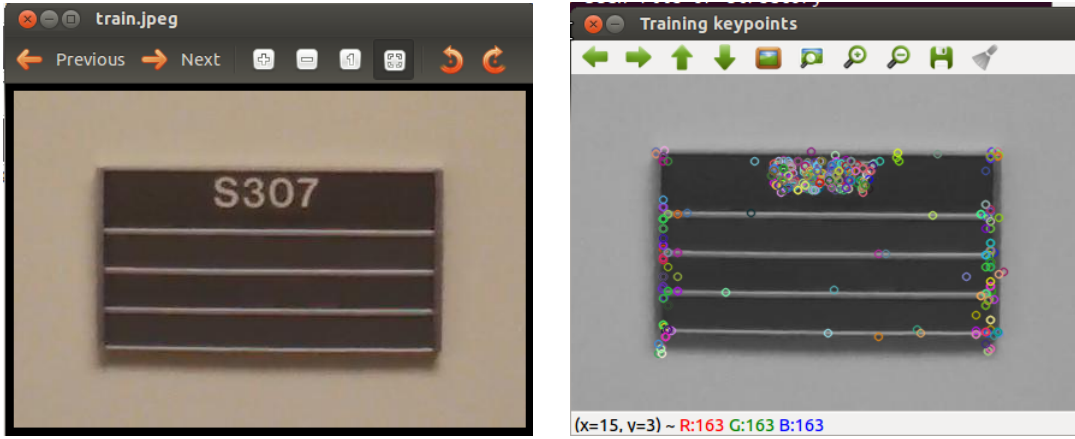


Figure 11: An image of a plaque (left) and the feature points detected (right)

6.3. Plaque Classification and Transform of Training Images

Given a set of thresholded matches found as described in subsection 6.2, an appropriate transformation must be determined so that the numbers can be classified. We used the RANSAC algorithm built into OpenCV to estimate the orientation based on the spatial locations of the matches. RANSAC is robust to outliers, so for a given set of matches, it can find an orientation even if some of the matches are incorrect.

Once the homography was estimated, we used it to map the training image to the corresponding locations in the drone camera image. Through bilinear interpolation, we constructed the current frame equivalent of the training image, with identical size and orientation. Once the potential plaque has been aligned with the training image, we use normalized cross-correlation to compare the two images. We mask out the locations of the digits so that they do not contribute to this confidence measure. If the correlation

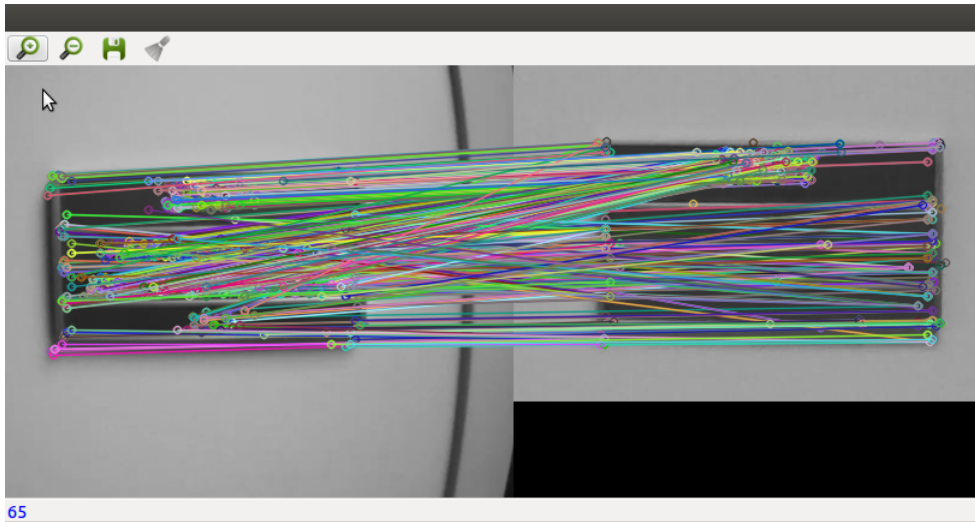


Figure 12: Matchings between points detected on a training image (right) and the current frame (left).

is above 0.9, the detected potential plaque is classified as a plaque, and our program proceeds to digit classification.

