Rovio and Juliet: Vision-based Autonomous Navigation with Real-time Obstacle Learning

Jae Yong Sung, Jong Hwi Lee, Supasorn Suwajanakorn

***Abstract*— Our Rovio project's goal is to make Rovio a totally autonomous robot which can follow waypoints only based on image and reach the goal position while learning and avoiding obstacles on its way. Rovio will utilize the SURF (Speeded Up Robust Features) to determine the direction. Rovio uses vision to detect the properties of obstacles with various shapes and colors and uses IR sensor to learn whether they are obstacles or non-obstacles such as a flat tiles on the floor. It then analyzes the information about the obstacles and decides which direction it should move to based on the trapezoid safe zone. Our program uses CGI commands for communication with our Rovio, as written in Rovio API Specification. Wrapper classes were written in Java, with standard java.net for network connection and other open source projects for image processing.**

I. Introduction

For a robot with only a visual sensor, autonomously moving from a point to another is a difficult task. Many algorithms have been developed and optimized in different surroundings. In common, the main points of those previous researches were on how to interpret the sensor data and more accurately analyze the robot's situation from what it knows. Well known techniques relevant to this Rovio project are robot localization, image filtering, image segmentation, feature detection, moving object recognition, distance computation, obstacle avoidance, to name a few.

Our Rovio project utilized some of known algorithms combined with our own implementation. Through extensive tuning of parameters, Rovio can effectively learn and avoid obstacles and move around following the waypoint image on both carpets and glossy surface. Navigation was based mostly on comparing the waypoint image and image from Rovio through SURF (Speeded Up Robust Features). Real-time obstacle learning was crucial in that many types of floor tends to have tiles which obvious is not an obstacle.

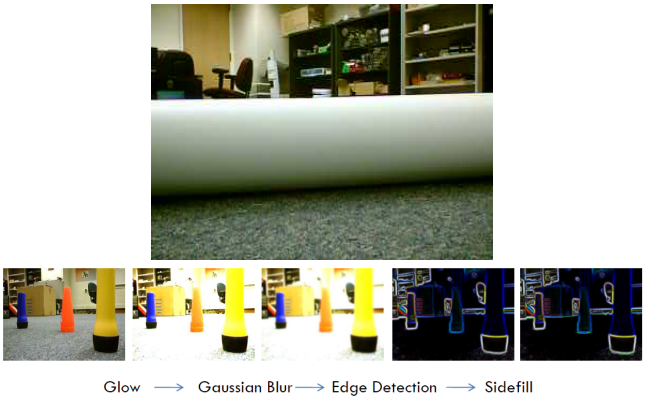
II. Robot

Rovio is a three-wheel robot that is able to move forward, backward, sideways and turn any degree. Video streaming resolution is set to 320 x 240. Rovio also has a height-adjustable camera that can be positioned at low, middle, or high level. It is also equipped with a mic/speaker and an IR sensor.

III. Our Approach

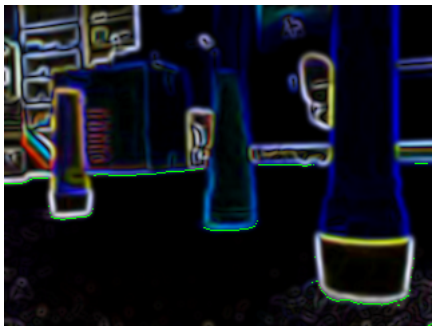
A. Filters

The Java Image Editor project from JH Labs provides a number of image filters, and some of them were used in the previous step of edge detection. Gaussian Filter was used to reduce the noise of objects and the floor, which is a mixture of dots with similar but not identical colors. Glow Filter was used to resolve non-sharp edges, such as the shadow of rolled white paper, which is shown as a gradient from white to black.



B. SideFill Algorithm

After all the filters are applied, we have the edge detected image. This algorithm is necessary for computer to understand where the obstacle is located. For each column of pixels, we sweep from the bottom and find the first edge within that column. The green lines in the image show the closest obstacles in every column of pixels.

