CS 4758/6758: Parrot Sentry Drone

Job Davis, Electrical and Computer Engineering, Senior, jmd343@cornell.edu

Brian McGlade, Computer Science, Senior, bdm38@cornell.edu

Abstract: Our project goal was to create a robot that can hover stationary over a given location, until a person with a badge wants to pass the location it, at which time it will move away from the location until the person passes. Once the person passes, the robot will move back to its original location. We used the Parrot AR Drone and built our code off of the existing codebase for the robot. We used the Speeded Up Robust Features (SURF) algorithm to compare an image of the badge to the image fed from the camera and determine if the badge has been presented to the robot. We used a logistic regression of our training data in our testing. We gave movement commands to the robot based on a state machine where the number of times the robot moved was stored and when no badge was present, the robot would reverse the movements it took. We also used a colored place marker on the ground to indicate the point where the robot would hover over, and used a color filter to allow the robot to make fine adjustments to return to the point by directing it to the center of the marker.

Keywords: Object detection, Object avoidance, Logistic Regression

Robot: The Parrot AR Drone

I. Introduction

The goal of the Parrot Sentry Drone (PSD) was to create an autonomous sentry, which would hover and keep watch around a fixed location until a moving object (a person) attempts to pass through said location. The moving person has a specific badge attached to it that allows the robot to recognize it as an object that it should let pass. For the badge, we choose to a Cornell University ID because of its distinct look and prevalence on campus. The person attempting to pass the PSD would hold their badge in front of the front camera of the robot, so that it could compare it with an image of the badge itself and determine that it needs to move to let the person pass.

The badge detection function that we implemented takes in a pre-stored image of the badge and the last image imported from the front camera of the drone and returns a value corresponding to whether or not the badge was detected in the camera image. The badge detection algorithm used the Speeded Up Robust Features (SURF) algorithm to extract key points from a base image of the badge and the image from the front camera of the drone. The keypoints were compared using a nearest neighbor algorithm and any matching points were linked together. Using training images with positives having the badge in the image and negatives not having the image, we created a logistic regression with the input being the number of matching pairs and the output being whether or not the badge was in the image. We used this regression to determine whether or not the robot should move.

When the robot detected a badge, it had to move from its sentry point to allow the person to pass. The robot was coded to move to the right when it was in the avoidance state. When the robot no longer saw a badge, it needed to move back so it would reverse every movement to the right by moving to the left an equal number of times. In order to minimize the distance that robot moved from its starting point, the robot also rotated to the left when in avoidance mode. This reduced the chances of it losing its starting point and kept it facing the person so that they could continue presenting the badge for as long as they needed to pass the robot. The rotations to the left were also reversed with rotations to the right.

We used the image feed from the bottom camera in a separate function that isolated the sentry point marker on the ground using its color and found its center. The robot was given small movement commands so that it would attempt to line up the center of its bottom camera with the center of the marker on the ground.

There was no prior work done on this project before this course. Our offline results were fairly reliable in detecting a badge presented to a camera. However, when we started online testing, we had a few sources of inaccuracy that led to errors in some of our results. Despite this, our PSD performs its intended task of moving out of the way when presented with a given badge.

II. Robot

We used the Parrot AR Drone with a front and bottom camera for this project. We also used an Xbox 360 controller attached to the computer running the codebase to alert the robot to take off and switch to the algorithm we implement in planner.cpp. Before running the AR Drone solely using the algorithm, we used the controller to manually guide it to the sentry point and steady it over it.

III. Approach

The badge detection was one of the main parts of making a reliable sentry. We decided to use the Cornell ID cards as our badge, as most people on campus will have their badge with them, and this method would be easily adaptable to any other school or organization that requires a badge. Our main environment that we tested was a hallway, as sentries are generally located in enclosed environments.

We tried four main approaches to determine how to properly track our features. We originally used a color-based detection, but this did not translate well when moving to a more sophisticated badge. We then moved onto an optical flow algorithm, which was very good at tracking an ID moving between frames, but was not well-suited to determining if an ID was actually in-frame based on a pre-loaded image of an ID. The next method ended up being our final method, with was using SURF to detect features in the images, and perform nearest neighbor on the feature vectors. The third approach was to use an edge-detection filter (cvCanny) on both images, and then apply SURF and nearest neighbors on the results. This ended up being much worse than both optical flow and our final method. While the edge detection could locate the badge, it frequently had incorrect pairs.