6.4. Digit Classification

The locations of the room number digits on training images are known. Given a linear transformation mapping the training images to the drone camera image, expected locations for each digit in the drone camera image are known. We collected images of digits on plaques and resized them to the scale of the digits on the training plaque.

To classify digits, digit images are extracted from the newly classified plaque. These digit images are thresholded by their arithmetic mean, so that a pixel is 1 if it is greater than the mean and 0 otherwise. The thresholded digit images are then compared with the collected digit images through cross-correlation. Each digit on the plaque in the drone camera image is assumed to be in the same class as the training image digit that correlates highest with it. We calculated the cross-correlation between our training digits, shown in Table 3. Larger values in the off diagonal suggest a higher likelihood of a digit of the type of that row being misclassified as the corresponding column digit type.

6.5. Evaluation of algorithm

In testing, we were unable to detect plaques and thus unable to classify digits. FAST feature detection predictably selected locations with detail, such as the corners of the plaque and its lettering. Many matches were correct and many more mapped to very similar-looking regions. However, due to the lack of detail on the plaque and that many parts of the plaque look very similar, and the algorithm ultimately was not robust

Table 3: Normalized Cross-Correlations Between Training Digits

	0	1	2	3	4	5	6	7	8	9	S
0	1	0.216493	0.376813	0.530408	0.232386	0.513008	0.635605	0.178364	0.602058	0.635074	0.525822
1	0.216493	1	0.426928	0.453184	0.423459	0.315046	0.274686	0.325746	0.328872	0.318676	0.342755
2	0.376813	0.426928	1	0.506675	0.237537	0.324227	0.343133	0.42865	0.408901	0.457231	0.420339
3	0.530408	0.453184	0.506675	1	0.297511	0.558521	0.552304	0.333717	0.647519	0.516464	0.590467
4	0.232386	0.423459	0.237537	0.297511	1	0.270157	0.314545	0.173128	0.33	0.234436	0.259168
5	0.513008	0.315046	0.324227	0.558521	0.270157	1	0.638289	0.219216	0.601602	0.476769	0.595664
6	0.635605	0.274686	0.343133	0.552304	0.314545	0.638289	1	0.213383	0.707524	0.538491	0.588077
7	0.178364	0.325746	0.42865	0.333717	0.173128	0.219216	0.213383	1	0.272895	0.269547	0.265671
8	0.602058	0.328872	0.408901	0.647519	0.33	0.601602	0.707524	0.272895	1	0.582703	0.723266
9	0.635074	0.318676	0.457231	0.516464	0.234436	0.476769	0.538491	0.269547	0.582703	1	0.602215
S	0.525822	0.342755	0.420339	0.590467	0.259168	0.595664	0.588077	0.265671	0.723266	0.602215	1

enough to handle this problem. Another difficulty with our approach was that the feature detection algorithm tends to select features on the lettering, which has no match in the other frame.

Given more time, we could limit orientations to more local spreads, which could make the algorithm more robust to false positives and filter out the high frequency content for feature detection so that the letters would not be so devastating to our approach. Alternatively we could switch to a line-based approach, which likely would demonstrate a higher performance in such a low-detail environment.

Despite failing to locate plaques, we did successfully implement our algorithm and were able to demonstrate its shortcomings toward this task.

6.6. Computer Vision Summary

We were able to remove through software the distortion caused by a fisheye lens. We encountered errors in the documentation of OpenCV parameter estimation, but we were able to adjust the distortion model to fit observed behavior of the output of OpenCV. We did not use the model to produce distortion removal code on the raspberry pi. We successfully implemented a feature algorithm to estimate the orientation of a plaque in an image, but the algorithm itself was not robust enough to handle the problem

7. AR Drone 2.0 License

In our initial proposal, one of our proposed goals was for our robot to autonomously map floor plans. Parrot, the company that produces the AR Drone 2.0 that our project is based on, attaches a license agreement to their documentation prohibiting the use of their API, SDK, and documentation for any purpose other than education or entertainment. The license agreement also explicitly prohibits cartography[1]. Our software, and most software available for the AR Drone 2.0, heavily relies on the AR Drone 2.0 API described in[2], so we had to change the objectives of our project. However, our design plan was not affected because we believe that the obstacle avoidance and computer vision capabilities of the proposed F.R.I.A.R. will be useful in educational and entertainment purposes.

