Vision Based Autonomous Navigation in Unstructured Static Environments for Mobile Ground Robots

D. Novischi*, C. Ilas*, S. Paturca*, M. Ilas**

**Dept. of Electrical Engineering, Politehnica University of Bucharest, Bucharest, Romania (e-mail: dan.novischi@ gmail.com,* constantin.ilas@upb.ro, *sanda.paturca@upb.ro).*
***Dept. of Electronics and Telecommunications, Politehnica University of Bucharest, Bucharest, Romania (e-mail: m.ilas@hlx.ro)*

Abstract: This paper presents an algorithm for real-time vision based autonomous navigation for mobile ground robots in an unstructured static environment. The obstacle detection is based on Canny edge detection and a suite of algorithms for extracting the location of all obstacles in robot's current view. In order to avoid obstacles we designed a reasoning process that successively builds an environment representation using the location of the detected obstacles. This environment representation is then used for making optimal decisions on obstacle avoidance.

Keywords: autonomous robot, robot vision, image processing algorithms, unstructured environment, Canny edge detection, agglomerative clustering.

1. INTRODUCTION

Computer vision is a field of computer science that has been heavily researched in the recent years. Its applications in robotics are diverse, ranging from face recognition to autonomous navigation (see Forsyth, Akella, Browning). General object recognition for mobile robots is of primary importance in order to generate a representation of the environment that robots can use for their reasoning processes. The ability to localize obstacles in real-time and with a high degree of accuracy based on vision provides a foundation for the development of many state of the art autonomous robotic systems (see Gopalakrishnan, Allenya, Stronger, Novischi 2009). Directions such as optimal implementation, robust detection, especially when using limited performance cameras and optimal avoidance decisions continue to be of primary interest.

This paper focuses on real-time vision based autonomous optimal navigation for basic mobile robots in an unstructured static environment (see also Zhao). Simply stated, the robot has to travel through an unknown environment, where still objects are randomly placed, in order to reach a specified destination. For this, the robot relies only on visual information for the reasoning process. Furthermore, the camera and the robot hardware and sensing are rather limited in terms of performance (see section 2). We have used for obstacle identification an approach based on Canny edge detection and a suite of algorithms for extracting the location of all obstacles in the robots field of view. We also designed a higher reasoning process that allows the robot to make optimal decisions to avoid collisions in order to reach the specified destination.

In order to ensure maximum flexibility during the algorithm development, we implemented the main image processing tasks in Matlab and interfaced the robot with the Matlab. Thus, the robot acquires the images, sends the image to Matlab, which localizes the obstacles, makes an optimal decision and sends a trajectory update to the robot. The application we designed runs in real-time due to the efficiency of the developed algorithms.

Of course, the algorithms can be then implemented directly on the robot controller and we present a processing time estimation for this situation.

2. HARDWARE & SOFTWARE USED

The hardware used for physically implementing the robotic system includes a Blackfin SRV-1 robot (Fig. 1) and a personal computer. The SRV-1 robot is equipped with a DSP microcontroller running at 500 MHz, a 1.3 MP camera with adjustable image resolution and a Wi-fi hardware module. It also includes four DC brushless motors.

Fig. 1. The Blackfin SRV-1 robot used for implementation and testing.

The application algorithms were implemented in Java and in Matlab. These languages were selected because of the overall performance and flexibility that their combination offers mainly: complex data structures, cross platform and fast matrix operations.

In order to interface the SRV-1 robot with a PC we developed a Wi-fi software communication module. So, from the application standing point a PC can be viewed as the processing unit and the robot can be viewed as the sensing and actuating unit. This module supports requests such as movement commands, setting the image resolution and reading the data from the sensors such as the RGB colour image.

3. APPLICATION ALGORITHMS

3.1 The main Algorithm

The main algorithm of our application is structured according to three tasks that the robot has to perform namely: the obstacle detection task, the decision task and the travel task. These tasks are interleaved in order for the robot to reach its given goal. The main logic of the algorithm starts from the idea of a permanent interaction between the robot and the surrounding environment. In this interaction the robot not only detects obstacles, but it also successively builds a representation of the environment in order to make an optimal decision to avoid certain obstacles. After an obstacle has been avoided the robot repositions its self to reach the given goal. The main algorithm diagram is presented in Fig.3 below.