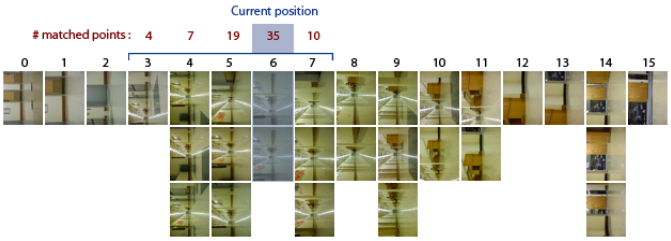


C. Navigation

We use SURF (Speeded Up Robust Features) as our feature detection algorithm. Given an image, SURF finds a set of interested points for that image and compares it to a set of interested points of another image. We did an experiment to find a threshold for similarity and analyze the matched points to determine the transformation. Once we know how one image is transformed (translated, rotated, or scaled) into another, we use that as a guide for Rovio e.g. how close it is to the actual waypoint image, or deciding whether Rovio is moving straight forward.

D. Waypoint

Our waypoint is a sequence of multi-layer images that are taken every 3-6 feet from the starting point to the destination. In some area, we have to take multiple pictures of the same place from different angles for feature detection because SURF algorithm does not perform well with pictures under perspective transformation. To determine the current location of Rovio, we take a window of size 5-9 that spans across the waypoint, then find the image that most matches the picture from the camera and assign that as the new current position.



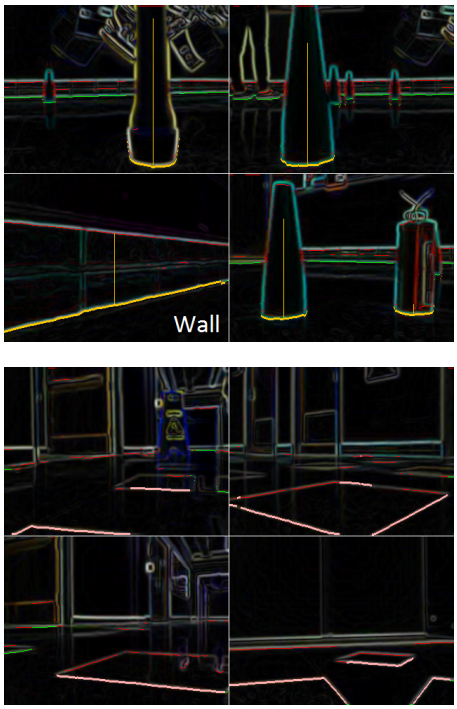
E. Guide

When Rovio localizes itself on the waypoint, we are able to provide some guides for Rovio. A guide associated with each point on the waypoint includes how far apart one waypoint is from the next one, how much to turn, and which direction Rovio should go if it gets lost and doesnt find any matched points. In real situation, we only use guide in some location that SURF gives too few interested points.

F. Real-time Obstacle Learning

We implemented real-time obstacle learning algorithm that learns obstacles as Rovio finds its way to the goal location. We do not give any pre-trained values. It automatically learns obstacles when it is possible and use those learned values to identify obstacles. Such real-time obstacle learning was crucial in that we often found fake obstacles such as tiles on the floor.

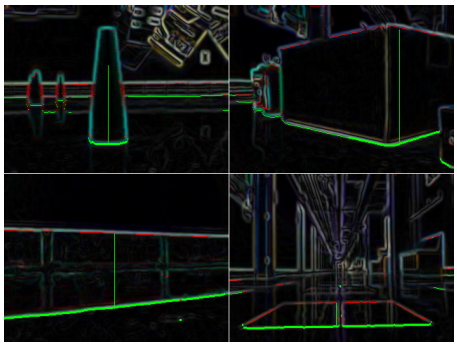
1) Distinguishing Obstacles: To distinguish between obstacles, we used color, texture, and size (either length or height) of obstacle. For color and texture of an obstacle, average of the color and variance of the color were used respectively. By approximate size of an obstacle, it means we just took an average of the height/size of an obstacle. The height of an obstacle was used when it is a real obstacle, and the length of an obstacle was used when it is a fake obstacle.



