CS 4758 Final Report
Autonomous Object Search with the AR.Drone Quadrotor

Kritarth Jain, ECE '11 <kj83>
Hyundo Reiner, ECE '11 <hpr6>

15 May, 2011

***Abstract*– Our project goal is to have the AR.Drone perform an autonomous object search in a known environment and return the location of the object being searched to the user. The environment is represented as an undirected, connected graph and mapped out *a priori*. Path planning depends on the history of where objects have been found in the past. The optimal path calculated assumes that the previous locations found by the robot are "hot-spots" for the object in question, and adapts to search around these sooner than other locations.**

1 Introduction

We used the AR.Drone to implement an autonomous object search in a rectangular room. Our basis for localization was a vision-based marker detection scheme with several discrete states, involving a color filter to locate rectangles of color paper on the walls in different configurations. We attempted to navigate around the room by traversing the network created by the edges between these discrete states, which involves only moving towards different markers in the drone's field of view. A probabilistic map of the room was initiated at uniform probability, and maintained after each trial run. The algorithm for searching the room was a breadth first search of the network, where the root was the drone's current position, and searching for the path that will maximize reducing the drone's unsearched area. Because of our approach to the problem, it is only possible to to fully locate an item in a room if it is found on two different edges, though finding it on one edge still helps the robot in the future by updating the probability along that edge.

2 Related Work

We did not use any existing implementation of localization, navigation, or path planning in our design. We used the AR.Drone API to interface with the drone, and put most of our work in Cooper Bills' Linux SDK `planner.cpp` file that was included in the original package. We opted not to use much of the existing code and specific implementation that we had expected to use when we first proposed our project.

Because we didn't want to put all our effort into localization, as it was not the goal of our proposed project, we chose to use OpenCV's free libraries to assist us in this task. We also considered using ARToolKit(Plus) for marker detection, but decided to go with OpenCV since we were already somewhat familiar with using it from the course. Similarly, we did not make object recognition robust because it was not our focus, though we did experiment a lot with different feature detection algorithms.

3 Approach

Due to the scope of the problem, it was necessary to break the problem into several independent steps that are essentially just states in the overall state machine that governs our program flow, as illustrated in Fig. 1. We made several modifications to our approach to make the end goal more attainable. Among these include altering the state space of the robot to include only a small number of states that are placed in strategic positions around the room, represented by a graph, while maintaining a search space of much higher resolution by representing this as a matrix and defining a mapping between the two. The main stages of the problem were: predetermination of the environment, cheap localization, navigation within the environment, and path planning based on an end goal of finding a known object.

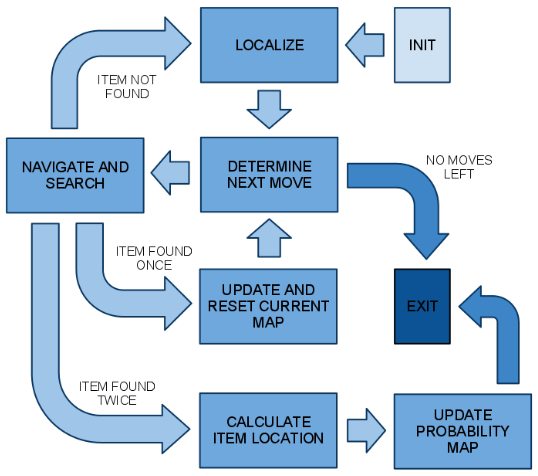


Figure 1: Overall program execution flow.

3.1 Pre-determination of the environment

Our representation of the environment is in some ways favorable because it allows for a high resolution of search space while maintaining a low state space for the drone. We were able to do this by representing the state space of the drone as an undirected, connected graph, where edges in the graph map to a much more continuous matrix space that represents locations in the room. Doing this made navigation much simpler because decisions are only between a few discrete choices at any point in time, and localization also easier because we define our state space size to be smaller and there are less choices that the drone must choose from.