We took an image of a Cornell ID against a feature-less background as our base image. Our method for badge-detection was derived from find_object, an example from the OpenCV library, which consists of the following steps. First, take in a new image, which we called the object image, from the front camera. Next, run the SURF algorithm on the object image. Lastly, we used a nearest-neighbor search to detect similar features in the object and base images. We chose to use the built in SURF algorithm in OpenCV, as we read that it offers a significant speed-up over a SIFT-based algorithm, while having similar results. We then used another OpenCV algorithm, called find_pairs, to perform our nearest neighbor search on the two image's feature arrays which were returned from the SURF algorithm. As seen in the image below, there are many common features between the base and object images, which are represented by a white line connecting the two features. There is some error, as you can see that some features in the base image match up with wrong features in the object image.

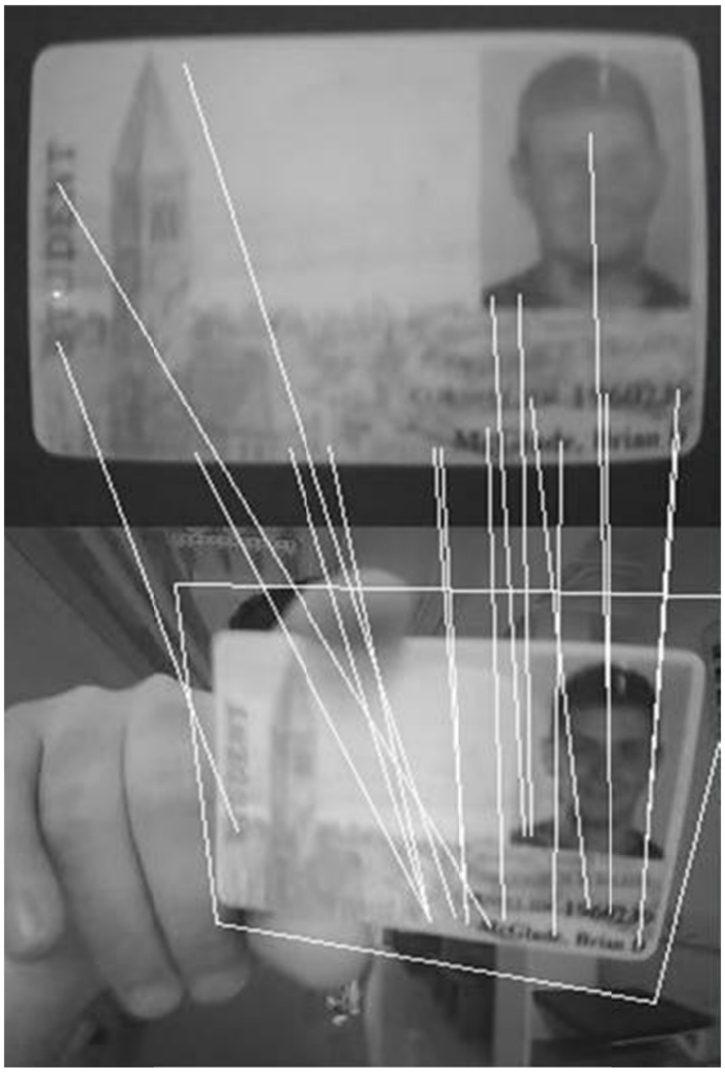


Figure 1: Badge Detection Algorithm Output

We needed to teach the PSD what a valid Cornell ID looked like, so we choose to employ a logistic regression. We gathered training data, which was analyzed to find how many similar features the parrot found between the base image and each training image.

Using this data, along with a binary value where 1 represented a positive image and a 0 represented a negative image, we solved the over-determined system to get initial guesses for my coefficients in the regression equation:

$$g(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

From there, we perform a few steps of Gauss-Newton iteration in order to refine our constants. This regression equation is then used on any live image data taken from the parrot. If g(x) is larger than .5, then the badge identifier function returns a value 1 for the variable badge_present, and returns 0 otherwise. Below we have included our regression model, where the red asterisks indicate training data points.