2) Learning: Rovio has IR sensor that only gives true/false when object is within about 8 inches of an obstacle. And it only detects when obstacle is located relatively right in the middle of an image. Any obstacle that is off to the side from the center of an image has chance of not being detected even though they are close to Rovio. And if there was any combination of a real obstacle and fake obstacle, IR sensor will say true which will be the false positive for the fake obstacle. To prevent such false positive and to satisfy all the restrictions of IR sensor, our learning algorithm trains the obstacle when there is only one obstacle in front that is located relatively right in the middle of an image.

It takes following step for real-time learning. It first finds each obstacle and retrieves the characteristics of each obstacle: color, texture, height and length. We first retrieve both height and length because he does not yet know whether it is a real obstacle or not. It checks whether each of them were trained before by comparing with data we have (more detail in section 4 of this subsection). If they were not trained before, it checks whether there is only one obstacle or many obstacle. If there is only one obstacle and if it located relatively at the middle of the image, Rovio will go forward about a foot until it is detected by IR. We go forward because IR sensor only supports distance up to about a foot. This is safe way to train it because if it is a fake obstacle, it can just go forward without any problem. If it is a real obstacle and is detected by IR, it will stop as soon as IR says ”true” and move backward by the amount it came forward. After we have figured out whether the obstacle we saw was real obstacle or not, we store the characteristic we retrieved in our array of obstacle. When we first retrieved characteristic of obstacle, we calculated both height and length. Now, it only saves one of them: height if it is a real obstacle, length if it is a fake obstacle. Also, along with characteristics of

obstacle, we store the indicator that says whether it is an actual obstacle for comparing purpose in the future.



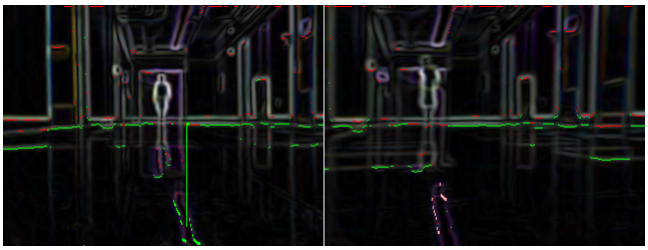
(image: Thick green line indicates obstacle being trained. Vertical green line indicate approximate height of an obstacle)

3) Finding Obstacles: Using the SideFill applied image, it needs to figure out how many obstacles are close to our trapezoid area. It finds the pixel column number of obstacle in image that indicates start and end of an obstacle. In order to find the starting column number of an obstacle, we sweep from left to right and find an edge that's close to our trapezoid. Once I find the start of an obstacle, we continue until there's sudden change of the edge which will indicate the end of an obstacle. Once it finds the start and end of each obstacle, it already knows whether condition for training is satisfied. And for each obstacle, we also need the top edge in order to compute height/length of an obstacle. In order to find the top edge of an obstacle, I apply SideFill algorithm on just that obstacle sweeping from bottom of an obstacle until it reaches top of an obstacle. More details on height and length computation to follow in the next section.

4) Distinguishing Already Learned Obstacles: Before Rovio tries to learn or avoid one or more obstacles, it first extracts all the characteristics described earlier and compares with set of already trained data of obstacles. If it matches with any of them with certain threshold we set, we mark it as either real obstacle or fake obstacle according to trained data. We marked fake obstacles with pink line and indicated real obstacles with yellow line. The yellow vertical line indicates the approximate height it measure to compare with set of trained data. And whenever we have multiple obstacles that are not trained yet or one obstacle that is off to the side of an image, we will assume it as a real obstacle until trained. Using the data we trained, the obstacle will be detected as long as it is not seen from completely different angle. If it is seen from different angle, it will train that side of the obstacle.

G. Challenges on Glossy Surface

With the algorithm described so far, it worked fairly well on carpets. However, the glossy surface in the hallway of Upson Hall, there were many challenges. First, the light reflected on the floor causes it to look like an white obstacle. This problem was automatically solved when it trained itself that light was just an fake obstacle.



