# Thirsty Rovio – Autonomous Mini-Keg Locating Robot

Jeff Melville and Tim Sams

*Abstract*— **Vision and planning are two critical and evolving fields in the field of robotics. This project seeks to implement planning, vision, and classification algorithms in the context of locating a Heineken mini-keg with an unknown environment. The algorithms will be implented on the WowWee Rovio commerical robot platform. The environment was restricted to spaces such as hallways where the predominant direction of motion is apparent. The types and arrangement of obstacles remains unknown and must be avoided as the robot navigates in search of the mini-keg.**

## I. INTRODUCTION

Our project, entitled Thirsty Rovio, was to use the Wowwee Rovio webcam platform to move through a hallway environment and identify a Heineken brand mini-keg, shown in Fig. 1. Simple to use home robotics platforms are increasingly popular for many of these types of projects because they are inexpensive and relatively simple to use. While our time with the Rovio was not without despair, it proved to be an able platform for our image algorithms in order to identify said mini-keg. Our methods were such that we were not limited too much by the movement capabilities of the robot, but more so by the camera and the images we were able to acquire. All of our software algorithms were written in the C# language using .NET wrappers that we found available as open source libraries [4], [3]. All of our vision algorithms were implemented using OpenCV libraries from [2]. In order to find the keg we implemented both motion and vision algorithms. This report will detail the specific nature of our implemented algorithms and the results we were able to achieve with these installations.

## II. PLATFORM

### A. Hardware

In order to understand the operation of our algorithms and software, its important to have a good introductory grasp of the WowWee Rovio platform. The Rovio robot (Fig. 2) is a webcam equipped robot with 3 wheels that is capable of movement in all directions on any floor. We found, however, that the motion was much more precise when used on carpet versus a smoother floor like tile, but will comment more on this later. The connection to the robot is through a Wi-Fi area network. You can configure the robot to create its own ad-hoc network or you can connect it to an existing network in your location. We found that it would respond better when using the private ad-hoc version because there was relatively no other traffic on that network compared to a public network like on the Cornell campus. You can control the robot in two different ways: through the web interface provided or through the API by sending HTML based commands. The web interface is somewhat limited and provides very little in terms of actual custom programming. That being said, the web interface is very useful in the event that your custom programming becomes frozen or when you need to change several of the settings on the robot itself. We accessed the Rovios instruction set programmatically by sending specifically formatted HTML addresses as described in [5]. For our purposes we found it easier to use a modified version of an existing C# library, as discussed below.


Fig. 1. Heineken mini-keg

### B. Software

This project utilized two software libraries to accomplish its goal. The first library was RovioLib [3], an C# implementation of the published Rovio control API. This library enabled control of the Rovio and access to sensor and webcam data. Some desired functionality was not implemented in the current version of RovioLib

Fig. 2. WowWee Rovio

and was added. In particular, the camera, IR sensor, and localization functions were added to RovioLib for this project. The extra functionality will be contributed back to the RovioLib project. RovioWrap [4], which is written in VB.NET, was also used briefly before switching to RovioLib for language consistency. RovioWrap source code assisted in adding functionality to RovioLib.

The second major library used in this project was Emgu CV [2]. Emgu CV is a .NET wrapper for Open CV, Intel's open source computer vision library. In addition to exposing the standard Open CV functionality, Emgu CV also encapsulates Open CV into managed classes for a more object oriented environment.

