Visual SLAM using Rovio

Jeffrey Buente, Rohan Sharma and Daniel Mejía

***Abstract*—Localization and mapping are common requirements in many robotic applications. Mobile robots often rely on sophisticated sensors to obtain information about the environment. However, when the sensing capabilities of the robotic platform are limited, creating maps and keeping track of the localization become cumbersome. Our project is aimed at providing a solution to this problem by using computer vision algorithms to extract information from the environment. More specifically, we extracted images from the single camera in the Rovio platform, and use them to detect obstacles, extract distances, and keep track of the position and orientation of the robot in order to enable autonomous navigation.**

I. Introduction

Vision has become one of the most useful techniques in robotic environments. Images provide extensive information that can be used to implement common tasks that enable robots to understand their surroundings. Since cameras are usually cheap and power-efficient devices compared to other more sophisticated sensors, the use of computer vision becomes a good alternative for sensing purposes. Nevertheless, extracting information from images is not straightforward task. Images usually consist of two dimensional grids composed of pixels that contain color and brightness data. This representation maps real objects into two dimensional structures and therefore the depth information is lost. Consequently, distances and angles are not easily identifiable as may be the case with other more expensive sensors.

Currently, many frameworks already exist for doing localization and mapping. Since these implementations are not dependent on the type of sensing device, one should be able to use the camera to sense the environment and then feed the obtained data to the existing algorithms. As long as accurate measurements are extracted, the robot should be able to implement robust localization and mapping techniques. The actual problem lies on obtaining the correct information from images. Our main goal was precisely to implement vision techniques to obtain accurate data, and then use it to enable the robot to perceive the environment and act accordingly.

Recently, the area of visual processing has become more popular and significant research has resulted in very sophisticated vision algorithms. More precisely in robotics, the area of visual SLAM (Simultaneous Localization and Mapping) is currently receiving much attention. Some good examples are given in [2], [3], [4]. Some of the previous work is based on using a single camera to obtain information directly from images [3], whereas others use multiple cameras to perform different types of triangulations in order to obtain the desired information. [4]. One of our goals was to implement similar approaches and combine them to obtain accurate distances to obstacles.

The end goal of our project was to implement localization and mapping using the single camera in the Rovio platform. Rovio is a WIFI-controlled robotic webcam developed by WowWee. It has navigation capabilities and it constantly streams images that can be retrieved and processed. We chose Rovio because toys and home applications are among the most promising market fields for the use of vision in robotics. In order to achieve localization and mapping we divided our tasks into three stages. The first stage consisted in accurately measuring the distance to the obstacles around the robot. For this purpose, we combined multiple distancing algorithms. The second stage consisted on keeping track of the angle rotation of the robot while exploring the environment. For this, too, we used vision techniques that enable the robot to identify features that enable it to keep track of orientation. Finally, the obtained data from the distance detection and the localization was send to an implementation of an occupancy grid that represents the environment. The rest of the paper describes each of the three stages in more detail. Our implementations were tested in the robotics laboratory at Cornell University, and the results are presented at the end.

II. Object Distancing

A. General

To extract the distancing to the obstacles from the images, we first used a combination of filters to extract the edges. We use a 7x7 Gaussian filter to smooth out the image in an attempt to ignore the edges normally detected in the coloring of the carpet. The image is then converted to grayscale. In order to extract the edges from the image we use a canny filter with threshold values of 0.25 and .09 for the initial detection and the connected edge detection, respectively. This produces nice strong edges to base our measurements on. These threshold values were varied and the outputs analyzed to determine the best tradeoff in terms of false ground edges and strong obstacle edges.

Fig. 1. Rovio image with head in lower position and edge extraction.

