Obstacle Avoidance and Color Detection using Image Segmentation

Kamruzzaman Rony (kr334) and Norris Xu (nx26)

Abstract

Rovio, produced by WowWee, is a small robot equipped with a webcam, IR sensor, base station localization signal, and a unique wheel setup allowing it to move sideways and diagonally as well as forward and backward. The goal of this project is to program Rovio to successfully avoid any obstacles while navigating an indoor environment and simultaneously searching for a color-coded target object.

For this project, we used only Rovio's webcam. We used some median filters to smooth the image and then applied image segmentation on the smoothed image; we then decided which of the segments was the floor and used the position of other segments to see where the obstacles were and avoid them. We used a simple color filter to find the target object and programmed Rovio to move sideways to avoid obstacles while keeping the target in view. We tested our code in a variety of different situations and the success rate was very high: in almost all trials, Rovio was able to successfully avoid obstacles and find the target.

1. INTRODUCTION

1.1. Robot

The robot that we used in our project was Rovio, which is produced by WowWee. Rovio is a small robot with a web interface. Its sensors include a webcam, simple IR sensor (binary: either there is an obstacle in front of Rovio, or there isn't), and localization using a signal projected onto the ceiling by the base station. It also has a unique wheel setup which allows it to move in the eight major compass directions, and can raise its camera. However, we used only the wheels and the webcam for our project.

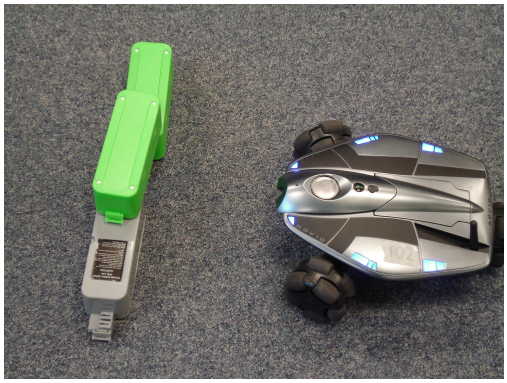


Figure 1: Rovio in a typical environment

1.2. Environment

Rovio is designed for indoor environments only, so those are the environments we designed our algorithm to work in. Our typical test environment had a carpeted, level floor and was well lit by overhead fluorescent lighting; we had a wide range of obstacles including chairs, tables, cardboard boxes, flashlights, cups, other Rovios, and a bright orange cone which was our target object.

1.3. Goal

Our goal was to get Rovio to navigate its environment and avoid obstacles while searching for the target object, using only its webcam for sensor data. Once Rovio sees the target object, it should drive towards it while continuing to avoid obstacles, and once it gets close enough (we defined "close enough" as about a third of a meter) to the target, it should halt.

1.4. Method Overview

Since our sensor data consisted of images, we used image processing to accomplish our goal. Our algorithm was based on image segmentation, augmented with median filters to eliminate noise and improve the

segmentation algorithm accuracy. We then classified each of the segments into one of three categories: the segment was either the floor, the target object, or an obstacle. The robot would then behave accordingly: if there was an obstacle in front of it, it would navigate to avoid the obstacle; if it sees the target it will try to move towards it or stop if close enough; otherwise, it will simply go straight forward.

2. RELATED WORK

A great deal of research has been done in the field of robotics and computer vision for obstacle detection and avoidance using monocular vision [2] and stereo vision sensors [3] as well. In recent years, using the monocular vision has been proved to be a very efficient method for obstacle avoidance and navigation. Many different types of methods have been used for obstacle detection and avoidance ranging from appearance based obstacle detection [4] to obstacle avoidance with localization and mapping using visual sonar [5]. Our project uses graph based image segmentation [1] along with our own classification algorithm for obstacle detection. To avoid obstacles and to reach the target object we also used our own color detection and navigation control algorithm. To detect obstacles using monocular vision Ulrich [4] classifies each individual image pixel belonging to either the obstacles or the ground based on that pixel's color appearance. Lenser [5] uses visual sonar for obstacle detection, localization and mapping by developing an algorithm that takes monocular vision as its input and generates output like a sonar sensor. Visual sonar has the capability of detecting modeled objects and obstacles in its course and also avoiding unmodeled or unknown objects and obstacles. The approaches that Choi [6] and Wang [7] employed for obstacle detection (differentiating between the floor and other objects in the image) are similar to our (image segmentation) approach.