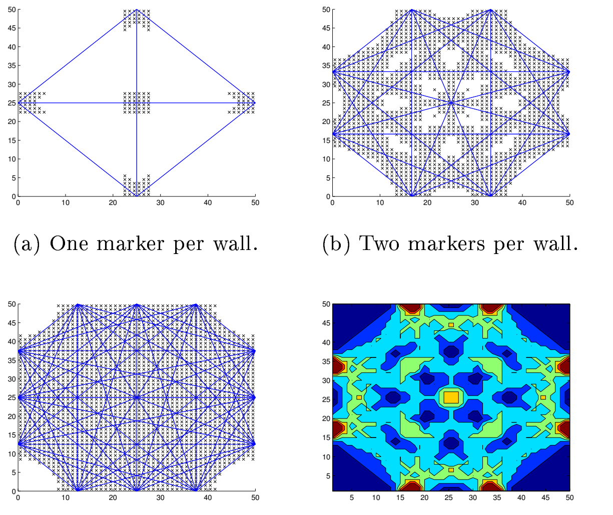
The relationship between number of markers (i.e. nodes) and number of edges between them is given by

$$E = \frac{3}{4}N^2$$

where E is the number of edges and N is the number of nodes. The factor of $\frac{3}{4}$ is there because we disallow traveling between states on the same wall. Clearly, having more markers increases the number of possible actions we have at each node, and increases coverage of the search space up to a certain point, while not costing as much in terms of number of markers we have to keep track of or the allowable spatial resolution between them without losing accuracy.

Because the network is so dense, we chose to precalculate the representation in MATLAB as three separate files: the initial probability map, which is has all the locations that are reachable set to 1, the adjacency list paired, which we found to be the most convenient way to store a graph, and the mapping from edge to points in our map, also stored in a matrix. We defined points that are mapped to as being within a certain distance of the edge that is defined by the two vertices at either end. This distance is calculated empirically by checking what radius location the drone reliably sees when hovering at a fixed height, then relating that quantity to the dimensions of the room. Thus, when the drone moves from one location to another, the drone actually covers a wide array of points along the path that it traveled, giving a better approximation of the room's continuous state space.

To see how effective or detrimental of a strategy this really is, we divided the room into a 50×50 space and calculating the graph with 1, 2, and 3 nodes per wall, equally spaced but not at the corners, then calculating the space covered when the distance threshold from the edge is 5 percent of the width or height of the room, in this case 2.5 (for a typical classroom like the ones we were using for testing, this is reasonably about 1m). Space covered was taken to be space that is traversed by at least two edges, since with only one edge we won't be able to localize the object. As shown in Fig. 2 having only one marker per wall gives 6.24 percent coverage, while two gives 67.36 percent, and three gives 90.72 percent.



(a) One marker per wall. (b) Two markers per wall.

(c) Three markers per wall. (d) Contour plot of levels of coverage, with 50×50 map states and two markers per wall.

Figure 2: Demonstration of coverage by at least two edges while altering the number of edges per wall in Fig. (2a) to (2c). In Fig. (2d), a contour plot demonstrates the distribution of the levels of coverage for a specific case.

3.2 Localization

For localization, we use the drone's front facing camera as the main sensor. Due to the limited resolution of the camera, we decided to use a color blob filtering method to identify regions of interest along the boundaries of the room, and localize based on these points. For the detection of specific colors, we use the OpenCV function `inRangeS` which takes in as argument the colored image and the region of color we want to detect, and outputs a grayscale image with a boundary enclosing the color of interest with the center of that region. After testing with RGB, we decided to use an HSV color scheme, because we found it easier to demarcate colors based on the Hue region they cover over their different intensities. Then the value and the saturation fields determine the brightness or the dullness of that color. We implemented a function which gave us the exact hue regions over a given color wheel, which was very helpful in determining the "regions of interest" as shown in Fig. 6.

We used specific color squares arranged in fixed patterns as markers to be placed along the boundary of the room, as in Fig. 3 and 5. The distances between these were determined based on the size of the room and the angle of vision of the drone. Every marker consists of two different colored rectangles placed at diagonal to each other. Having two distinct colored rectangles allowed us to determine a marker uniquely. For each marker, we stored the center point of both the colored rectangles as well the color of the rectangles.