Fig. 3. Rovio lowest detected edges with head in lower position and distance output

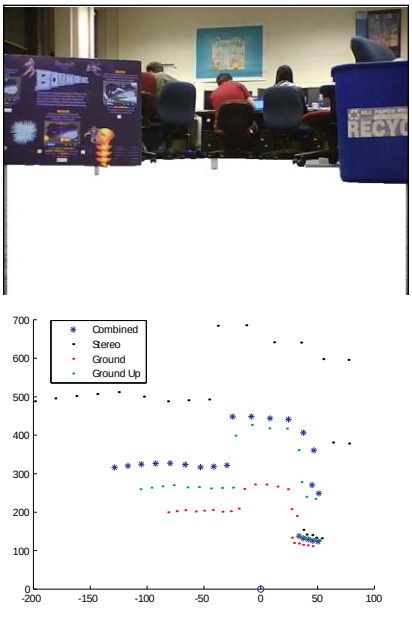


Fig. 2. Rovio image with head in lower position and edge extraction.

B. *Stereo Distancing*

The stereo vision algorithm uses an image from each of the Rovio's head positions at a stationary position. This enables us to determine the distance between the same edge in both images, which correlates to an actual distance to the object. The first step in this algorithm is to determine the pixel distance between corresponding edges in each of the images. Our approach was simply to find the bottommost edges at each column of the image taken with the upper head position. This was done with the upper image because the lower image often detected edges in the carpet immediately in front of the robot, while the vantage point of the upper head position did not detect these edges. By only taking into account the bottommost edges we were able to focus on the obstacles in the immediate vicinity of the robot and did not have to deal with more complex ways of determining what constituted a unique distinct edge, and matching edges between images. Using data obtained from controlled experiments we determined the pixel distance ranges that would be experienced in the lab environment, based on the closest obstacle that would show up in both images being the lower end and the size of the lab being the upper end of the range. Since we are dealing with the closest edge to the bottom in the upper image, the corresponding edge in the lower image will be the first one seen in the same column starting from the minimum pixel distance and moving to the maximum. An edge that was not detected between these ranges indicates a missing corresponding edge in one of the images and that measurement is ignored.

The bottom edge could not be used from the bottom image because of the existence of false edges detected in the carpet. This is the reasoning behind using the bottom edge in the top image and iterating through the range of possible pixel distances from that edge to find the corresponding edge in the lower image. A built-in Matlab regression algorithm was used to fit the data points taken to a function that returns the actual distance to an obstacle given the pixel distance between the stereo images.

C. *Ground Distancing*

Two other algorithms were used to make the distancing robust because of the subtle differences in edges detected by the obstacle-carpet intersection. Depending on the lighting and the particular obstacle edges would be detected at different heights especially between the upper and lower images most likely because of the vantage point difference. For example, the edge detected between a trash can and the carpet would always be higher than one detected on a wall at the same distance. For this reason, to make this algorithm robust and applicable to a real world environment with varied obstacles it was necessary to make multiple distancing algorithms to compensate. This second approach involved using the absolute pixel distance from the bottom of the image to range the objects. The theory behind this is that further away obstacles appear higher up in the image, or closer to the horizon. This enables us to determine a mapping from the lowest edge pixel location in the image to an actual distance for each of the upper and lower images. Again, a built-in Matlab regression algorithm was used to fit data points taken for each of the head positions to functions that return the actual distance given the edge location in the image.

The absolute pixel location of the bottom edges in the image from the upper head position was already determined from the stereo vision algorithm, so this merely required passing that location into the distancing function. The pixel location of the edges in the image from the lower head position was a bit trickier. After some analysis of the edge extracted images in different lighting environments it was determined that the false edges detected in the carpet were nearly always below a certain pixel location in the image.

Thus, any pixels below this point were discarded and the pixel location of the lowest edges of the resulting image was determined for each column. These locations were then passed into the function to map the pixel locations of the lower head position image into obstacle distances. An example image is given in Figure 3 (the two seemingly

random edges detected in the middle are actually objects on the carpet).