3. METHOD

3.1. Segmentation

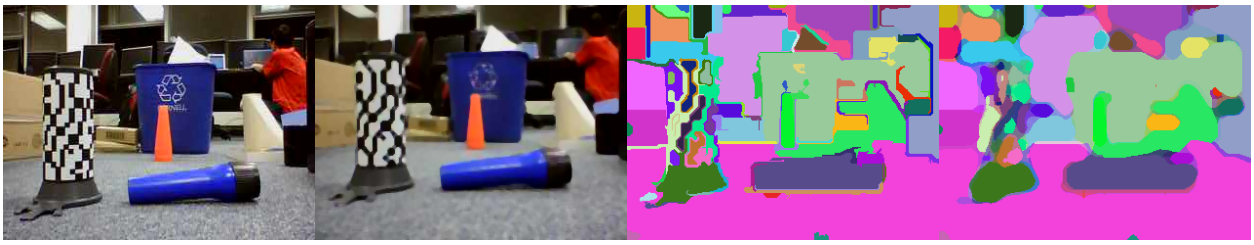
Our method first attempted to find all objects seen by Rovio's camera. Since we are only avoiding the obstacles, this part does not have to be very precise — we simply need to know the general location and size of the objects; we do not need to know their exact shape or other details. Thus, we first applied a very strong median filter to smooth the image out and help eliminate noise as well as give textured surfaces (such as the carpet) a more uniform appearance, to improve the accuracy of the segmentation algorithm [1]. We then ran the image segmentation algorithm on this filtered image, and then ran another median filter on the segmented image. This second median filter's purpose was to help reduce erroneous segments, which occasionally appeared on the floor, as well as to reduce the number of segments overall and thereby reduce the complexity of the segmented image. The filter eliminated very small or very thin segments; for example, boundaries between objects tended to become their own very thin segment, which the filter removed. Again, we only need a general idea of where the objects are, so we could afford to apply a very strong median filter.

3.2. Interpretation

Since Rovio would be operating in environments where the floor is flat, and Rovio's camera is parallel to the ground, we divided this filtered segmented image into two halves: a top half and a bottom half. The line dividing the two halves is the horizon, and it is not possible for the floor to be seen in the top half (assuming that it is flat). We constructed a list of segments; for each segment, we stored its bounding box in the image as well as its size in pixels.

3.2.1. Obstacle Avoidance. We assigned the floor to be the largest (by number of pixels) segment in the bottom half. This method worked most of the time; however, in the case where an object is very close to Rovio and occludes most its field of view, this method would erroneously assume that this object is the floor. To handle this case, we introduced a heuristic where if a sufficiently large part of the "floor" was in the top half of the image, then the algorithm would decide that this segment was in fact a nearby object and reverse away from it instead. We defined "sufficiently large" to be 70%: if the number of pixels of "floor" in the top half was at least 70% of the number of pixels of "floor" in the bottom half, then we would reverse. In our trials, this number worked very well.

If we did not reverse, then we need to decide if it is possible to go forward without hitting anything. We defined a rectangle in the middle of the image (annotated in Fig. 3 as the obstacle rectangle), and checked to see if any non-floor segments intersected it (intersection tests are very simple since we have the segment bounding boxes; this is less accurate than a pixel-by-pixel test but we can afford to be imprecise). The rectangle is offset from the bottom of the image to avoid erroneous carpet segments, which tended to appear in the very bottom of the image (probably due to camera focal effects); one such segment is visible at the very bottom of Fig. 2d.



(a) Raw image (b) Filtered image (c) Segmented image (d) Filtered Segmented image

Figure 2: Results of image segmentation

If nothing intersects the rectangle, then we can move forward.

If something did intersect the rectangle, then we need to turn. We have a choice between left and right, and want to pick the one that requires the least amount of turning before we can move forward again. To do this, we simply look at the left and right bounds of all of the obstacles in the lower half (we ignore the upper half of the image since that is above the horizon and thus background), and find the leftmost and rightmost obstacle boundaries. We turn towards whichever of these is closest to the center of the image.

3.2.2. Target Tracking. Besides avoiding the obstacles, we will also want to move towards the target object, if we can see it. To determine whether or not we can see the target, we used a simple color detection algorithm which looks for pixels in the image that are close to a given target color (we define "close enough" as the sum of the squares of the differences of intensities in red, green, and blue is under a specified threshold, which we tuned through experimentation). The color detection algorithm is run on the raw image, since this time we do care about the shape of the detected object, as knowing the exact shape will greatly improve accuracy of the next step. We also only run the detection algorithm on the bottom half of the image, since again assuming a flat floor, any objects should have at least a few pixels in the bottom half, depending on how far away they are. The next step is matching a segment with the detected pixels. Again, we use the unfiltered segmented image for matching, since it is more accurate. We simply count how many detected pixels each segment contains, and pick the target segment to be the one with the most detected pixels.