## III. APPROACH

### A. Navigation

In Fig. 3, you can see a graphical representation of our algorithm for the Rovios motion. This all runs in our main program loop that updates for each frame of image data from the camera. First we look at the newest frame to see if there is a box drawn that indicates a green object. If there is not a box drawn, then we poll the infrared sensor to see if there is an object currently detected. In if there is, we move left or right based on the canny edge detection direction decision. If there is not an object here we continue to move the robot in a forward direction. Now if there is a box drawn we again check to see if there is an object detected. If there is an object detected here, we move on to the SVM classifier and if that passes, we have found our Heineken keg. If the object in stage two has not been found, we check to see if the center of the box is located in the center of the image, to the left, or to the right. Depending on the direction, we compensate for the direction offset by moving and then continue the loop back to the beginning. In the even that we reach the SVM decision process and it is not a match,

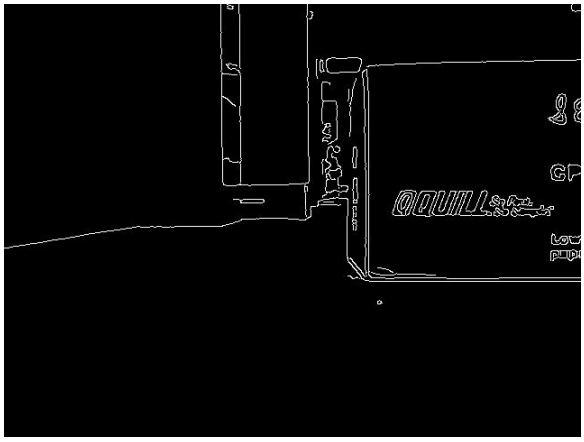

Fig. 4. Canny Edge Detection w/ Obstacle

we avoid the obstacle just like before, using the Canny edge detection method in Section III-B.

### B. Obstacle Avoidance

When the Rovio encounters an obstacle, the Rovio needs to avoid it without losing track of its original trajectory, as described in Section III-A. Avoiding the obstacle includes the decision whether to go around the obstacle on its left side or its right side. The Rovio IR sensor indicates the presence of an obstacle without any information about its size or orientation. A vision algorithm was developed to assist in the "left-or-right" decision. The first step was to perform a Canny edge detection on the current webcam image. Fig. 4 shows the output of the Canny edge detection when the Rovio IR sensor first detected this obstacle, which is a box in the right half of the image.

From this image, several assumptions are made. The first assumption is that an obstacle will always produce an edge where it meets the floor. The second is that this edge will occur in a narrow range of y-coordinates in the image if the image is taken immediately when the IR senses a new obstacle. Third, the texture in the floor is insufficient to show edges with the applied algorithm parameters. Finally, walls will create edges in the extreme x-coordinates of the image.

From these assumptions, it follows that for obstacle avoidance, two regions of interest exist within the image. They each occupy the narrow range of y-coordinates, across x-coordinates except for the left and right sides. The division of the regions is in the middle of the picture, as shown in Fig. 5. It also follows that the region with more white pixels contains the obstacle, and that the Robot should move to the other side.