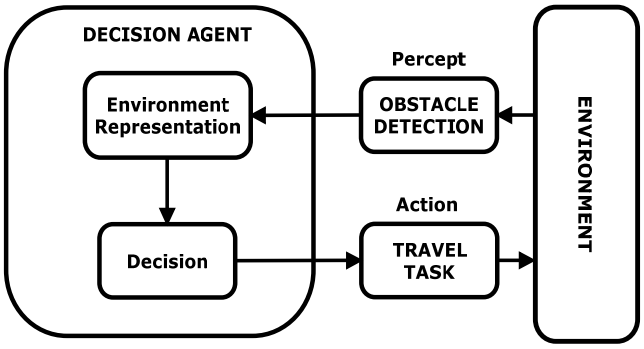

Fig. 3. Main algorithm diagram.

3.2 The Travel Task

The travel task is responsible for driving the robot. This is accomplished by sending movement commands via the Wi-fi communication module. Due to the fact that the SRV-1 robot is not equipped with any odometric sensor such as encoders, the trajectory is not expressed in fixed coordinates, but in adjustable movement segments.

3.3 Obstacle Detection Task

The obstacle detection task is responsible for localizing all obstacles in the current field of view that are on possible collision courses and for signalling this to the decision task. For this purpose the obstacle detection is performed in two separate steps, namely: object feature detection and feature analysis. In the first step the objects edges are detected using a Canny edge detector. This detector was selected because of its higher performance in comparison with others such as: Solbel, Roberts or Laplacian (see Forsyth). After applying the Canny edge detector, we separate the edges of the objects of interest from the edges belonging to the objects in the background by selecting only the bottom part of the image. This is because obstacles that are closer to the robot and influence its travel path are always located in the bottom part of the 2D image plane. The algorithm for the feature detection and its results for every step are presented below.

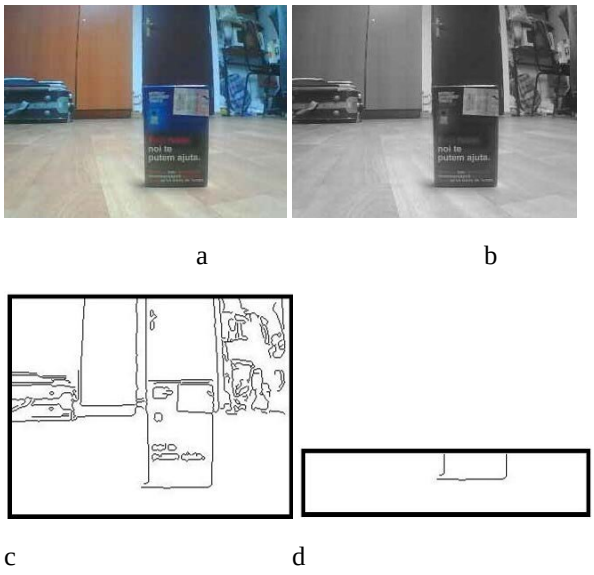

Fig. 4. SRV-1 camera images: a – RGB colour image, b – gray scale image, c – edge image, d – bottom part of edge image.

Objects Feature Detection Algorithm

Get RGB image from the robot

Transform the image to gray scale

Apply the Canny edge detector

Select the bottom part of the image

In the second step we analyze the detected edges in order to compute the location of the obstacles in the 2D image plane. In the ideal case where object edges are perfectly detected, such as the one in Fig. 5, we could compute the objects coordinates in the 2D image plane.

Fig. 5. Ideal case edge detection: left – bottom part of the camera captured image and right – ideal edge image.

Despite the fact that tests were conducted in an office environment (see also Tanaka), the detection of edges is influenced by various factors such as ambient light conditions, shadows and similar overlapped objects. Due to these factors the object edges appear as in Fig. 6.

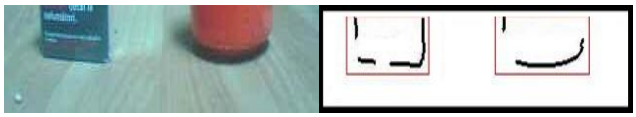

Fig. 6. Real case edge detection: left – bottom part of the camera captured image and right – real edge image.