(Left: Training light, Right:Pink indicates it is an fake obstacle)

There was real big challenge when the tile got separated by light. It was big issue because our training algorithm does not train when there is two obstacle in front. So, when there are multiple obstacles, after we extract characteristic of each obstacle, we compare them and see how closely they are located. If they are closely located and have similar properties, we merged two obstacles into one big obstacle. With such algorithm, it was able to train on tile.

(Left: Tile separated by light, Right:Thick green line indicates one obstacle being trained)

H. Calibration of Distance & Height

Throughout the experiment, we used 320 * 240 pictures. Rovio has a V-shape camera view angle. The angle between two edges of V are 52. In this section, actual x-distance and actual y-distance is defined as the value computed respect to Rovio's camera located at (0,0) facing north. The y-axis lies toward the north from Rovio's viewpoint, the x-axis lies from east to west of Rovio. Unit of actual distance is in centimeters.

1) Actual Location of a Certain Pixel: When identical objects were located in different places, the y-coordinate of the object's bottom in image increases as the actual y-distance of the object from Rovio increases. The actual x-distance is independent from object's y-coordinate in image but is proportional to the max actual x-distance Rovio can see in that object's y-coordinate (remember, that Rovio has V-shape sight). The result of observation was that as the y-coordinate in picture increases linearly, the actual y-distance increases quadratically, and the x-distance

If (a,b) is the x,y-coordinate of pixel in the image, then the relation is

actual y-distance = $0.0062b^2 + 0.0266b + 26.292$

actual x-distance = $(x - 160) * a * tan(26)/160$

This equation is effective 0(pixel) $\leq b \leq$80(pixel), where corresponding a is around 26-65 (cm). Equation for larger b was not tested. The equation was obtained from regression to quadratic polynomial, and the correlation R^2 was 0.99674, which is accurate enough.

2) Length calibration: Once we know the actual location of pixels in respect to location of Rovio, we can calculate the distance from Rovio to start of an fake obstacle and far end of the fake obstacle. And the difference would be the length of an obstacle.

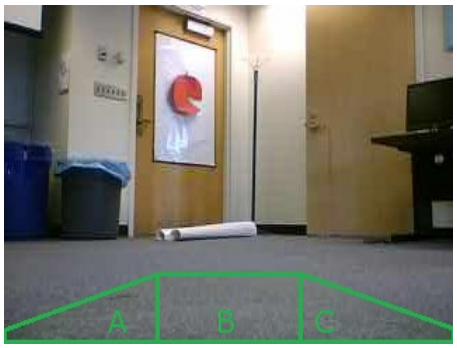
3) Height calibration: When identical objects were seen in different places, the height of them in picture decreases as the actual y-distance from Rovio increases, and was independent from actual x-distance. If 'b' is the actual y-distance from Rovio's camera in centimeters and 'h' is the height in picture in pixel, then the relation is

actual height = $h/(0.0036b^2 - 0.5b + 22.613)$

The data was obtained from 16 different measurements with two obstacles of different height, and regressed into a quadratic polynomial. The correlation R^2 had very high value of 0.99757. Error bound for test points was around 0.5cm.

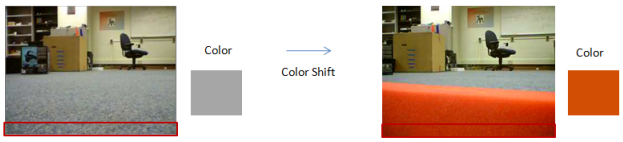
This equation is effective for $26(cm) \leq b \leq 90(cm)$. If a is less than 26, then Rovio's camera cannot detect the bottom of the object, and no testing was done for a larger than 90cm.

I. Trapezoid "Safe" Area



Based on obstacle learning, we know whether they are real or fake obstacles. Whenever we are moving around, we should not be hitting any obstacles. This trapezoid helps Rovio to understand whether it could go forward or not. Using the output of the SideFill algorithm, if there is any obstacle within trapezoid we defined, we turn around. The side it turns and the angle it turns is determined by where the obstacle is located within the trapezoid. If there are obstacles within region A or C, it would make small right or left turn respectively. However, if there are obstacles within B region, it would first whether obstacles are leaned more towards left hand side or the right hand side, and Rovio would make either bigger left or right turn accordingly.

