APPLICATIONS OF DSP & COMPUTING VISION COMPUTER VISION GROUP PROJECT : SENSOR ROBOTS FOR ENSURING POST-INCIDENT MINING SAFETY

Suzanne Bizot, Alejo Martin, Cyril Limam, Hubert Lacote, Pierre-Edouard Iung

Cranfield University Students in Msc Computational and Software Techniques in Engineering
Option : Digital Signal & Image Processing

ABSTRACT

This paper is about computer vision with one camera embedded robot rovio manufactured by WowWee. We use image processing and analysis to solve several modern technical issues, such as autonomous robot navigation in an unknown environment, obstacle detecting, obstacle avoidance, but also map editing of an unknown environment. The final purpose of this project is to read physical sensors (our target) to extract information such as gas level in a mine.

Index Terms— Computer vision, obstacle detecting and avoidance using image processing & analysis, autonomous navigation in unknown environment, target tracking and reading.

1. INTRODUCTION

Nowadays, mine safety is becoming a big issue. The rescue of Chile miners in august 2010 took 69 days, hopefully they all survived. But 33 minors died in New-Zealand coal mine after a blast in November 2010. Such events, sadly too frequent, make people realize how unsafe was the mines. The mean idea of our project is to provide an autonomous agile robot rovio, capable of moving by itself in a mine. The robot is also able of detecting any obstacle and to edit a map of those obstacles. Then the robot searches for gas level sensors and read them, then reports. With this innovative solution, it may be safer - in the future - to rescue people in the mines.

2. LITERATURE REVIEW

A lot of researches have been done in vision-based navigation for indoor robots. Depending on the requirement of the navigation, several techniques can be applied. The review of G. N. DeSouza and A. C. Kak [1] presents three types of navigation: “Map-Based Navigation”, “Map-Building-Based Navigation” and “Mapless Navigation”. The first group cannot be applied in this research as the robot is not supposed to have a prior knowledge of the environment. The two others categories are of interest, one enabling to perform a navigation based on an “Obstacle avoidance behavior” while the other can allow to develop a more reliable navigation by creating a map of the environment.

Regarding the “Mapless Navigation”, Santos-Victor et al. have focused on “Navigation using Optical Flow” based on the visual behavior of bees. [2] Their idea is that the two eyes of a honeybee cannot provide sufficient depth information due to the small distance of the two eyes. They explain than a bee can maintain itself in the middle of a corridor using “the difference between the velocity information computed from the left and the right eye”. This behavior can be applied for a robot: “move to the left if the velocity measured by the right eye is larger than that measured by the left (or vice versa)”. Maksym Usov in his MSc report [3] successfully applied this technique to create an obstacle avoidance behavior for a robot that has a single camera. It is able to follow a corridor, to avoid obstacles and to detect front collisions. The major drawback of this technique is that you need to be able to compute optical flow. It will therefore not be applicable when no “distinguishable features” can be found [3], i.e. when the obstacles are not sufficiently textured. Another drawback is that this technique is applicable only when the robot drives forward and that the “rotational speed is not larger than a certain factor times the translational speed” [1].

A number of papers deal with “Map-Building-Based Navigation”. It can be seen in the survey of G. N. DeSouza and A. C. Kak [1] that this is often done using specific material or specific calibration procedures. Stereovision is for instance achieved in [4] using trinocular vision.

The work of Jeffrey Buente et al. [5] presents an implementation of the Simultaneous Localization and Map Building (SLAM) with the Rovio robot manufactured by WowWee. They use the specificity of the Rovio robot to create a simple stereo vision algorithm that can gives precise results without needing to deal with “complex ways of […] matching edges between images”. They use an occupancy grid, proposed by Moravec and Elfes [6] to map the environment. Their work enabled them to map accurately an indoor environment with “an average of 18.56 cm error”. Their technique has several drawbacks: the floor needs to be sufficiently uniform because of the simplification of the Stereovision. This can be addressed improving the way to determine “what constitute a unique distinct edge”.