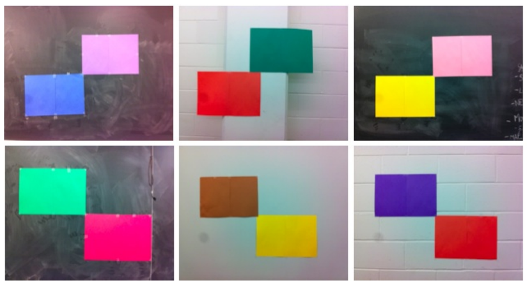
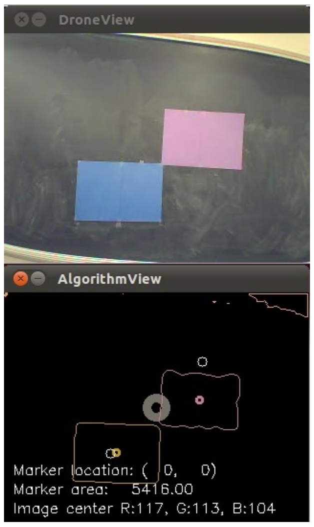


Figure 3: Some of the markers we used.

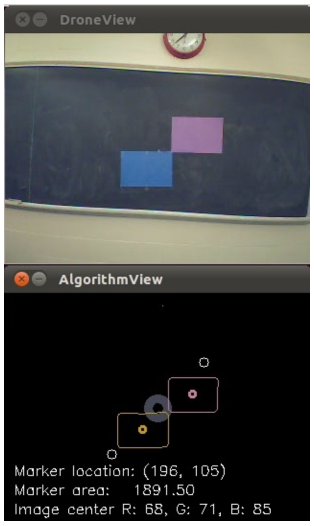
For performing localization, we use our HSV color filter to filter out both the regions corresponding to the two colors in a state and use these to determine the relative location of the two rectangles. The location of these rectangles in the frame of vision of the drone also tells the drone what angle and relative distance it is looking at the marker in, allowing the drone to correct his position with respect to to the marker using a proportionality controller for movement in the six different degrees of freedom to correct the error in the position.

When the drone has to determine what current state it is in, our algorithm runs the color filter for all the color pairs in our system as stored in an array, and if both these colors have a blob on the current frame, then we check among all the markers if these two colors represent that marker, as shown in Fig. 4. If it does, then we have successfully determined the marker.



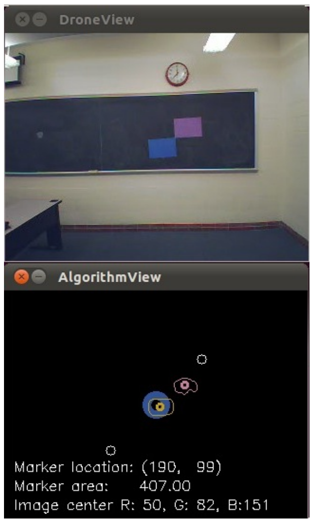
```
Marker location: (  0,   0)
Marker area:    5416.00
Image center R:117, G:113, B:104
```

(a) 2m distance.



```
Marker location: (196, 105)
Marker area:    1891.50
Image center R: 68, G: 71, B: 85
```

(b) 5m distance.



```
Marker location: (190,  99)
Marker area:    407.00
Image center R: 50, G: 82, B:151
```

(c) 7m distance.

Figure 4: Algorithm views for color filtering.

To define the whole localization space, we use the concept of a connected graph with all the nodes lying on the periphery and each node representing the location of a marker in the physical space. By joining these markers, we determine the paths we can move on in our locale. Thus here, the actual localization is performed with the drone starting out by determining his current state using the color blob method discussed above. Using the path planning algorithm as discussed later, we decide which is the next node we should go to. Here, first we perform a relaxed tracking and movement to the new node. Given the node, we use the color filter with relaxed constraints to find the marker of this node by rotating or yawing around its current position at its current node. If the drone identifies the next state within a certain measure, the drone proceeds to move in the direction of this next node by centralizing the location of both the color blobs of the marker which it sees. After reaching within a certain distance of the new node as determined by the distance between the center points of the rectangles of the marker, the drone proceeds to perform fine-tuned localization, in order to have a better localized position at this node. During initialization, the drone locates the nearest marker in its field of view as fast as possible and centers on that point. If no markers are in view, it rotates to the right until it sees one, though we always started the drone directly in front of a marker for convenience.