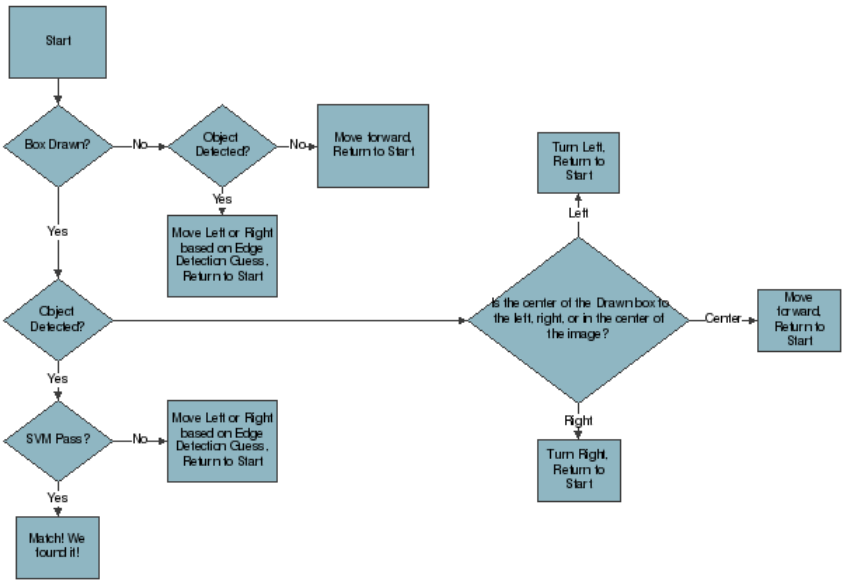


Fig. 3. Navigation Control


Fig. 5. Obstacle Detection Regions

### C. Keg Classification

To accomplish its goal, the Rovio must be able to determine when there is a Heineken mini-keg within the frame. As shown in Fig. 1, a Heineken mini-keg is a distinct color of green. The first stage of the classification algorithm leverages this characteristic. First, the webcam image is Gaussian filtered and converted to a HSV (Hue, Saturation, Value) representation. The HSV representation more directly correlates to whether a pixel is "green" rather than having a "green component." The Hue layer, shown in Fig. 6, is the most important for detecting green areas. The mini-keg is visible in the foreground center of the figure.

The next step is to create a binary image representing whether pixels are green or not. This is accomplished by creating a hue mask. OpenCV represents hue as a value from 0-180. The desired range of green hues is represented from 45 to 85. A mask is also used on the saturation layer to remove white pixels. The resulting binary image is edge and countour detected to determine green areas. The area of interest is defined as the rectangular enclosure of all contours with an area greater than a threshold value, which was experimentally determined. Contours with areas below this threshold are considered noise in the detection, and are not enclosed within the area of interest.The area of interest is shown in the live webcam image with a colored box, as show in Fig. 7.

This first phase will eliminate other brands of mini keg and micellaneous objects because they are not green. However, any object with the selected hues of green will generate a region of interest. To refine the detection, a SVM (Support Vector Machine) classifier was implemented. Instead of operating on each pixel, the SVM can use a histogram as its feature representation [1]. The histogram is advantageous because it reduces the vector space and reduces sensitivity to translation and certain rotation effects. The implemented SVM operates on a RGB histogram of the image cropped to the region of interest. Each dimension of the image space is split into 5 bins, for a vector length of 125. The histogram is normalized to account for the fact that

the region of interest will not always have the same number of pixels. The SVM uses a linear kernel with parameters that are automatically optimized by Open CV. The SVM trained on program startup using the contents of two different folders as its training data, one for each class. The training data consisted of 68 negative images and 66 positive images. Removing the background information before processing reduced the amount of training data necessary.

During operation, the SVM will outline the region of interest in pink for positive predictions (Fig. 8), and in black for negative predictions (Fig. 9).

## IV. RESULTS

As a whole, our keg-finding methods worked like we expected them to. Throughout the process, however, we experienced many issues which definitely slowed us down. First, we experienced many problems when it comes to control of the Rovio. Its almost impossible to have very precise and consistent control of the robot itself. An example of this is when we would try to strafe in either direction. Due to the design of the wheels, the strafing caused the robot to basically make a 90 degree turn in the opposite direction of the strafe because one of the wheels would catch on the floor. We found that this was somewhat minimized when we operated on carpet, but still presented a major problem. In order to overcome this we had to implement many fixes that would adjust when we had to initiate a strafe. One method was to use the navigation signal from the base station which provides an angle theta value as a difference from the perpendicular of the base. Using this, we were mostly able to correct the angle problem.