D. Output Distance

In each of these algorithms the image was segmented into 24 discrete measurements, each represented with its own angle off of the orientation of the robot. Through testing we had determined that the Rovio's angle of vision was around 48.5 degrees horizontally. The distances between the pixels corresponding to the discretely sampled angles were calculated using basic geometry theory. Assume a point object at a distance of d from a straight line. The perpendicular distance of the line from the point is going to be at vertical angle of 0 degrees. However, when the angle to the line form the vertical distance is varied on either side, the distance change of the new point from the central point is going to increase nonlinearly. Thus, to account for the variation in the pixel point as the angle is varied from -24.25 to 24.25 (where 0 degrees corresponds the central pixel), a simple sine function was used to account for the change in pixels. This is given as follows:

$$pixel\ value = 316 + 316 * sine\left(\frac{x}{48.5} * 90\right) \quad (1)$$

Where x represents the angle that the pixel value is desired and image width used was 632 pixels.

Thus the number of pixels associated with each of these angles is non-uniform and the area representing an angle at the edge of the image is larger than that in the center of the image. The range of pixels corresponding to each angle was taken to be from the center of the segments on either side of that particular angle. In each segment the median value was taken to be the measurement of that segment to ignore extreme values.

These distances were plotting according to the angular offset from the center of vision from the Rovio to plot the exact location of the obstacle.

This gives us three different distance measurements to work with, creating a more robust distancing algorithm. After much testing and analysis it was decided that the most effective way of combining these three distancing approaches was to average them together. This averaging was done for each of the 24 segments. An exception to this combination is when the distance given by the absolute pixel location for the image with the Rovio's head in the lower position is less than 80 mm. In this case the image with the Rovio's head in the upper position cannot see the intersection of the carpet and the obstacle because the distance is too small, so the lower head position measurement is used. The reason these measurements are averaged normally is that they generally compensate for small errors because of the imperfections in the edge detection and accuracy based on the size of the image. These combined measurements are very accurate and almost always give better results than any single algorithm. Although the stereo vision is very accurate, it is imperfect because of the edge detection picking up different edges based on the differing perspectives of the rovio's head in the lower and upper positions. In the upper position, the lighting can cause the rovio to detect a lower edge of the obstacle because of its better vantage point of the intersecting line between the carpet and the obstacle. This mainly occurs for more distant obstacles because the further away the object is the less defined the intersection is with the lower head position. In fact, for close obstacles all of the distancing algorithms give accurate values within 15mm of each other.

III. SIFT LOCALIZATION

A. General

One difficulty we encountered when trying to obtain a map of the environment was the inaccuracy of the movements of Rovio. While mapping a specific section, the robot is programmed to constantly rotate, take pictures, and process them to obtain distances to obstacles. Nevertheless, every time we send a specific movement command to the robot, it has a significant error associated with it. Furthermore, the commands would sometimes get lost in the network and the program would not have any feedback on the actual actions taken by the robot. Consequently, when rotating the robot multiple times, the error would increase and the positions of the obstacles would be mapped to wrong places in the map. Moreover, the robot would not have information about its orientation and therefore no more mapping would be possible.

For this reason, we needed a way to read the angle rotations that would allow it to keep track of its orientation without relying on the actual commands. Since the camera is the only sensing capability of Rovio, we decided to use a vision approach to obtaining the angles. The method we implemented consisted of extracting features from the images using the SIFT algorithm and the comparing them to obtain an estimate of the actual rotation angle.

B. SIFT

Scale-invariant feature transform is a computer vision algorithm published by David Lowe [1] that extracts local features from images. These features are scale and orientation invariant, and to a certain extent, they are also invariant to noise and illumination. From an intuitive perspective, the algorithm works by applying a cascade of filters at different scales, and then finding the features that are invariant to certain transformations. From a user point of view, the algorithm provides two important functions. The first one is for feature extraction and it receives an image input and outputs an array of descriptors that have feature information, together with the index in the original image. The second function is in charge of matching between two different images. It receives the images, the descriptors array for each of them and returns an array tuples of size two that have the index of each image for each match.

For our project, we used an implementation of the protocol for MATLAB developed by Andrea Vedaldi from the University of California Los Angeles.