J. Close-Object Detection(Floor Learning)



For close-object recognition, it might seem that we could have just used IR sensor to avoid such close obstacle. However, through many testing, we noticed that Rovio's very unstable IR sensor sometimes detects objects very far away. If it detects far-away object as obstacle, it causes Rovio to turn when there is really no obstacle in-front. Therefore, some learning was required to learn the floor data.

Floor learning algorithm is used to detect objects that are too close to Rovio and fail to be captured by Edge Detection algorithm. This problem occurs when Rovio interprets close-objects as the ground due to the edge lying above the trapezoid area. To avoid this problem, at every iteration, we check whether there is any edge within trapezoid area and also check IR sensor says false. If both conditions are satisfied, as we did in obstacle learning, we retrieve the color and texture of the bottom few rows of images, and store it separately as floor data. Later, if there are no edges within trapezoid "safe" zone, we always check the color and texture of the same lower part of the image from Rovio. If it is significantly different from the floor color and texture we stored, Rovio concludes that there is an obstacle right in front of it, and turns around to avoid hitting it. At each iteration, Rovio tries to retrieve the floor color as long as there are no obstacles, so it is able to adapt to the change on floor.

IV. Experiment Result

`http://www.youtube.com/watch?v=-lqu_kAOGrg`
`http://www.youtube.com/watch?v=LbBWiuL9fGU`

- a full video of Rovio on both carpet and glossy surface
part 1: Obstacle Avoidance in Robot Lab
part 2: Obstacle Learning in Robot Lab (disabled Rovio movement for showing purposes)
part 3: Obstacle Learning in Upson Hallway
- shows it can actually distinguish between real and fake obstacles
part 4: Navigation using SURF in CSUG Lab
part 5: Navigation + Obstacle Learning/Avoidance on bridge between Philips and Upson

A. Experiment with Rovio movement control

The irregularity of the floor and inaccuracy of the wheel motor makes Rovio difficult to move exactly as planned. We experimented with Rovio's movement under different speeds and directions of moving and turning. In the initial phase of this experiment, we suffered Rovio's crude turning and abrupt advance in case it detects an obstacle in front and caused it to hit another obstacle. Now Rovio is more cautious than before and does not carelessly bump into anything around. However, we still find the problem with the turning angle which is greatly affected by the battery level of Rovio and is also dependent on which Rovio we use.

B. Experiment with image processing

The Java Image Editor developed by the JH Labs has many interesting filters in it, in which some are appropriate for edge detection and some are not. We tried different combinations of filters which works best under the decent amount of light. Optimal result came out when we applied glow filter, Gaussian blur filter, and edge detection in order. We have tested under the darker room but it was detecting shadows

and carpets as the obstacles. We wanted to use segmented image rather than edge detected image because we believed that segmented image just would work better. But we were not successful in finding decent Java segmentation code and was not really able to connect Java with c++ based segmentation due to image file type support of Java.

C. Experiment with SURF Navigation

We did an experiment to find the best threshold to determine whether two pictures are of the same area. We first tried a ratio threshold which considers two images to be the same if the number of matched points divided by the number of interested points is above the threshold. However, this did not work well for pictures with too few interested points. So we tried an absolute threshold and used 5 matched points which gives the best result according to our experiment. The absolute threshold still also causes some problems, so we add another mechanism to allow Rovio to localize itself in both backward in forward direction on the waypoint and estimated its current position by the most matched picture based on the number of matched points, the number of interested points, and the ratio. When tested on Rovio, this method seems to give better localization and Rovio lost track of waypoint in less than 10

D. Experiment with Obstacles

As described earlier, we were successful in doing obstacle learning and avoidance of many types of obstacles just based on image processing. The only problem was such objects such as leg of chair which actually is very close to Rovio but seems to be located very far to single camera based Rovio.