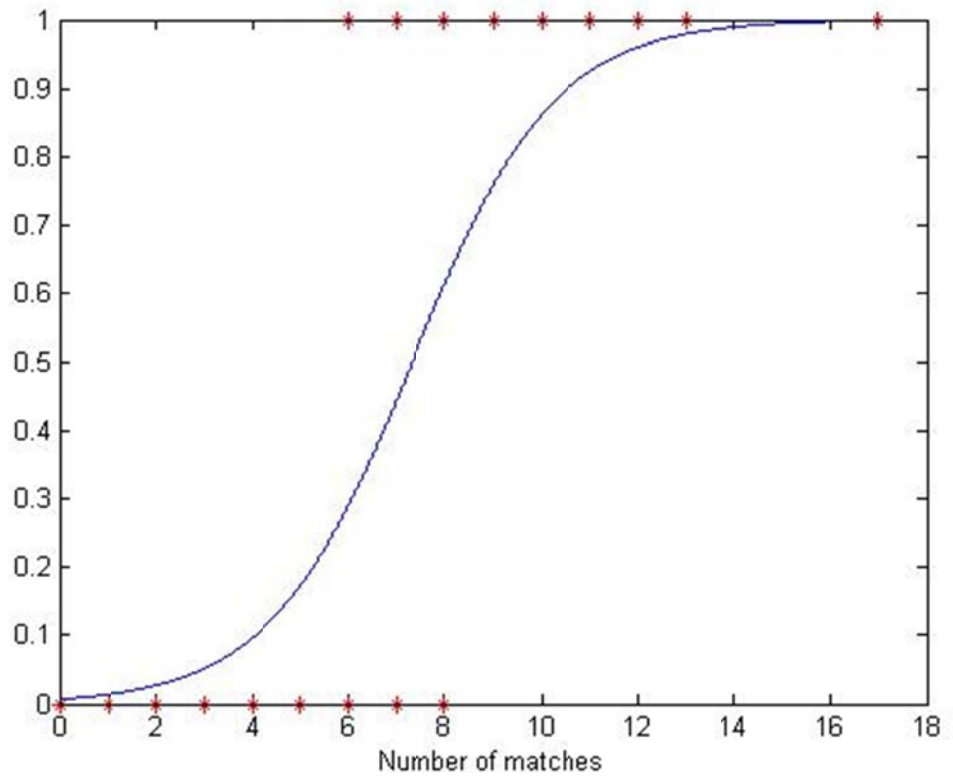


Figure 2: Regression Model Developed using Training Data

Once we determined that the badge was presented in front of the AR Drone, we had to give it commands to move from the sentry point. The robot was given commands to move to the right in order to clear a path for the person to pass. This was done by setting the roll parameter of the AR Drone to 500. When a move step is performed, the robot is also rotated to the left so that it faces the sentry point as it moves and stays within a smaller radius of the sentry point. The number of times these two actions take place is stored in a variable called movecount to maintain a record of how far the robot has moved. This move command is repeatedly sent to the robot (and movecount is incremented) for every new image until the badge is no longer detected by the camera and the badge locator function returns a value of 0 for the variable, badgepresent. After that point the robot is sent move back commands as many times as it moved, using the movecount variable, so it comes back near its original spot. The move back procedure entails setting the roll to -500, so that it moves back to the left and giving it a negative yaw value so that it rotates back to the right and hopefully faces the same direction it started out facing. When movecount is decremented all the way to zero, no new commands are given to the AR Drone, since it should have returned to the sentry point. If the badge reappears the AR Drone is once again moved, according to the move procedure described above. This process is illustrated by our state diagram below:

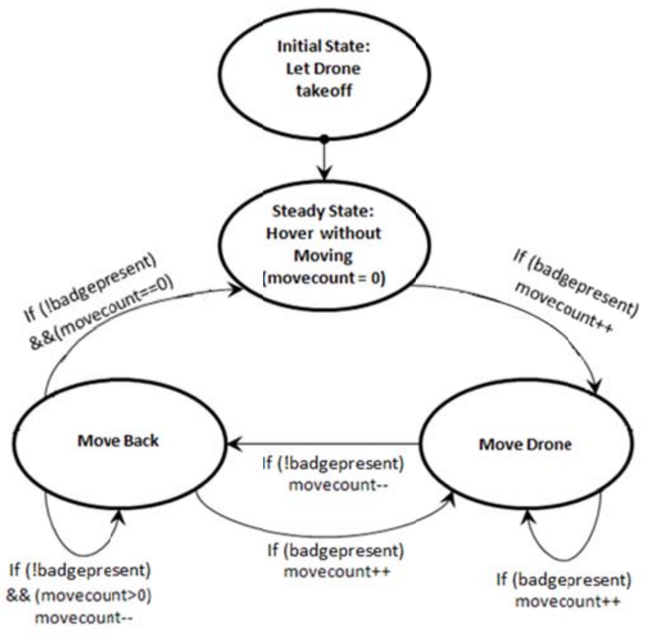

Initial State: Let Drone takeoff
Steady State: Hover without Moving (movecount = 0)
If (!badgepresent) &&(movecount==0)
If (badgepresent) movecount++
Move Back
Move Drone
If (!badgepresent) movecount--
If (badgepresent) movecount++
If (!badgepresent) && (movecount>0) movecount--
If (badgepresent) movecount++

Figure 3: State Diagram

Even when no movement commands were sent to the AR Drone, it would not remain steady over a fixed point due to the numerous variables present during its hovering process. In order to limit the distance the robot drifted from the sentry point and to correct imperfections in the Drone's move back stage, we used the bottom camera of the drone to attempt to locate the center of the sentry point and moves towards it. This was done by marking the sentry point with a distinctive orange colored piece of paper (taped to a weight so it wouldn't fly away). In order to isolate the marker, we examined each picture received from the bottom camera and ran it through the function, cvInRangeS, which is part of the openCV library. This function take one image as an input and returns a new image with white pixel for every pixel in the original image with RGB values in a range we give it and a black pixel for every pixel in the original image out of that range. With some parameter tuning by hand, using sliders for the high and low values for red, green and blue, we were able to determine a range that isolates the color of our orange marker fairly reliably. This range is used to produce our threshold image where white pixels represent the marker. This new image is run through a median filter to smooth the image and remove isolated pixels. An example of the isolation of the marker from an image taken from the AR Drone's front camera (larger camera used for clarity) is shown below:

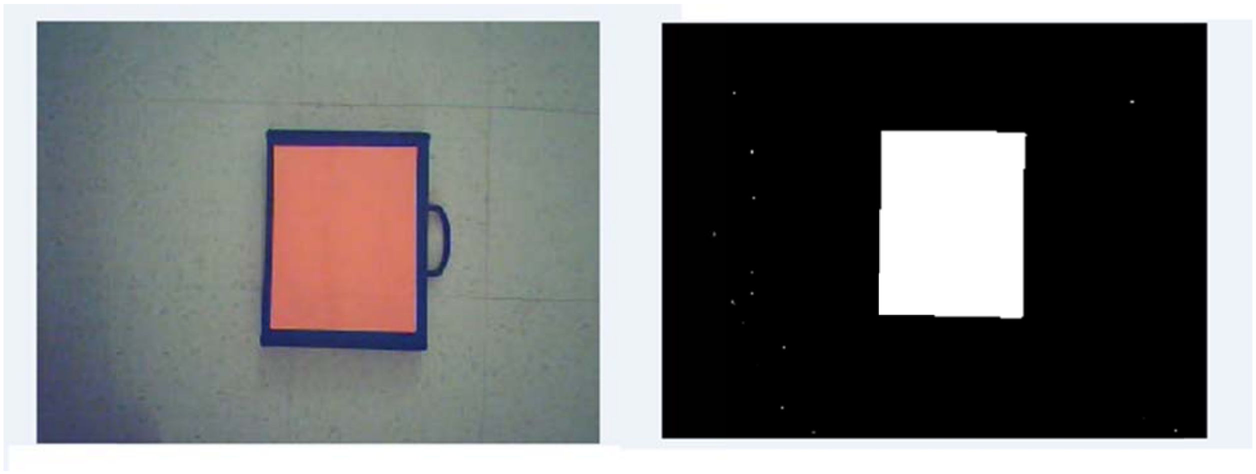


Figure 4: Image from AR Drone and Isolated Marker

After we had identified the pixels corresponding to the marker, we determined the center of the marker on the image by averaging the x and y values of each white pixel. We also kept track of the relative size of the marker by storing the number of white pixels. In order to line up the AR Drone with the center of the marker we determined how far the center of the image (for an 88 x 72 image is: (x,y) = (44,36)) was from the center of the detected marker pixels. Small move commands were then sent to the AR Drone in the direction that the center of the marker differed from the center of the image in order to reduce this distance. These commands were not sent if the badgepresent variable was asserted or the movecount variable so that this wouldn't interfere with that process. Also, if the size of bottom isolated image was too small (less than 20 pixels) these commands were not sent since these pixels could likely be noise and the marker may not have been present on the bottom camera at all.

IV. Experiments

A. Data

Our main experiment that we collected data from was tracking the features found in both the base and object images. For our training data, we took 150 negative images, where we either had nothing, a person, or a non-Cornell ID in the frame. We also took 100 positive images, which had a Cornell ID displayed prominently in the frame. Most of the analysis we did on feature tracking was from offline testing, after we were satisfied with those results, we verified the results from online testing. We then gathered 100 test images, with an even split between images with a Cornell ID in them and those without the ID. After applying our regression model to the test images, the following results were observed.

	Images identified as positive	Images identified as negative
50 images with a Cornell ID	32	18
50 images without a Cornell ID	1	49

Table 1: Results from Test Images sent through Regression Model

The correct identifications are the ideal scenario; however for our application, it is better for our sentry to not move from its location and when a person presents a badge than to have the sentry retreat when no badge or an incorrect badge is presented.

B. Accuracies

As described above, while badge detection algorithm did correlate reasonably well with presence of the badge, there were cases where inaccuracies occurred. We had some cases were the badge was present in the image, but did not correlate enough features to be recognized. We also had a couple cases where the badge was not present in the image, but there were enough incorrect features matching so that the algorithm returned that the badge was present.

Other than effectiveness of our algorithm, one source of inaccuracy was the limited resolution of the camera that prevented features from being detected when the badge was too far from the camera. This meant that we had to hold the badge right in front of the camera for it to be detected. We also noticed that an important requirement for the algorithm to work was that our base image had to be taken in similar lighting to where the sentry would be operating. This limits some of the extensibility. We believe a significant contributor to his effect is the Cornell ID having a holographic image that looks different depending on the light conditions. This holographic nature of the badge also causes it to change appearance depending on the angle at which it was held.

C. Robot Experiments

We conducted experiments to test the effectiveness of our algorithms by setting up the parrot over the sentry point marker in a hallway and walking towards it with a badge to see if it would move out of the way. We got mixed results with the online tests of the AR Drone, largely due to its drifting nature. Our motion state machine was largely dependent on position and because of the constant motion, it was difficult to predict or keep track of where the robot's location. The algorithm using the bottom camera to keep the robot in position was beneficial when the robot was hovering over the fixed point but it could not help when the robot was avoiding the oncoming person. Even when the robot was near the marker, the viewing angle of the bottom camera was so narrow that the robot would have to be really close to the marker to begin with for the process to work. This could have been remedied using a larger marker than the paper size marker we used in our experiments. Despite the difficulties, we were still able to detect badges and move the AR Drone out of the path of an oncoming person presenting it to the robot. Pictures of our robot experiments are shown below.