We then find the target segment in the blurred segmented image by looking for the segment with the same color. Finally, we want to move towards the target while continuing to avoid obstacles. First of all, we want to keep the target centered: we try to keep the centroid of the target segment within a triangular region (annotated in Fig. 3 as the target bounds). To do this, we will try to move forward only if the centroid is within the region; if it is outside of the region, we will move diagonally forward to move the centroid back within the region (originally we attempted to get Rovio to turn very slightly, but it turns far too quickly to be able to center the target). If we are very close to the target, then instead of moving diagonally forward, we slide sideways instead. If the target is sufficiently close, then we've reached the target and stop. We check the proximity of the target by measuring the distance between the bottom edge of the target segment and the bottom of the image.

Of course, we must do all of the above while continuing to avoid obstacles. If we can see the target, then we only care about obstacles between us and the target; i.e. obstacles with a bottom edge that is lower than the bottom edge of the target. We can safely ignore all other obstacles since they are behind the target. We do the same check as before on this reduced obstacle set to see if we are able to move forward. If we are not able to move forward, then there is an obstacle between us and the target and we must dodge it. We check the leftmost and rightmost obstacle boundaries, as before, but this time, if we can see the end of the obstacle (that is, the obstacle does not extend all the way to the edge of the image), then we will sidestep it and thus keep the target in view. If the obstacle does extend all the way to the edge, then we can't be certain that we can sidestep it, so we just turn as before and hopefully find our way around it to the target.

4. TRIALS

We tested Rovio on three courses: a simple U-shaped obstacle course (Fig. 4a), another U-shaped course with low barriers added to test the algorithm's sidestep routine and see if Rovio could successfully avoid the obstacle while keeping the target in view and continuing to move towards it (Fig. 4b), and an open

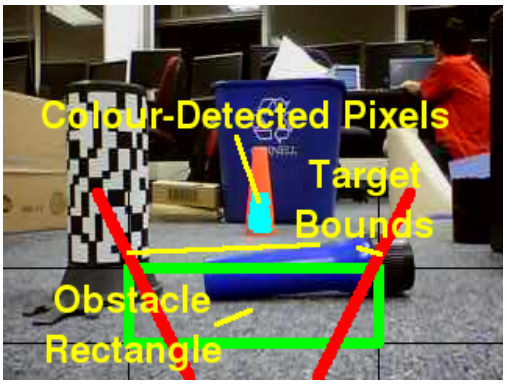


Figure 3: Illustration of object bounds

environment that was very large and had many obstacles; it was in fact a classroom (Fig. 4c). We ran many trials on each course; without targets, Rovio's obstacle avoidance was almost perfect (the only time it ever hit anything was when the obstacle was in front of Rovio and sufficiently far to the side that it was not seen at all by the camera, yet close enough to hit Rovio; this only happened once or twice). With the target object, Rovio would occasionally hit obstacles while moving sideways to keep the target centered, since its camera can only see what is in front of it. The paths shown on Fig. 4a and 4b are the paths taken by Rovio; see the videos. Overall, the success rate of Rovio being able to find and reach the target object without hitting an obstacle was above 80%.

We also did a few tests in very different environments: specifically, in a hall with a reflective floor (the hall outside Upson 317) and during the poster session (in Duffield Atrium). We found that the algorithm performed very poorly on a reflective floor: reflections caused the floor to be split into multiple segments, so Rovio would try to avoid obstacles that were not there. In Duffield Atrium, the algorithm's performance was okay, though not as great as before; this was due to the low lighting which caused the segmentation algorithm to sometimes be unable to distinguish the brown boxes we used from the floor, as well as the increased lag due to many people attempting to use the wireless connection simultaneously.

5. CONCLUSIONS AND PROBLEMS ENCOUNTERED

Our image segmentation method of detecting and avoiding obstacles performed very effectively; the success rate was very high. Nearly 100% of obstacle collisions were due to Rovio moving sideways and colliding with something that it could not see; if we could eliminate all sideways movement, the collision rate would be almost 0%. The main difficulty that we encountered was the lag inherent in our control method: by the time we received an image to process, it was already out of date. For forward and backward movement, this did not present a problem since the image did not change much when moving forward or backward, but when turning, this presented a major problem. Due to lag and a high minimum turning speed, we were unable to use turning as a method for centering the target, as by the time the algorithm saw that the target was centered, Rovio had already turned too far; instead, we were forced to use the unsafe sideways movement for centering.