C. OUR IMPLEMENTATION

To keep track of the rotation angle of the robot we used the invariant features provided by SIFT. The implemented method consisted of using the images that were already taken for the distancing algorithm, and perform feature extraction using SIFT. After each rotation, a new picture is taken and new features are obtained. The features from subsequent angle positions are compared and their difference is mapped to a rotation angle.

The feature descriptor provided by SIFT gives information about the position and rotation of the features. Since we know a priori that the robot is only rotated we can reinforce the requirement that features must have very similar y-axis position, and orientation, and features that do not fit these requisites are discarded.

Initially, the idea was to either fit an experimental function, or use a learning algorithm to accurately map the pixel distance of the features to an able. Nevertheless, a simple linear function using the able span of the Rovio camera and the number of horizontal pixels of the image gave excellent results. Figure 5 illustrates the angle extraction method. The first two images are the pictures taken by Rovio. The left-hand image is the one taken in the initial position, and the right-hand image is the one taken after rotating left. The second picture presents the extracted features from the environment, and the green lines connect the features that matched between the two pictures. Clearly, the images with the features are a cropped version of the original images, and this was done intentionally to increase the speed of the

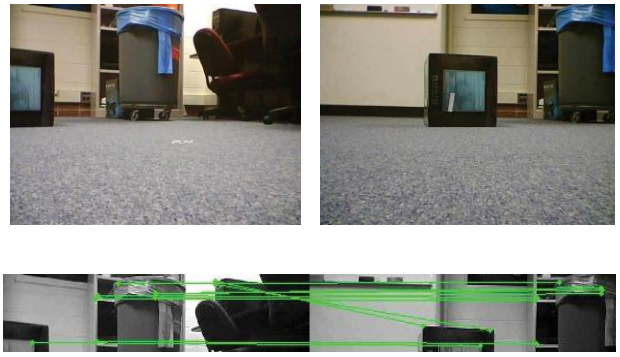


Fig. 4 Feature extraction for angle estimation.

algorithm, because it is very time consuming. Furthermore, we know that the bottom most part of the image which includes the carpet is not going to be optimal for feature extraction. In addition, we used a very high threshold for the feature extraction function and for the matching function as well. This is the reason for which there are very few green lines in the image with feature matches.

Furthermore, at some instances the input angle to the function was predefined, and the implemented function could use this information to further increment the speed. In these cases, the image would also be cropped vertically because for example for a 20 degree rotation angle, we know a priori that certain sections of the picture are not going to match and we don't need to use them in the extraction and matching process.

Overall, the precision of the implemented technique was very satisfactory. We found that after doing two full 360 degree turns with increment rotation of 20 degrees, the robot would t the end have a mismatch of approximately 2 degrees which is very accurate considering that there were 36 turns involved and also considering that without our implementation the robot could reach mismatches of more than 60 degrees under the same circumstances.

IV. MAPPING

A. General

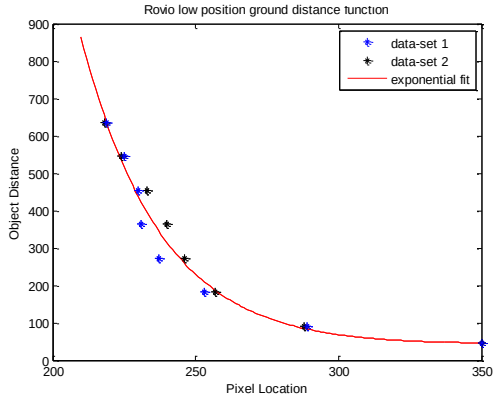
The above two sections describe the information that the Rovio robot analyzes to define its environment. However, in order to successfully navigate the robot in its surrounding, a map of the environment is required to track the placement of obstacles and the pose of the robot. Different options can be used to represent the map of the environment but the two basic options that were considered were a point cloud representation and an occupancy grid.