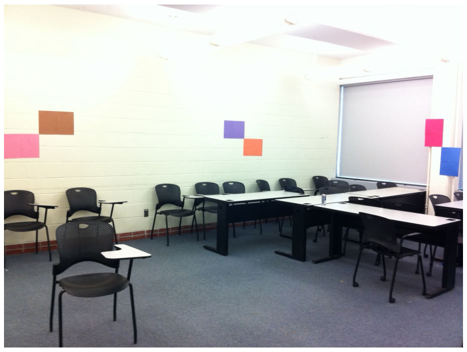
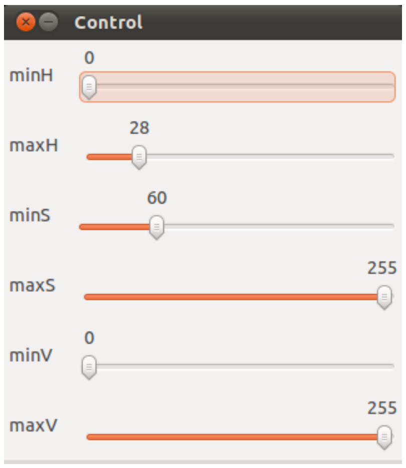


Figure 5: Our setup for the AR.Drone experiments.

Figure 6: GUI used for determining accurately HSV values in real time.

(a) Slider bars.


Control

minH 0

maxH 28

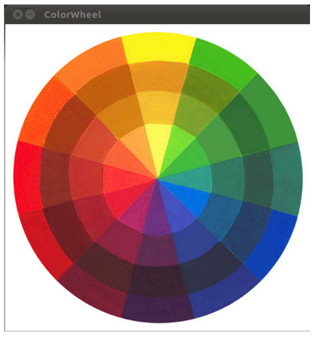
minS 60

maxS 255

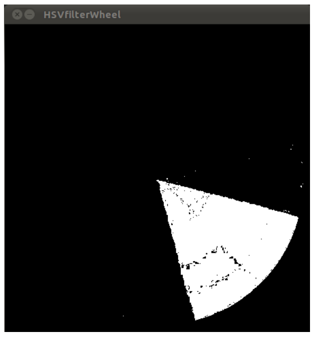
minV 0

maxV 255

(b) Reference image.



(c) Filtered image with set HSV values.



3.3 Navigation

Our navigation algorithm is simple, and described in Alg. 1. It completely and easily allows for navigation along the network we defined. Once the desired marker is found, the drone only needs to keep the marker centralized in its view as it moves forward, until the volume of the marker reaches a certain level. We implemented a simple proportional controller which we thought would have been adequate enough, but we found that the AR.Drone does not respond smoothly to control inputs. Because the cameras are so noisy, especially with auto contrast, using volume as a metric has a lot of room for error. We also tried using distance between the marker centers as the measure for stability, in that when the line segment connecting the two colors stops changing in the robot's field of view, then we have reached a stable configuration. This also didn't work as well as we hoped, and the AR.Drone would sometimes be able to centralize itself for a brief moment, but often times would drift off and lose sight of the marker.

3.4 Path planning

Our path planning was solely defined by the current values of the probabilities that lie on the edges that propagate from the current node which the drone is located. The basic algorithm is outlined in Alg. 2. We instruct the drone to traverse the graph greedily, choosing the edge with the largest probability sum (essentially, covering the most unexplored area possible and maximizing chances of finding the item from

Algorithm 1 Navigation algorithm. Negative feedback from the front camera keeps the drone centered as it flies forward.

```
for (each viewable marker M) {
    identify which marker M is;
    calculate M's score if traversed (see Alg. 2);
}
rotate to desired marker;
keep marker in center of view;
fly forward until volume > VOLUME_THRESH;
```

the current node) at every step. Because the graph is highly connected, we thought that using a greedy algorithm wouldn't have that many detrimental effects because the drone has a large number of choices at each node. The structure is very similar to creating a graph with probability weightings on its edges, but the difference here is that ours directly maps to the search space while using just a graph structure would require some further calculation to translate to location.