8. Budget

All items on the budget are per unit costs that would be recurring costs if this product was manufactured.

Table 4: Budget

Product	Quantity	Unit Price	Total Price	Purchaser	Vender
AR Parrot Drone 2.0	1	\$278.07	\$278.07	Dr. Murray	Amazon
Raspberry PI 2.0	1	\$41.00	\$41.00	Dr. Murray	Amazon
LV-MaxSonar-EZ1	5	\$29.95	\$149.75	Dr. Murray	maxbotix
AR Parrot Drone Battery	1	\$49.99	\$49.99	Dr. Murray	Amazon
Xbee	2	\$22.95	\$45.90	Dr. Murray	sparkfun
Xbee USB Adapter	1	\$24.95	\$24.95	Dr. Murray	sparkfun
Xbee Breakout Board	1	\$9.95	\$9.95	Dr. Murray	sparkfun
Wiring	3	\$2.50	\$ 7.50	Dr. Murray	sparkfun

Total Cost \$599.61

Our budget has not swayed very much at all from our original budget. We are using sonars and wires already in the lab, so we did not spend money on either from the initial budget.

9. Schedule

Our initial projected schedule was very ambitious. We did not get everything we wanted to accomplish done. A detailed comparison of the present schedule and the projected schedule is shown in Appendix B.

10. Conclusion

We set ambitious goals for the F.R.I.A.R. project, and we fell substantially short. However, we did develop some significant deliverables. Our mounted embedded system with sonar sensors can be used by future developers to develop autonomous navigation of the AR Drone. Our manual control interface is useful for debugging such a system. The distortion removal model and software can be used in a variety of applications. Anyone using the webcam on the AR drone will likely be interested in restoring the image by removing the barrel distortion. Our barrel distortion removal software also could be useful in removing barrel or pincushion distortions from other cameras, and our user manual describes in detail how to use the OpenCV software to measure the parameters to then use the distortion removal software. The feature detection software does not work because the object of interest (room number plaques) is too smooth. The feature detection software likely could be adapted to locate more distinct objects.

References

- [1] url: https://projects.ardrone.org/attachments/376/PARROT_License_JUNE_2011_v2_1.txt
- [2] S. Piskorski; N. Brulez; P. Eline, "AR Drone Developers Guide SDK 1.7." Parrot. May 2011.
- [3] D. C. Brown, "Close-range camera calibration," *Photogrammetric Engineering*, 37(8):855866, 1971.
- [4] E. Rosten and T. Drummond, "Machine learning for high speed corner detection," in *9th European Conference on Computer Vision*, vol. 1, 2006, pp. 430443.
- [5] E. Rosten, R. Porter, and T. Drummond, "Faster and better: a machine learning approach to corner detection," in *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2010, vol 32, pp. 105-119.
- [6] C. Harris and M. Stephens, "A combined corner and edge detector," *The Plessey Company plc*, 1988.
- [7] Agrawal, M., Konolige, K., & Blas, M. R. (2008), "CenSurE: Center surround extremas for realtime feature detection and matching." In *Computer VisionECCV* 2008 (pp. 102-115). Springer Berlin Heidelberg.
- [8] J. Matas, O. Chum, M. Urban and T. Pajdla, "Robust wide baseline stereo from maximally stable extremal regions," *British Machine Vision Conference*, Cardiff, Wales, pp. 384-396, 2002.
- [9] C. Schmid, R. Mohr, and C. Bauckhage, "Evaluation of interest point detectors," *International Journal of Computer Vision*, vol. 37, no. 2, pp. 151172, 2000.
- [10] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," In *European Conference on Computer Vision*, 2010.
- [11] E. Dubrofsky, "Homography Estimation," University of British Columbia, 2009, pp. 1012.
- [12] http://www.maxbotix.com/documents/MB1000_Datasheet.pdf
- [13] <http://www.ijpam.eu/contents/2011-71-2/7/7.pdf>

Appendix A Distortion Model Comparison

A.1 Brown Distortion Model

Brown [3] suggests a model (2) can be used to represent barrel distortion (shown in Figure 6):

$$\begin{aligned} x' &= x + x \left(\sum_{n=1}^3 K_n r^{2n} \right) + (P_1(r^2 + 2x^2) + 2P_2xy)(1 + P_3r^{2n}) \\ y' &= y + y \left(\sum_{n=1}^3 K_n r^{2n} \right) + (2P_1xy + P_2((r^2 + 2y^2)))(1 + P_3r^{2n}) \end{aligned} \quad (2)$$

in which x, y denote the coordinates corrected for distortion, x, y denote the measured coordinates, P_1, P_2 , and P_3 are coefficients of decentering distortion, and K_1, K_2 , and K_3 are coefficients of radial distortion. x', y' and x, y are shifted so that the center of distortion is at the origin, and both sets of coordinates are normalized by the focal length. The variable $r = \sqrt{x^2 + y^2}$ represents the normalized distance from the center of distortion in the distorted image domain.