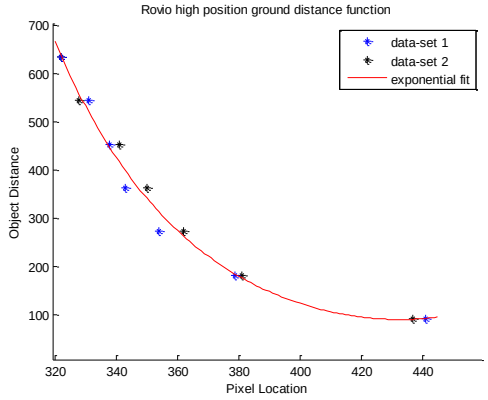
B. Mapping Representation Considerations

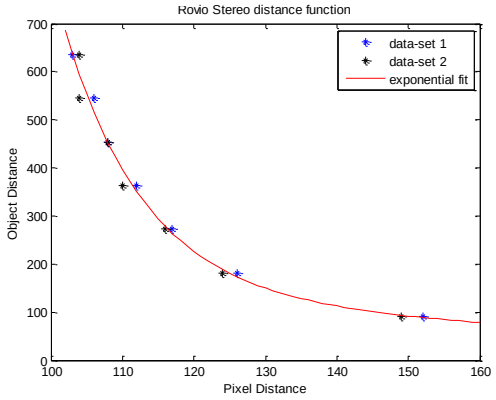
A point cloud representation of the map displays the actual distances that are calculated by the distancing algorithms. This can give rise to obstacle features and corners in the map. However, this method requires the ranging information to be highly accurate such that the obstacles can be given shape during representation. As will be showed in the experiments sections, the distancing algorithms have a certain error associated with them and thus, the point cloud would not be able to account for the variation in the ranging information and might skew the shape of the obstacle as a result and would thus be undesirable. Another con of using a point cloud in a real time moving robot is that since each distance range found at an angle is stored, the number of points to keep track of could increase drastically. Because of these issues, a point cloud was an inefficient way to keep track of obstacle positions.

In our setup, an occupancy grid was more preferable because of its discrete nature and limited memory usage. It can be used to efficiently track an approximate position of the obstacle and avoid it well before the robot reaches it. An occupancy grid is represented by a mesh of cells (rows and columns) and each cell has a certain fixed area on the map associated with it such that all the points in that area are represented by that singular cell. The area represented by a cell of the occupancy grid can be set using any parameter/ condition as desired by the specific implementation. For example, a very fine grid would give an (almost) point cloud representation of the map. A very large cell area might surround the robot with obstacles in the map even if there are only a few surrounding the actual robot. For our implementation, the maximum variation in the distancing algorithms was used to determine the edge of a cell in the map. Even though the map was represented as a discrete grid, the robot movement was continuous.

Figure 5: Vision Distancing data







To define whether an obstacle exists in the occupancy grid, a threshold value was set such that if the number of distancing points detected in a particular cell exceeded that threshold, the cell would be marked as occupied. Thus, the occupancy of a cell would depend on whether its probability was high enough.

C. Control Sequence

A control sequence was implemented to control the efficient movement of the robot and to ensure that the map of the surrounding environment was created efficiently and the pose of the robot was tracked accurately. The following sequence was following by the Rovio robot in mapping: (Initially, robot placed in some location on the map with pose (0,0,0) corresponding to position and orientation)

- Rotation sequence begin - At the current pose (starting from the initial position), the robot was to move its head up and down and grab the images corresponding to the 2 head positions.
- These images were forwarded to the distancing algorithm to find the distanced to each of the obstacles at particular angles. These obstacles were mapped to the occupancy grid using the current pose.
- The robot was then turned left by 20 degrees and an image was grabbed at the new orientation. The SIFT algorithm was provided with the image from before and after the turning to determine the precise angle of rotation during the turn. This angle change was reflected in the robot pose as well.
- If the robot did not turn 360 degrees in the rotation sequence, then the robot would keep on turning. Otherwise, the rotation sequence would end and the robot would analyze the current map to obtain the most open or the cleanest direction that it can travel in. It would calculate the approximate distance to travel in this direction and would move until it is a certain distance away from the map obstacle or edge.
- After traveling the required distance, the robot would initiate the rotation sequence again and start building another part of the map. It would discover new regions of the map and would reinforce old areas as well.