Figure 1: middle image used to detect the first obstacle in every column

Figure 2: down image with the first obstacle detected

Figure 3: output of the Canny edge detector on the middle image

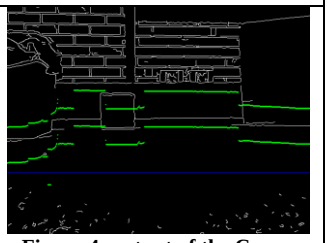

Figure 4: output of the Canny edge detector on the down image

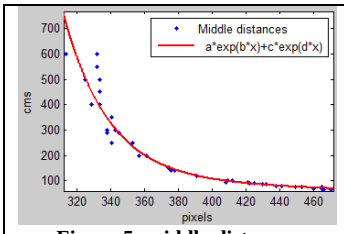

Figure 5: middle distances

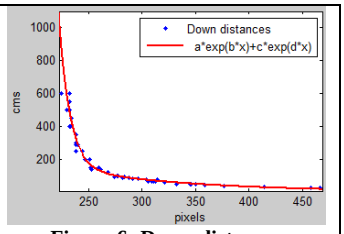

Figure 6: Down distances

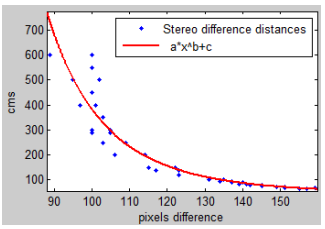

Figure 7: stereo difference distances

Figure 8: distance of obstacle

In their demonstration, the robot maps only the environment from two positions. Therefore, if more movements are involved, the accuracy could really become worst.

Their work is still of great interest for us as it can provide a measure of the distance to the obstacles. This can be used to make a simple obstacle avoidance algorithm that can enable the robot to move safely in an unknown environment and it can also be used to map the environment, keeping in mind that the information extracted should be used with care.

3. NAVIGATION STRATEGY

3.1. General Navigation strategy

At the beginning, the robot does a 360° turn in order to map everything around it.

Then in order to find all the sensors, the robot has to explore as much as possible all the part of the mine. Each time the robot go to a new position all the points in its field of view with a distance less than 70 cm are considered as "visited". If the robot hasn't detected any sensor in these points, it means that there is no sensor and we don't have to go there again. When the robot arrives in a new position it calls the sensor detection function (cf part 4.1). If the function returns several non-read rectangles, we take the one which has more feature points. Then the image is split in 3 parts. If the rectangle is in the left(respectively right) part the robot turns left(respectively right) and then the robot goes straight forward if it cans. If the rectangle is in the center of the image the robot tries to go straight forward. If it can't, then the image is split in 2 parts. If the rectangle is in the left(respectively right), the robot turns left(respectively right) then goes straight forward if it can and finally turn right(respectively left).

After several tests we estimated that each time the rotation function is called the robot turns of 23 degrees. We also estimated that when the straight forward function is called the robot does 15cm.

If no sensors are detected the robot goes to the closest non explored point accessible. If there is no unexplored accessible point, the robot goes randomly in an accessible position. Finally, after moving, the robot updates the map with what is in front of him.

3.2. Obstacle distancing

To extract the distance to the obstacles in front of the robot, we used the method developed by Jeffrey Buente et al. [5]. Assuming the floor is sufficiently uniform greatly simplifies the depth recovery from the stereovision. The aim of this work is to provide a prototype and therefore, this part should be refined for a more robust application.

The specificity of the Rovio robot is that it can raise its camera in three positions that are fixed. For the stereovision, we only use the down and the middle position of the camera. This method uses three measurements extracted from every column of the two images to detect the distance to an obstacle. The first measurement is the difference of the position of the first edge detected in the floor in the two images. The second one is the position of the first edge detected in the floor in the middle image (see Figure 1) while the third one is the one detected in the down image (see Figure 2). To detect the edges, we first apply the Canny edge detector on the two images (see Figure 3 and Figure 4). We can use the middle image from the bottom as there is no noise (see Figure 3) whereas we must start from a fixed row that has been manually tuned to avoid the noise in the down image (see Figure 4). The green lines in the Figure 4

represent the interval in which the edges found in the middle images must lie in the down image.
As stated in the paper, we calibrated the distance output using a set of stereo images with known distances to the obstacles. We then used a built-in Matlab regression algorithm to fit our data points (see Figure 5, Figure 6 and Figure 7).