A.2 OpenCV Distortion Model

While OpenCV claimed in their documentation to be using the Brown distortion model described above, we observed a different model from the distortion parameters calculated using their code. The following model is our revised model for compatibility with their numbers:

$$\begin{aligned} x &= x' + x' \left(\sum_{n=1}^3 K_n r'^{2n} \right) + (P_1(r'^2 + 2x'^2) + 2P_2x'y')(1 + P_3r'^{2n}) \\ y &= y' + y' \left(\sum_{n=1}^3 K_n r'^{2n} \right) + (2P_1x'y' + P_2((r'^2 + 2y'^2)))(1 + P_3r'^{2n}) \end{aligned} \quad (3)$$

in which x, y denote the coordinates corrected for distortion, x, y denote the measured coordinates, Once again, P_1, P_2 , and P_3 are coefficients of decentering distortion, and K_1, K_2 , and K_3 are coefficients of radial distortion. Like before, x', y' and x, y are shifted so that the center of distortion is at the origin, and both sets of coordinates are normalized by the focal length. The variable $r' = \sqrt{x'^2 + y'^2}$ represents the normalized distance from the center of distortion in the undistorted image domain.

Appendix B Detailed Schedule Comparison

The timeline in Figure 13 is split into many different tasks, which are described below.

Concept

The working concept is a short description of our project.

Proposal

The proposal outlines the objectives of the project. The objectives of the project were revised to comply with Parrot's software license.

Design Report

This document is the design report. The design report is a technical document summarizing design specifications, justifications for engineering decisions, simulation results and evaluation, a budget, and other relevant information.

Website

The website of the team shows the progress of the design team on this project and showcases our work.

Presentation

Presentations on the status of this project are given after the proposal phase, cycle 1 of the design phase, and cycle II of the design phase.

Concept Video

A video illustrating the objectives of this project was developed for the proposal phase of our project. Another video was developed demonstrating the capabilities of the project.

Status Reports

A weekly status report was produced summarizing hours worked, weekly accomplishments, new obstacles, and other information.

Order Parts

Dr. Murray has generously offered to purchase this team an AR Parrot Drone and other parts related to our project.

OA Sensors

Sonar sensors were mounted onto the drone to detect obstacles.

Sensor Integration

A printed circuit board was designed to connect the sonar sensors to the Raspberry Pi. Software was developed to read the sonar sensor data using the Raspberry Pi.

OA Software

The sonar sensors were successfully used to detect objects and to maintain perpendicular distance to a wall. A glitch in our drone prevented us from rotating the drone. As a result, full navigation, including obstacle avoidance, was not implemented.

Power Supply

We had initially hoped to replace the battery with a longer lasting battery. We ran out of time and did not replace the battery.

RC Comm

In order to test the obstacle avoidance software, a remote control communication protocol to give instructions to the robot was designed.

Auto RC Comm

For the safety of the robot and nearby humans, an autonomous robot should still have some human control. This communication will require less human intervention than the remote control communication, so the protocol will likely be slightly different. We did not complete this aspect of our project.

Navigation

Our goal is for the robot to autonomously navigate by following the wall.

Plaque Detection

The robot needs to be able to identify plaques to simplify the number recognition problems. We failed this aspect of the project.

Number Recognition

The robot needs to be able to read room numbers autonomously. We failed the plaque recognition part of the project, so the code developed for the room number reading was never fully tested.

User Manual

Our project is meant to be an extension of the commercially available AR drone 2.0. The User Manual document provides instructions for the use, assembly, and storage of our robot, as well as safety recommendations and other useful information.



Figure 13: Timeline: Projected time worked that was worked is in green. Projected time overruns are in red. Time projected not spent is in white.