When transitioning between states, the drone is scanning the ground for any signs of the object along the edge that it is traveling on. After an edge has been traversed, if the object was not found, then all the probability values that were just searched, which includes every point that is linked to from the edge, is set to zero for the remainder of the search iteration. Note that this is not a permanent update; it works to guide the drone to unsearched areas after the "hotspots" have been exhausted. If the entire search space has been zeroed or the sum of the remaining scores does not change after a certain threshold of turns, then we declare the item unfound, and the drone exits its search.

However, if the item *is* found, then the drone *continues* searching. This is one trade off that results from defining the search space and the drone's space so that they are separate. If the drone can find the object along an edge, then it attempts to localize the object on that line segment by finding object on another line segment. Thus, after the drone has first found the object, then it reinitializes its probability map, but only with values from the original map if the point in the map is within a certain physical distance from the edge just traversed. In addition, the edge just traversed is removed from the graph, to make sure that the new edge (if any) will be unique. Thus, during the second pass search, the map only has value in area that was covered by the edge in which the item was just found, though the edge itself is removed.

Lastly, once the search phase resolves, we update our knowledge based on the outcomes of the trial. There are three possibilities: the object was found on two edges, the object was found on one but not in the second pass, and the object was not found at all. If the object was found on two edges, then we update the probability of our initial starting map to have a higher score on all points that intersect the two lines. If the object was only found on one edge, then we update the probabilities on that edge to be *slightly* higher. Finally, if the object was not found at all, we don't update the probabilities at all.

Algorithm 2 Path planning algorithm, performed until the object is found or the algorithm exits.

```
while(item not found) {
    if (sum of map values is < EXITTHRES || num
        turns which sum is same > EXIT2THRES) {
        exit with "not found";
    } else {
    for (each outgoing edge E) {
        sum the total map matrix values along E;
    }
    if (max(E) = 0) {
        navigate along a random edge;
        delete that edge from the graph after
            traversal;
    } else {
        navigate along max(E);
        zero map values at points of max(E);
    }
}
if (total num times item found == 1) {
    set map values to non-zero starting values
        only along last found edge;
    repeat;
} else {
    calculate item location;
    increase points at location in stored map;
    exit with "found";
}
```

3.5 Failures

Our original idea for this project was to implement localization, navigation, and object detection in a populated environment without any constraints. However, we quickly realized that performing all the tasks without existing implementation or major simplification would be infeasible. We initially hoped to use existing implementations to aid our project, such as Cooper's implementation for object avoidance using the XBee chip and extra sonar sensors, and Tung Sing Leung's code for localization using Bundler, which is a program that tries to generate a sparse point cloud from a collection of images. We decided that having the extra implementation of real-time object avoidance would not be applicable to our end goal nor increase our performance by a substantial amount, and thus not worth the effort of integration. We did not get a chance to try Tung's localization implementation because he is no longer a TA for this class.

A big time sink that we put a lot of time and effort into without yielding any usable results was our attempt to detect arbitrary objects using SURF feature detection. We tried several methods to get this to work for our application, since if it did, we would be able to use it for both localization and robust object detection. Some methods which didn't work include using using several training image sets for a single object, averaging and thresholding values passed through SURF to reduce noise and false positive rates, even with multiple cameras. The main problem was that the resolution on either drone camera is too poor to effectively use SURF as it stands for practical applications. We found that SURF could detect an object if the drone was within at most a meter from the object, and scales with resolution, which means this range is much less for the drone bottom camera. We didn't see it worthwhile to do an error analysis of this algorithm because there would be no way that we could use it in our final design, and neither of us had a background strong enough in computer vision to give us a favorable probability of succeeding with it, so we moved on to color based filtering techniques.

4 Results

4.1 HSV Detection of Markers

All our markers consisted of two 16" by 12" colored rectangles in certain flag orientations. Using the color detection algorithm as discussed in the localization section, we are able to determine within a fair estimate of distance and lighting the complete description of the markers in our environment. We have the observations displayed in Tab. 1 on performing these color tests with the static AR.Drone camera. Note that the lighting was determined based on the location of the marker in the room and is difficult to quantify.