For every column, we have real distance measurements that we combine together using the same method as in [5]. We discard invalid values, we use the mean of all the valid distances except when the robot is too close to an obstacle where we only use the down distances. We then take the median distances to discard invalid values for a given number of intervals. We can then obtain a distance representation to the obstacles (see Figure 8)

3.3. Map editing

Defined as "the branch of the robotics which deals with the study and application of ability to construct map or floor plan of the environment by the autonomous robot and to localize itself in it", in our case, considering that the robot it's not particularly quick, we determined that it was necessary to create a floor plan representing the reality around the robot in order to not repeat unnecessary readings and keep a record of the area already explored. For this goal, first we had to decide what is worth to store and what was going to be the use of the map for the functionality of our robot.

For the first decision we realize that storing lots of the information in the map was not a good decision due to the fact that the accuracy of the mapping strongly depends on the accuracy of the robot localization. This means that in our case of study, as the robot has some localization random error with each forward movement due to his mechanics the map is going to have some imprecisions. So we decided to store only two pieces of information for each point of the map, the estimation of the objects position with an associated reliability and the sensor exploration state.

Once we decided what to store, we studied the use of the map for the navigation. As our robot tries to find sensors in the area, we implemented the ability to give advice to the navigation function, about in which direction is more likely to find a sensor. The functionality of knowing previous sensor maps was proven to be useful for avoiding reading them several times. Another interesting functionality was the estimation of the possibility to make a straight forward movement based on the objects already mapped. For future works it could be interesting to implement a dynamic size map and a path planning algorithm to go to a specific point in the map, which is not reasonable at this moment due to the robot localizing errors added to our mechanical issues.

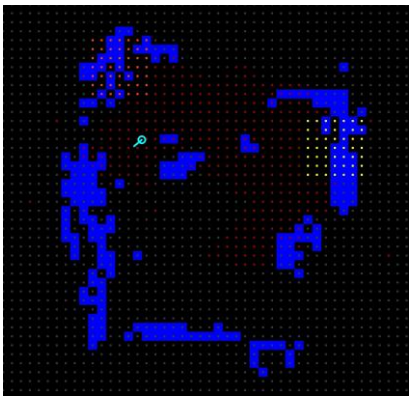

Figure 9 : Map of obstacle and sensors

4. THE SENSORS

4.1. Detecting

4.1.1. Previous work

To detect an object there are many different methods, we thought at the beginning about using a template matching as it is describe in the article [7]. But the problem is that we would only have detected sensors at a fixed distance, indeed the template has a certain size so it matches with the scene image when the robot is at a fixed distance from the sensor. We could have tried this method but we also wanted a method that can detect the sensor even if it is partially hidden. So we thought about the features detection and matching. We decide to use the SURF algorithm to extract the points of interest (it's the most scale/rotation invariant method).
We found some explanations about the SURF algorithm in the course lecture notes of Computer Vision of Toby Breckon [8] and in some examples of opencv function using SURF in the Opencv2.1 and Opencv 2.2 folder (findobject.c,matcher_simple.cpp,matching_to_many_images.cpp).

Our work can be separate into two steps.

4.1.2. Our First approach

We use the SURF algorithm to extract the points of interest of the images of the scene (query image) and of the sensor (training image). Then we try to find some matched pair between the points of interest of the scene and the sensor. Two points are matched if their SURF descriptors are similar (ie the distance between them is lower than a certain threshold). Then we define a rectangle in the image where most of the points are located. We also try to calculate the homography matrix which will enable us to find the position of the sensor in the scene image (we only use this information as a proof of good detection). In order not to have false detection, we add a test to verify if there are some

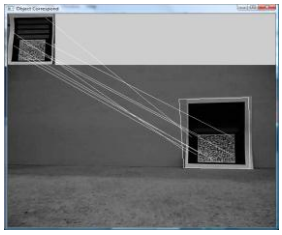

Figure 10 : Matching sensor keypoints and image keypoints

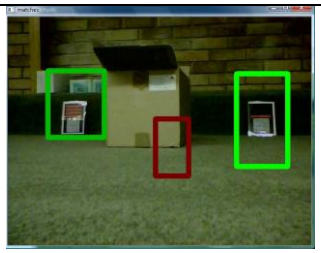

Figure 11 : output of the sensor detector

red pixels in the image. If there aren't red pixels we consider that there is no sensor in the image.