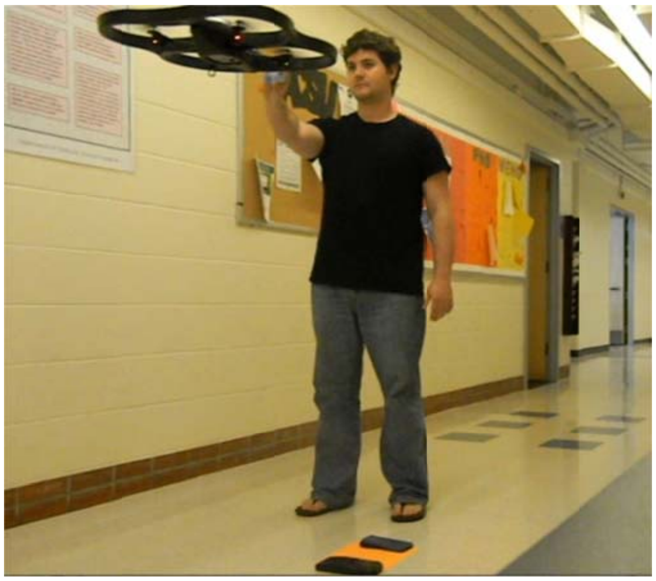
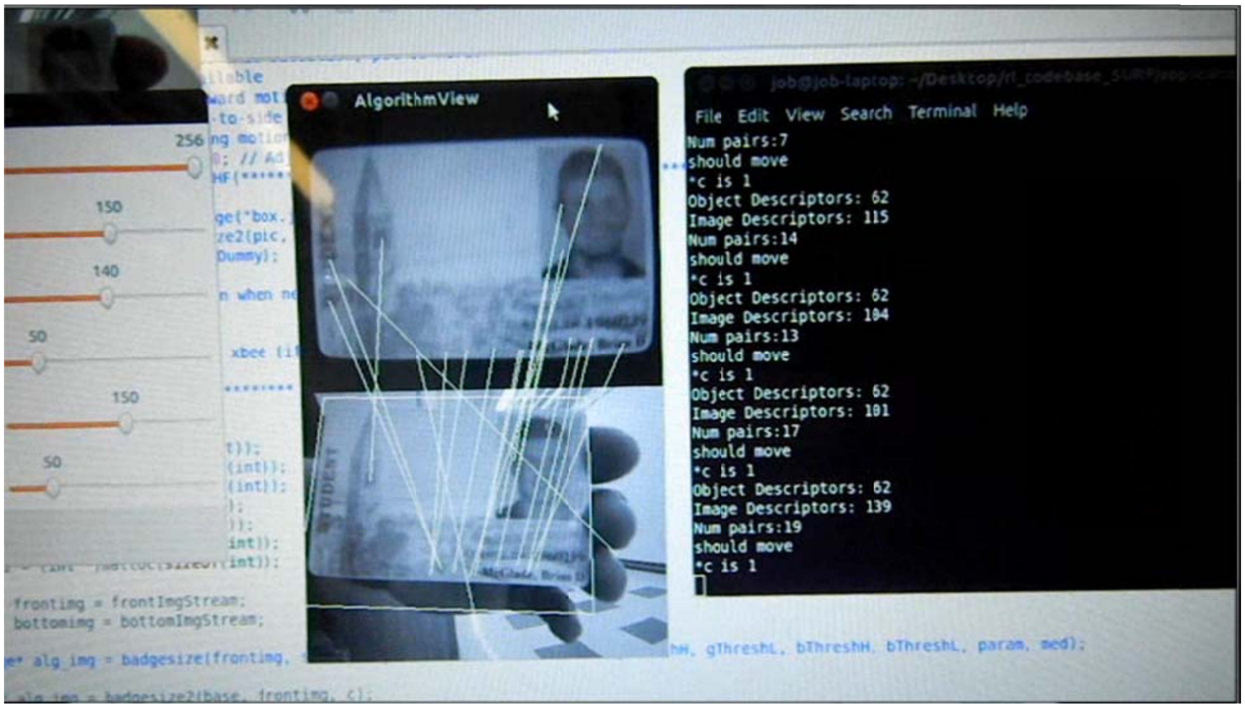


Figure 5: Robot Experiment



```
Num pairs:7
should move
*c is 1
Object Descriptors: 62
Image Descriptors: 115
Num pairs:14
should move
*c is 1
Object Descriptors: 62
Image Descriptors: 104
Num pairs:13
should move
*c is 1
Object Descriptors: 62
Image Descriptors: 101
Num pairs:17
should move
*c is 1
Object Descriptors: 62
Image Descriptors: 139
Num pairs:19
should move
*c is 1
```

Figure 6: Screenshot of Algorithm Running

VI. References and Acknowledgments

We used the AR Drone API website and the OpenCV wiki extensively during our project. Our badge detection algorithm was based on and partially built off the example file, obj_detect.cpp, found in the samples folder of the OpenCvV2.2 library, and authored by Liu Liu. We'd like to thank Cooper Bills and Tung Leung, our TAs for this class, for all their help and guidance throughout the completion of this project.