Reflective floors also posed a difficulty for the algorithm. Reflections of the overhead fluorescent lights in particular caused the floor to be segmented into two separate segments, which resulted in Rovio attempting to avoid obstacles that were not there. A possible method for rectifying this would be to merge similar segments, or possibly to use Rovio's IR sensor to see if the obstacles are real or not. Likewise, for future work, we would like to be able to teach Rovio to not avoid flat "obstacles" such as pieces of paper; again, using the IR sensor would be a possible method. It might also be possible to lift and lower Rovio's camera to simulate stereo vision and use that to figure out whether detected obstacles are real or not. Another possible direction for future work would be ramps: our algorithm makes extensive use of the horizon line, so it would be unable to handle steep or very long ramps, or other sudden changes in slope.[1] Again, stereo vision would be useful here.

6. ACKNOWLEDGMENTS

- ROS [2]
- OpenCV [3]
- Cornell CS Robotics Lab at Upson 317
- Jonathan Diamond (for Rovio ROS drivers)
- Ashutosh Saxena (for guidelines and suggestions)

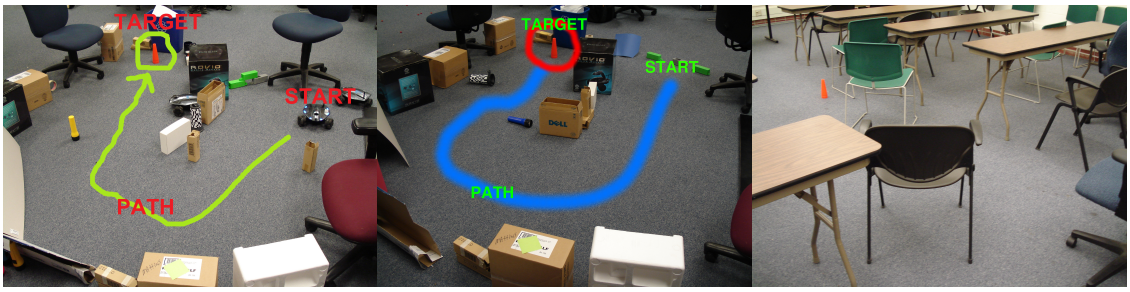
References

[1] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient Graph-Based Image Segmentation," *International Jour-*

[1] We did a brief test on a ramp, and found that physically, Rovio's motors aren't powerful enough for it to perform well on a ramp, either: Rovio tended to slip down, especially when moving laterally on the ramp. Any future work in this direction would probably require a different robot, or at least better motors.

[2] http://www.ros.org/wiki/

[3] http://opencv.willowgarage.com/wiki/



(a) Obstacle course 1 (b) Obstacle course 2 (c) Obstacle course 3

Figure 4: Obstacle courses

nal of Computer Vision, vol. 59, no. 2, pp. 167–181, September 2004.

[2] T. Taylor, S. Geva and W. Boles, "Monocular Vision as a Range Sensor," in *Proceedings of CIMCA*, Australia, 2004.

[3] M. Kumano, A. Ohya and S. Yuta, "Obstacle Avoidance of Autonomous Mobile Robot using Stereo Vision Sensor," in *Proc. of the 2nd International Symposium on Robotics and Automation*, Monterrey, pp. 497–502.

[4] I. Ulrich and I. Nourbakhsh, "Appearance-Based Obstacle Detection with Monocular Color Vision," in *17th National Conference on Artificial Intelligence*, Austin, Texas, 2000.

[5] S. Lenser and M. Veloso, "Visual Sonar: Fast Obstacle Avoidance Using Monocular Vision," in *Proceedings of IROS03*, 2003.

[6] S. Choi, T. Jin and J. Lee, "Obstacle Avoidance Algorithm for Visual Navigation using Ultrasonic Sensors and a CCD Camera," *Artificial Life and Robotics*, vol. 7, no. 3, pp. 132–135, September 2003.

[7] Y. Wang, S. Fang, Y. Cao and H. Sun, "Image-Based Exploration Obstacle Avoidance for Mobile Robot," in *Chinese Control and Decision Conference (CCDC)*, 2009.