This sequence corresponds to a semi-online SLAM implementation as the robot only builds the map, localizes itself and calculates the distances when it's in the rotation sequence but does not do any image processing while it is moving towards its next goal position (as that would introduce a large amount of lag in the movement). This implementation gave a very good representation of the map of the setup environment as is explained in the experiments section.

V. EXPERIMENTS

A. Distancing

The distancing algorithms had to be calibrated for each individual Rovio because of the non-uniformity in the design of the Rovios. The camera angles of the Rovios were not all exactly the same in the down position and the distances they moved as well as the angle of the up position were varied even more. This lead to acquiring data for many Rovios until a working one was settled upon to use exclusively. The data acquisition to fit functions to the stereo and each of the ground distancing algorithms was performed by hand because of a lack of a real distance sensor on the Rovio. Thus we used two different obstacles, each of which gave a slightly different behavior of the edges detected as previously described. For each of these we took a number of data points at discrete distances and recorded the edge location of the obstacle in the edge extracted image for each of the lower and upper head positions. After analyzing the data patterns and fitting different function types to the data it was determined that the data for each of the algorithms followed exponential functions. Using Matlab's built-in regression fitting algorithm the data-sets were fit to function of the form

$$f(x) = ae^{b*x} + ae^{b*x} \qquad (2)$$

The data used for the ground algorithms was simply the pixel location of the obstacle in their respective images. The data used for the stereo algorithm was the pixel difference between the two edges. The independent variable was the pixel location, or distance, outputting the actual distance to the obstacle. The data and exponential function fit to each of them are given in Figure 5.

B. Mapping

To test the rovio's distancing algorithms and SIFT localization, an environment was set up with a variety of features. Obstacles were included in the environment that had a well defined edge and that could be mapped easily. Other varied obstacles were also included to test the distancing algorithms on a wide variety of surfaces. The distribution of the obstacles in the environment was varied as well such that we could test obstacles that were closely stacked and obstacles that were sparsely placed. The whole setup was segmented into 2 sections such that the rovio is able to go to a different section when its moving and map distances to new obstacles placed in that section. A few images of the environment as shown as follows and the actual scaled environment representation is shown in figure 6:

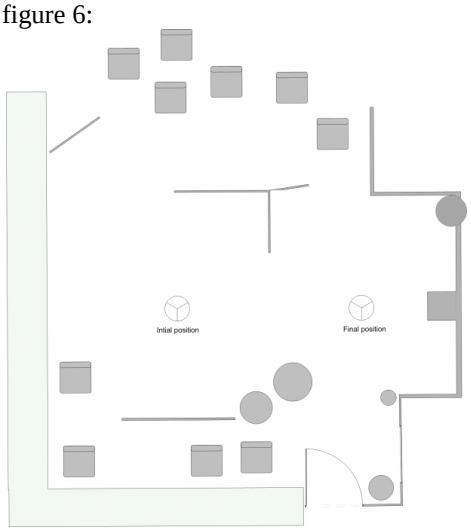
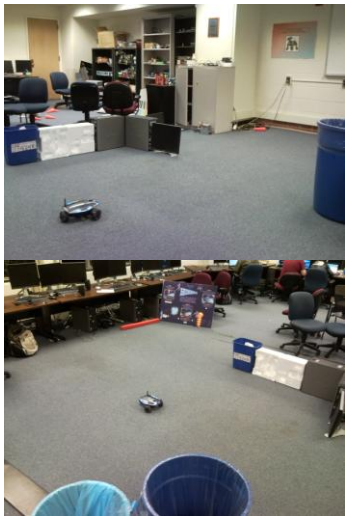


Fig. 6. Environment map (walls are thicker than they appear)



Using this environment, the control sequence was executed to test the response of the rovio. As was shown in the video during the poster session presentation, the rovio was able to map the room very well at its initial location and was able to find the cleanest direction in the map. It was then able to travel in this direction and stop at a certain distance away from the obstacle and map the environment again. The following is a snapshot of the final pose and map obtained by the rovio as shown in the video.