In order to distinguish which edges belong to which object we developed a modified version of the bottom-up agglomerative clustering algorithm. The elements that are clustered in this algorithm are the individual edge pixels. The merging step of two clusters is based on a threshold rather than on a rule such as single linkage of the standard algorithm. The modified version of the algorithm is presented below:

Agglomerative Clustering Algorithm

Put each edge pixel into its own cluster

Compute the distance matrix

While (! no cluster to merge)

For every cluster i

For every cluster j

If (distance between clusters < = threshold)

Merge clusters

Update distance matrix

The output of this algorithm is separate edge images which contain individual objects. The results for the images in Fig. 6 are shown in Fig. 7 below.

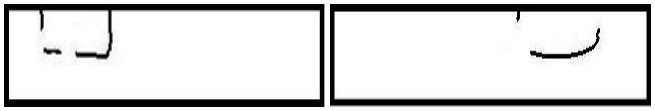

Fig. 7. Agglomerative clustering results.

After the image segmentation into individual obstacles, the coordinates of each obstacle are computed in the 2D image plane. Then we compute the coordinates of a central obstacle which includes all obstacles that are on the direct collision course with the robot. This approach is depicted in the Fig. 8 below. Finally, the central obstacle is merged with some of the obstacles that are not on the direct collision course, but for which the distance between them and the central obstacle is smaller than the robot width and thus, the robot could not pass between these obstacles.

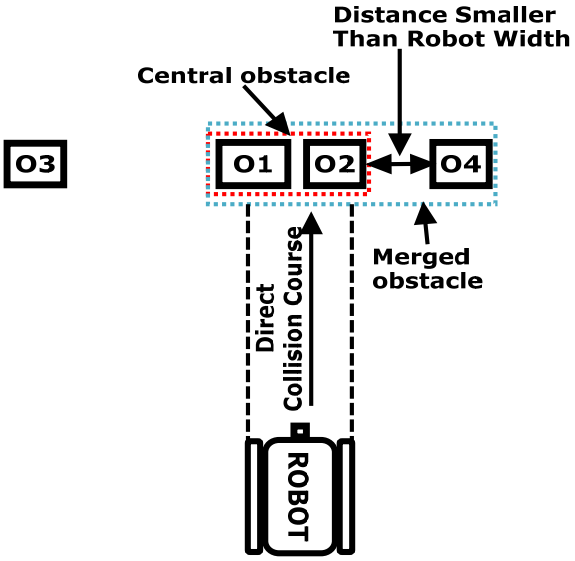

Fig. 8. Obstacle localization diagram – including obstacle merging for close obstacles (O1, O2 and O4).

The algorithm for obtaining the location of the merged obstacle in the 2D image plane is presented below:

Obstacle Localization Algorithm

Segment the image into individual obstacles

Compute each obstacle coordinates

Compute the coordinates of the central obstacle

Merge obstacles

3.3. The Decision Task

The decision task is responsible for successively making optimal decisions on how to avoid obstacles in order to reach a given goal. This means that ideally, the shortest travelling path should be selected. In order to make such decisions when avoiding the obstacles, we maintain and successively update an environment representation in form of a graph. In this graph the nodes represent the possible positions that the robot can reach and the edges weights represent the distances the robot must travel in order to reach these positions. These distances are estimated from the processed image, by counting the number of pixels between given pair of positions. Making an optimal decision as to which successive positions the robot must reach in order to avoid the obstacles is based on Dijkstra's single source shortest path algorithm (see also Dumitrescu 2009). To correctly compute the shortest path, the graph is updated after each obstacle

detection and the start node is set before the computation in question.

When the robot first starts the graph is initialized to contain only two nodes, namely: the start node and the goal node, depicted in figure 9.

Fig. 9. Environment graph for path decision task; Initial graph – V1 is start node and V2 is the goal node.

As the robot travels and detects obstacles on its travel path, the graph is successively updated according to two situations: the detection of a new obstacle and the detection of a secondary obstacle. A new obstacle is the obstacle detected while the robot is moving straight toward the goal. By secondary obstacle, we refer to an obstacle detected while the robot is trying to avoid a previously detected obstacle. In the first situation, the graph is updated with two nodes that are placed between the start node and the goal node. These nodes correspond to the left and right positions that the robot can reach in order to avoid the obstacle edges, as depicted in the Fig. 10 below.

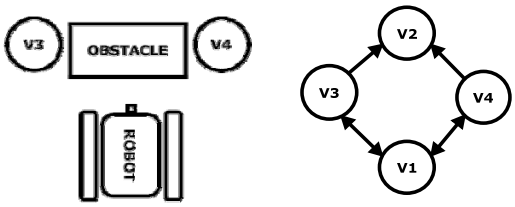

Fig. 10. Found new obstacle graph update: left – the nodes position relative to the obstacle and the robot, right – the updated graph.