Marker	Color1	Color2	2m	5m	7m	10m
0	Orange	Green	Both	Both	C2	C2
1	Pink	Brown	Both	C2	C2	None
2	Purple	Orange	Both	C1	C1	None
3	Red	Blue	Both	Both	C2	C2
4	Brown	Yellow	Both	C1	C1	None
5	Green	Red	Both	Both	C1	C1
6	Blue	Purple	Both	Both	Both	C1
7	Yellow	Pink	Both	C1	None	None
Success rate:	(single color)		100%	75%	50%	25%
Success rate:	(both colors)		100%	50%	12.5%	0%

Table 1: Marker identification for different color combinations.

4.2 HSV Color Determination

Using the self created HSV color determiner shown in Fig. 6, we obtained the ranges shown in Tab. 2 for the colors we use in OpenCV standard.

4.3 Navigation

We performed initial testing and experimentation for precision control of the AR.Drone, within reasonable interests of our implementation. We found that of 10 trials where the drone was commanded to hover in place, 9 times out of 10 the drone would drift in an arbitrary direction, and 7 or more of these would be significant and detrimental to our localization capability. Furthermore, the movement control supplied is extremely difficult to standardize; even after lengthy testing of controlling the drone in a desired way, the drone would not behave as desired in many cases.

Color	H	S	V
Blue	0-37	60-255	140-255
Green	25-87	80-255	120-255
Yellow	90-100	60-255	200-255
Brown	100-113	160-255	0-255
Orange	110-116	60-255	60-255
Red	117-126	111-255	0-255
Pink	117-126	60-255	150-255
Purple	135-180	40-255	140-255

Table 2: HSV color determination results.

4.4 Further Results

Unfortunately, we were unable to obtain experimental results for the rest of our implementation. Nearly all of it has already been coded out and compiled, but due to the instabilities and inaccuracies in the AR.Drone control we were unable to get navigation up to a suitable level to have meaningful results. In light of this, we attempted to at least simulate this in MATLAB, and this is still a work in progress.

5 Conclusion & Future Work

Given that the AR.Drone is a relatively cheap robot with limited sensors, we set out to achieve solving a problem as difficult as localization. We were able to lay out algorithms which we believe, given the required time and patience with the AR.Drone, can lead to successful and promising results. We achieved success in the area of localization using color markers, and we have laid out the guidelines for an implementation for autonomous object search using computer vision. We believe that given better sensors, or a more robust robot, the shortcomings of the AR.Drone can be countered. Having said this, we have found that the AR.Drone is a fun and educational research tool.

There would be many ways that we could improve the performance of our drone in our project. For one, the biggest bottleneck we felt was localization, and a large part of why that was so was because the camera quality of the drone's isn't good enough to be able to perform the more robust, higher order search functions that we initially implemented such as SURF. A different robot with a better camera would be much less constrained by the physical construction and the hardware. Since localization is basically necessary to perform all other functions in our proposal, poor performance in this area brought down overall performance. Similarly, since we used a vision based scheme for navigation and flight stabilization as well, improving the camera or the drone would definitely help in that respect.

Other improvements could involve optimization and making algorithms more in depth. A good example of this is that our current algorithm for searching a graph is very greedy and not very optimized. Another example is using a Kalman filter to stabilize noisy measurements. We recognize that there is still a lot of work that could be done with our project. The entire project incorporates a vast number of tasks and topics, each of which need more time than a single semester to produce a top quality result.

6 References

1. Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, "SURF: Speeded Up Robust Features", Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346–359, 2008. <ftp://ftp.vision.ee.ethz.ch/publications/articles/eth_biwi_00517.pdf>.

2. Stephen Poskorski, Nicolas Brulez, "AR.Drone Developer Guide" SDK 1.6, February 24, 2011.

3. "Bundler: Structure from Motion (SfM) for Unordered Image Collections". <http://phototour.cs.washington.edu/bundler/>.

4. "ARToolKit". <http://www.hitl.washington.edu/artoolkit/>.

5. "Handheld Augmented Reality". Christian Doppler Laboratory. <http://studierstube.icg.tugraz.at/handheld_ar/artoolkitplus.php>