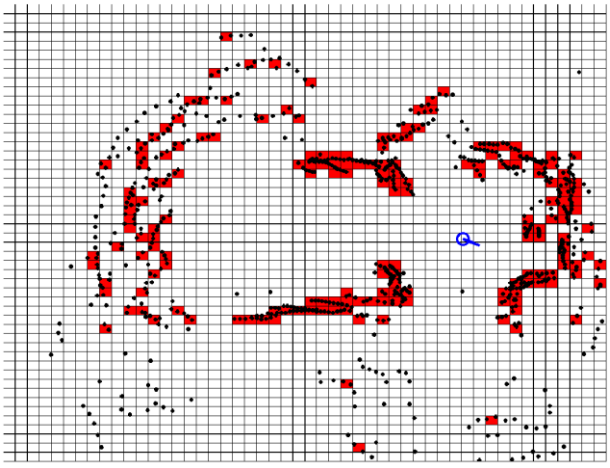


Fig. 7. Experimental map. Black = data point. Red = occupied grid

A to-scale mapping of the environment is given in Figure 6. This corresponds closely to the output of the mapping function as given in Figure 7. In fact, all of the well defined obstacles in the occupancy grid are very close to their actual positions. The back desk is not considered a well defined obstacle because it is off of the ground but its shadow creates some edges, causing problems for the algorithm. As can be seen in Figure 6 there is no concentration of pixels for the desk, aside from the chair and backpack in the front of it, because of the shadow edges. The distancing in conjunction with angular localization and mapping worked very well, accurately mapping the environment with an average of 18.46 cm error. Table I gives some mapped positions and actual positions of the same points. The actual distances were roughly measured and are by no means perfect, but there is a close correspondence between these actual and mapped positions. The positions are based on the starting position of the robot at 500,500. All measurements are in cm.

The algorithm performed very well given the varied nature of obstacles used. This approach was used to determine the robustness and ability of the algorithm to adapt to new obstacles. More testing is necessary to verify it for varied, larger environments. Also, since obstacle classification is not performed this algorithm would not work well with tiled floors with different colored tiles or edges that would be detected between them.

TABLE I
Mapping vs. Actual Positions and Associated Errors

Mapped Position (x,y)	Actual Position (x,y)	Distance Error
(500,653)	(500,659)	6 cm
(632,637)	(649,653)	18.03 cm
(651,593)	(649,571)	22.09 cm
(500,331)	(500,350)	19 cm
(866,491)	(886,493)	20.10 cm
(858,401)	(841,402)	17.03 cm
(795,656)	(795,629)	27 cm
Average Error		18.46 cm

VI. Conclusions

The unique challenges presented by the usage of Rovio robot as the choice of mobile robot system allowed us to efficiently optimize our code and develop algorithms and control sequence to map out the environment and localize the robot in it. The limitations offered by the rovio robot with regards to connectivity, localization data and movement allowed us to explore different techniques to improve performance and dissect different aspects of the robot such that we could create a good representation of the surroundings. Given the extensive limitations of the rovio platform, especially with the useless localization and control commands, we were able to successfully map out a simple environment using only a camera. This work can be extended to different platforms and would be useful if implemented in simple robots with visual sensors. More work needs to be done to speed up the processing time and optimize the algorithms to be more robust and work in different environments with varying obstacles, but it has been proven to be effective thus far. Future work in this area can include SURF implementation for angular localization and feature magnification as another technique to find the distance to an obstacle/ feature in the image.

References

[1] Lowe, D.G. 1999. Object recognition from local scale-invariant features In *International Conference on Computer Vision*, Corfu,

[2] Kaess, M. and Dellaert, F., \Visual SLAM with a multi-camera rig," Tech. Rep. GIT-GVU-06-06, Georgia Institute of Technology, Feb 2006.

[3] A.J. Davison. Real-time simultaneous localisation and mapping with a single camera. In Intl. Conf. on Computer Vision. (ICCV), pages 1403–1410, 2003.

[4] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. In IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), volume 1, pages 652–659, 2004.