The algorithm for this graph update is presented below.

Found New Obstacle Graph Update Algorithm

Estimate distances to avoid the obstacle from 2D image plane obstacle coordinates

Create two nodes

Update source adjacencies and edge weights

Update the new nodes adjacencies and edge weights

In the second situation the robot detects an obstacle while trying to avoid another. The graph representation in this case is updated with only one node corresponding to the current position of the robot. This node is placed between the starting node and the node that the robot was trying to reach. The later node is also updated so that it corresponds to the position for avoiding the intermediary obstacle on the same side as the previous obstacle. This situation is presented in Fig. 11 below, where the intermediary obstacle is detected after the robot tried to avoid the first obstacle on the right side.

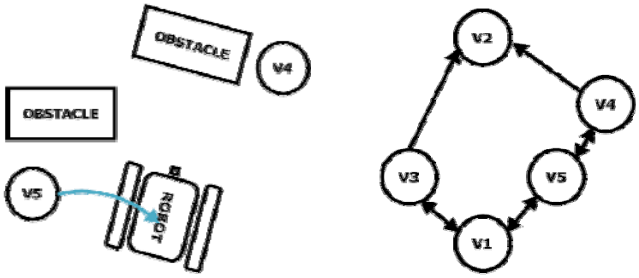

Fig. 11. Found intermediary obstacle graph update: left – the node's position relative to the obstacle and the robot, right – the updated graph.

The algorithm for this graph update is presented below.

Found Intermediary Obstacle Graph Update Algorithm

Estimate distances to avoid the obstacle from 2D image plane obstacle coordinates

Create a new node

Update source adjacencies and edge weights

Update adjacencies and edge weights for the new node

Update adjacencies and edge weights for the node that the robot was trying to reach

4. RESULTS

In the experiments that we conducted the behaviour of the robot was tested in various obstacle configurations. In this section we present two of the environment settings (obstacle configurations) that are representative for the application that we designed. The objects that were used as obstacles in the two settings are: various cups and glasses, small packages and note pads.

In the first setting we placed the obstacles so that the robot would successively detect only new obstacles on the travelling path to the goal object. This setting and the travel path of the robot from the start to the goal is shown in the Fig. 12.

The graph representation of the environment built by the robot for this obstacle course is presented in Fig. 13. It can be seen that the trajectory that corresponds to the smallest distance on the graph matches the trajectory depicted in Fig. 12.

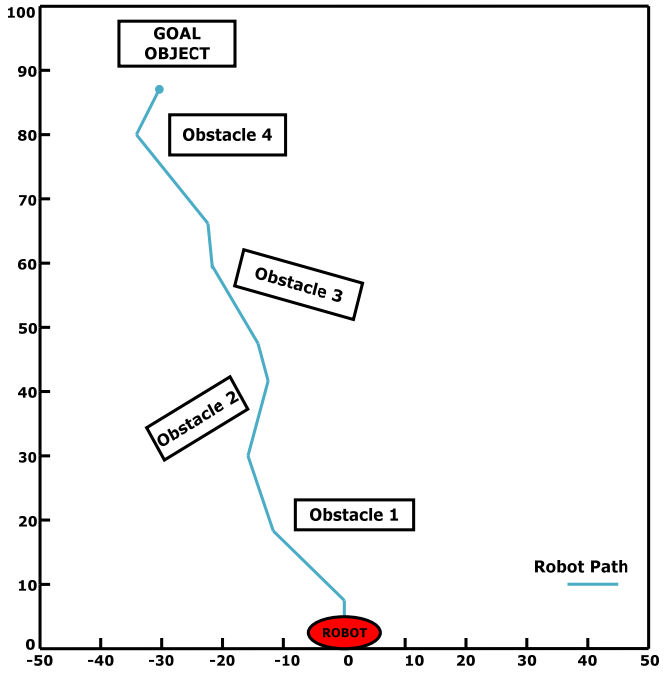

Fig. 12. Obstacle setting 1 and the robot travel path.