There was another problem with learning which was caused by instability of IR sensor. IR sensor sometimes wouldn't detect one-inch tall obstacles. In that case, Rovio would just run over it. Even though it was real obstacle, Rovio was able to climb over and pass-by the obstacle which is not a big of problem.

And as described earlier, though there were lots of challenges, obstacle learning and avoidance works well on both carpet and glossy surface of the hallway.

E. Experiment with WIFI Access Points

When Rovio changes AP, it freezes for a while. This especially causes a problem when Rovio is around the border between two AP's. It keeps changing the access points and therefore is frozen for a long time. To prevent it, a laptop was made as a WIFI point itself, and someone follows Rovio with it.

F. Experiment with Obstacle Size Calibration

Two very long and straight rods were used to measure the viewing angle of Rovio's camera. They located in front of Rovio and were adjusted so that they are just out of bounds from Rovio's sight, parallel to the vertical edges of pictures. As a result, the two rods positioned as a big V-shape with angle of 52, and any point inside could be seen by Rovio's camera.

To relate the coordinate of a pixel from the camera and its actual location, an object was placed in 11 different distances straightly in front of Rovio (the middle x-coordinate in picture). Pictures were taken independently for 1each datapoint, and the plot of y-coordinate of picture vs actual distance were drawn. It was found out that they have quadratic relation with y-coordinate, and linear relation with x-coordinate. The error bound was about 3mm.

Similar approach was used for height measurement. Objects with different heights were placed at different points (and those points were the same for all objects), and their heights in pixel were recorded. The relation between actual distance and the change in height in picture was measured and recorded.

G. Experiment with Battery Life

Rovio has a very short bettery life, and is often not charged very well. It was common for a Rovio (dramatically different from each Rovio) to run out of battery and stop in the middle. We tried to minimize the number of HTTP requests to Rovio to use less battery.

V. Conclusion

Rovio navigates reasonably well based on vision waypoint and is able to avoid various types of obstacles on both carpets and glossy surface as shown in the video. Rovio barely ever hits an obstacle as long as Rovio is fully charged and IR sensor returns correct value. When Rovio is not fully charged, they tend to make smaller turn which makes Rovio to sometimes hit obstacles by the side-wheel slightly. But even though it is not charged fully, it navigates without too much of a problem. Best part was that it was able to distinguish between real obstacles and tiles on the floor. It was able to navigate anywhere in Upson Hallway even though it had lots different colors and sets of tiles. One of the longest drive without hitting we were able to see was from Robot Lab to another Robot Lab (Prof. Saxena's Lab) and almost all the way back to the location where we started. Unfortunately, we were not able to take video due to problem where WIFI Access Point caused problem or Rovio's battery would die out before traveling farther.

Navigation using SURF also worked well given that the images on the waypoint have sufficient number of interested points. If the image has too few features, Rovio sometimes

goes too far forward and could not get back on the track. Another problem we had was when SURF and obstacle learning/avoidance was combined together. It worked fairly well as shown in video, but when there were too many obstacles, obstacle learning/avoidance algorithm would sometimes lead Rovio out of the path which causes SURF to lose the image it was searching for. If we had some more planning applied, it would have worked better.

Except the times when too many obstacles causes Rovio to be out of path, Rovio generally does not get stuck because we have a controller that prevents Rovio from making repeated moves, for example, it would not make a right turn if it made a left turn as the last move.

REFERENCES

[1] Obstacle Avoidance. Robo Realm. 20 Feb. 2010. http://www.roborealm.com/tutorial/Obstacle_Avoidance/slide010.php

[2] JH Labs for Java Image Editor project http://www.jhlabs.com

[3] Apache Commons Math http://commons.apache.org/math/

[4] JOpenSURF http://www.stromberglabs.com/jopensurf/

[5] 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

[6] Rovio User Manual http://www.wowwee.com/static/support/rovio/manuals/Rovio_Manual.pdf

[7] Rovio API Document http://www.wowwee.com/static/support/rovio/manuals/Rovio_API_Specifications_v1.2.pdf