The final outputs are a rectangle and an integer. The integer represents the probability that there is a sensor inside this rectangle. This probability is null when there are no red pixels within the image and depends on the number of features points inside the rectangle.

This method allows us to detect a sensor when the robot is in front of the sensor and that the distance between the robot and the sensor is less than 1 meter.

4.1.3. Our Second approach

In order to improve the results, we use some functions of opencv2.2 which enable us to extract features from several sensors image. The matching is then done with all these features. We also change the output of the function. The function draw rectangles around all cluster of points and then returns all these rectangles with a probability of sensor presence for each one. This enables the detection of two sensors within an image. (cf figure12).

Like previously, the probability is null when there is no red pixels within the image and depends on the number of features points inside the rectangle. To avoid false detection when the sensor is too far, we decide to take into account only the feature points that are close to the sensor (less than 1,50m).

We call the reading function when the robot is at 75 cm maximum of the sensor and after checking on the map that we haven't read the sensor yet.

	Good detection (1st approach)	Good detection (2nd approach)
Close sensors	71,43%	100%
Far sensors (> 1 meter)	0%	100%
Fuzzy image	0%	80%
All kinds	34,5%	96,6%

Figure 12 : Result for detecting sensor (29 pictures test)

4.2. Reading

Once the detecting part of our project has return a rectangle – region of interest of our current picture- mixed with probability of having a sensor in this rectangle, we can read the sensor. First, we find red pixels of the picture. We use the square distance between each pixel of the rectangle and a red pixel. In RGB we have red pixel = (255,0,0), let call P the current pixel, (r,g,b) the three color channel red, green, blue. The pixel values are (P.r, P.g, P.b). Then the quadratic distance between the pixel P and the color red is equal to d.

$$d = \sqrt{(P.r - 255)^2 + (P.g - 0)^2 + (P.b - 0)^2}$$

The second part of reading is to count the contours of the red stripes (from the sensor. To find the most interesting contours, we put a threshold on the size of the stripes areas.

Before optimization, the efficiency of this reader was 74%. The main error factor was blurred pictures, and the fact that the sensor was too far to be read. Now, with a good reading distance (between 20cm and 1,5m) the reader efficiency has grown to 97%. To avoid noise we took picture only when the robot is not moving, and we use the obstacle avoidance part of our project to determine how far the sensor is, therefore we usually have good reading when it is possible to read the sensor.

5. CONCLUSIONS AND FURTHER WORK

The robot moving precision measurement are quite poor, so more the robot moves in an unknown environment and more error we get on positioning it. The measurement of the moving distance is not so efficient (few cm of error usually) also. But all the object avoidance is quite strong as the sensor reading. Further Works may help us to improve the precision of the robot movement and our real time map editor would be very powerful.

6. REFERENCES

[1] Guilherme N. DeSouza and Avinash C. Kak, "Vision for Mobile Robot Navigation: A Survey," *IEEE Trans. Pattern analysis and machine intelligence*, vol. 24, no. 2, February 2002.

[2] J. Santos-Victor, G. Sandini, F. Curotto, and S. Garibaldi, "Divergent Stereo for Robot Navigation: Learning from Bees," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition*, 1993.

[3] Maksym Usov, "Vision Based Mobile Robot Navigation," University of Twente, June 2006.

[4] N. Ayache and P.T. Sander, "Artifical Vision for Mobile Robots: Stereo Vision and Multisensory Perception," eds. MIT Press, 1991.

[5] Jeffrey Buente, Rohan Sharma and Daniel Mejía, "Visual SLAM using Rovio", Cornell University, Spring 2010.

[6] H.P. Moravec and A. Elfes, "High Resolution Maps from Wide Angle Sonar," *Proc. IEEE Int'l Conf. Robotics and Automation*, pp. 116-121, 1985.

[7] Patrick Rossler, Sascha A. Stoeter, Paul E. Rybski, Maria Gini, Nikolaos Papanikolopoulos V isual Servoing of a Miniature RobotToward a Marked Target

[8] Toby Breckon, Image Processing (IP), Cranfield University 2010