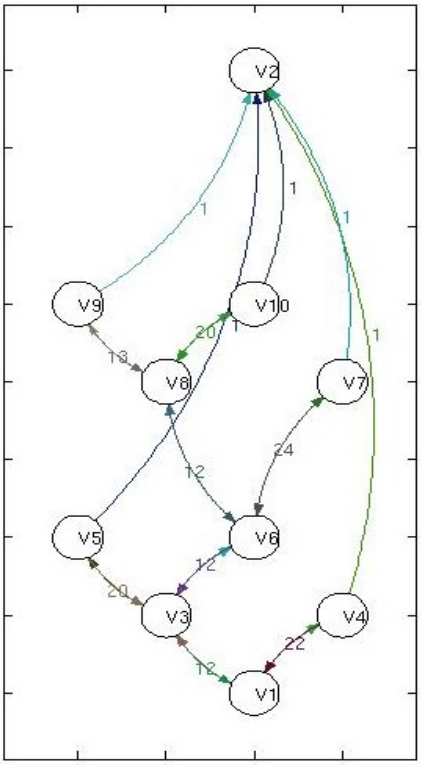

Fig. 13. The graph representation of the environment built by the robot for the path shown in Fig. 12.

In the second setting the objects were placed such that the robot would detect also a secondary obstacle and the distance to avoid that obstacle would be greater than the distance to avoid the first obstacle on the other side. Thus the optimal decision made by the robot in this case is to turn back and avoid the first obstacle on the other side. This can be seen from the Fig. 14 below:

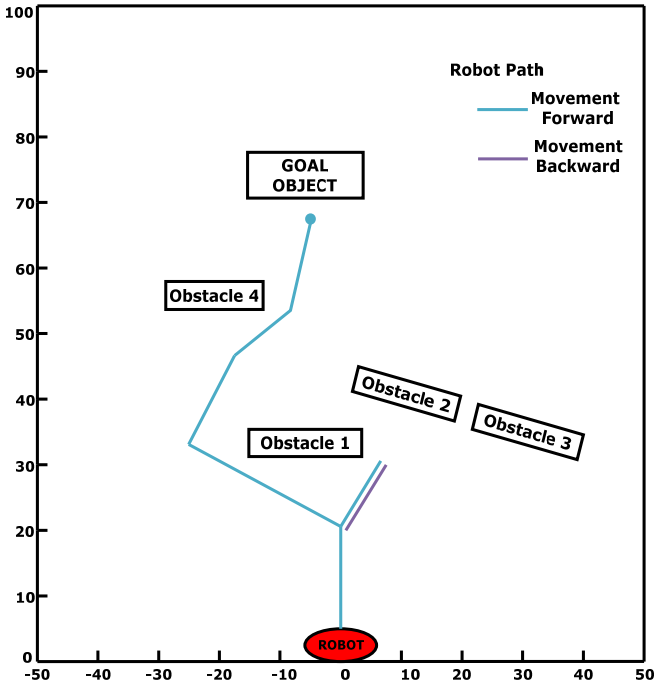

Fig. 14. Obstacle setting 2 and the robot travel path.

The graph representation of the environment built by the robot for this obstacle course is presented in Fig 15.

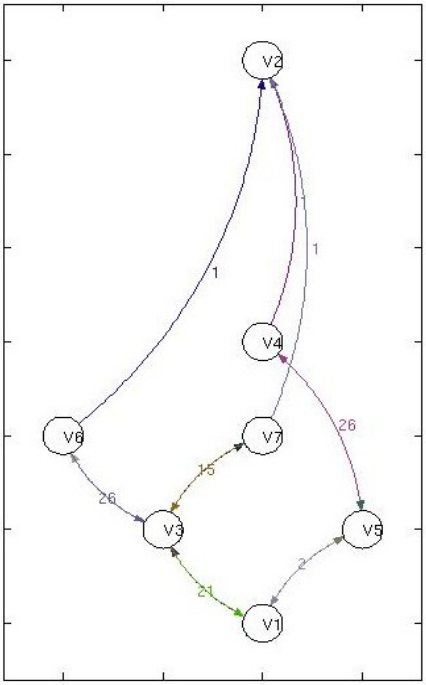

Fig. 15. The graph representation of the environment built by the robot for the path shown in Fig. 14.