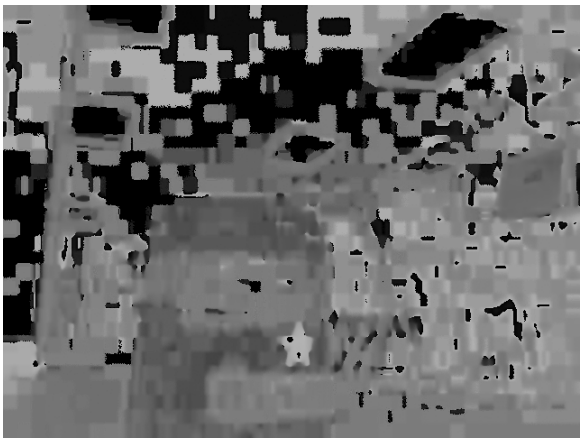

Fig. 6. Visualization of Hue Layer

Fig. 7. Region of Interest


Fig. 8. Positive Classification

Another major hurdle was the fact that our Rovio would hardly charge at all. After doing some research, it appears that this is a very common problem with the platform, and WowWee has not yet provided a fix for this problem. This required us to wait as much as 30 minutes between our test runs and caused our testing to be extremely lengthy.

For classification, the SVM training error is summarized in Table I. The SVM classifier was 80% accurate and was unfortunately more prone to false negatives than false positives. Better accuracy would have been desirable, but in most cases the SVM performed adequately. Much of the negatively classified training data never would have been evaluated by the SVM in the final system because it lacked sufficient green content. In operation, the SVM performance seemed to be sensitive to the lighting in the room of operation. When

Fig. 9. Negative Classification

| Classification | Samples | Errors | Accuracy (%) |
|---|---|---|---|
| Positive | 67 | 11 | 83.6 |
| Negative | 68 | 16 | 76.4 |
| **Total** | **135** | **27** | **80.0** |

TABLE I
SVM TRAINING ERROR

we were working it seemed that the light intensity or amount of light being let into the room could cause false positives and negatives on the identification of the keg. Moving to a different room than where the testing data was taken also impacted results. Despite these few setbacks, the Rovio was able to effectively locate and identify the keg in the majority of our tests.

## V. CONCLUSION

Overall, we were satisfied with the outcome of our project. The project largely fulfilled its original design objectives. The Rovio successfully discriminated between the Heineken keg and other objects the vast majority of the time, especially in the selected environment. The navigation segment worked reasonably well, despite correcting for issues with the Rovio dynamics. We saw the difficulties that can arise from implementing even fairly simple planning algorithms on a real, imperfect platform. However, it was rewarding to see class concepts implemented in a functional and entertaining project. The project also exposed us to practical methods and libraries for utilizing what had previously been mostly abstract mathematical concepts. The main disappointment was the difficulties operating the Rovio, as discussed in Section IV. An interesting follow up would be to implement similar algorithms on a different robotics platform and compare the results. Some aspects of the algorithms could also be improved. For example, the vision based obstacle avoidance algorithm could maintain a weighted memory system of predictions to improve performance avoiding obstacles that take up larger portions of the field of view when the IR sensor triggers. The SVM classifier could be tweaked to improve performance in different environments or light intensities. Some ideas include using a different kernel or a different feature vector such as a higher dimension and/or HSV histogram. Experimenting with SURF feature detection on the mini-keg could be interesting as well.

## VI. ACKNOWLEDGEMENTS

We would like to thank Professor Saxena and the TA's for exposing us to the concepts explored in this project, as well as providing time resources to assist its implementation. The RoboCommunity.com forums were helpful in working around Rovio issues and assuring that we were not alone in our difficultues. The project also would not have been possible without the software libraries mentioned in Section II-B.

## REFERENCES

[1] Olivier Chapelle, Patrick Haffner, and Vladimir Vapnik, *SVMs for histogram-based image classification*, 1999.

[2] Emgu Developers, *Emgu CV: OpenCV in .NET*, `http://www.emgu.com`, 2010, [Online; accessed 24-April-2010].

[3] Serg Podtynnyi, *Rovio API library (.NET)*, `http://roviolib.codeplex.com`, 2010, [Online; accessed 1-May-2010].

[4] Scott Settembre, *RovioWrap*, `http://roviolib.codeplex.com`, 2010, [Online; accessed 15-March-2010].

[5] WowWee, *WowWee rovio API specifications*, 1.2 ed., October 2008, `http://www.wowwee.com/static/support/rovio/manuals/Rovio_API_Specifications_v1.2.pdf`.