The computational complexity of the application that we designed is in principal given by the image transformation to

gray scale and the three core algorithms, namely: the Canny edge detection algorithm, the agglomerative clustering algorithm and the single source shortest path algorithm. The gray scale image transformation and the edge detection are the prime contributors to the application computational overhead, because there computational complexity is *O(mxn)* where *m = 240* and *n = 320* (for an input image of *320x240*). The agglomerative clustering has a computational complexity of $O(n^2)$ where *n* is variable and equals the number of edge pixels. The average number of the edge pixels equals *1000*. Thus, the agglomerative clustering algorithm generates a smaller overhead because the input size is considerably smaller compared to that of the canny edge detection. The single source shortest path algorithm generates the smallest overhead in relation with the application and has a computational complexity of *O(n log n)* The algorithm's input size comprises of number of nodes and the number of edges in the graph at a given moment.

In order for the robot to avoid a certain obstacle, these algorithms are executed in a sequential order. So, the running time of each processing step depends on the sum of machine instructions given by each algorithm computational complexity. This sum, in an average equals approximately *480k* machine instructions. For the SRV-1 robot DSP microcontroller at 500MHz the running time of the algorithm is around *0.96* milliseconds.

For comparison, the round trip exchange of information and processing using Matlab is around *1s*, which still ensures a smooth movement of the robot.

5. CONCLUSIONS

This paper presents a real-time application of a vision based autonomous navigation in unstructured static environments for mobile robots.

The results showed that the robot successfully navigates in unstructured environment making optimal decisions to avoid all the obstacles.

The application was structured according to the tasks the robot has to perform. The main idea was a continuous interaction between the robot and the surrounding environment.

Obstacles on the robot travel path were localized in two steps. In the first step we detected and separated edges of the objects of interest. In the second step we analyzed these edges to determine the individual objects and compute their coordinates. The detection of edges is based on Canny edge detector and the edge analysis is based on a modified bottom-up agglomerative clustering algorithm.

In order to make an optimal decision to avoid a certain obstacle we maintain and successively update a graph-based representation of the environment. The collision avoidance decisions are based on the Dijkstra's single source shortest path algorithm.

The overall obtained results prove that autonomous navigation in unstructured environments is possible with low cost hardware such as the SRV-1 robot. Therefore in the future we expect for many such robotic systems to be used in an unlimited number of applications, thus continuing to have an increasingly impact in people lives.

REFERENCES

Akella, M.R.(2005) Vision-Based Adaptive Tracking Control of Uncertain Robot Manipulators. *Robotics, IEEE Transactions on*, Volume: 21 , Issue: 4. pp: 747 – 753.

Alenya, G.; Escoda, J.; Martinez, A.B.; Torras, C.(2005) Using Laser and Vision to Locate a Robot in an Industrial Environment: A Practical Experience. *Robotics and Automation. Proceedings of the 2005 IEEE International Conference on*. pp: 3528 - 3533

Browning, B.; Veloso, M.(2005) Real-time, adaptive color-based robot vision. *Intelligent Robots and Systems, 2005. IEEE/RSJ International Conference on.* pp: 3871 – 3876.

Dumitrescu, E., Paturca, S., and Ilas, C., (2009) Optimal Path Computation for Mobile Robots using MATLAB, *Revista de Robotica si Management*, vol. 14, nr.2, pg.36, 2009

Forsyth D. and Ponce J. (2003), *Computer Vision: A Modern Approach.* Upper Saddle River, New Jersey: Prentice Hall.

Gopalakrishnan, A.; Greene, S.; Sekmen, A. (2005) Vision-based mobile robot learning and navigation. *Robot and Human Interactive Communication, 2005. IEEE International Workshop on*. pp: 48 – 53.

Novischi, D., Ilas, C., and Paturca, S., (2010) Obstacle Avoidance based on vision with Low Level Hardware Robots. Performance Comparison, *Eurobot International Conference*, Rapperswill, Switzerland, 2010.

Novischi, D., Paturca, S., and Ilas, C. (2009), Obstacle Avoidance Algorithm for Autonomous Mobile Robot, *Revista de Robotica si Management*, vol. 14, nr.2, pg.40, 2009

Stronger, D.; Stone, P. (2007) A Comparison of Two Approaches for Vision and Self-Localization on a Mobile *Robot. Robotics and Automation, 2007 IEEE International Conference on*. pp: 3915 – 3920.

Tanaka, K.; Yamano, K.; Kondo, E.; Kimuro, Y.(2004) A vision system for detecting mobile robots in office environments. *Robotics and Automation, 2004. Proceedings. 2004 IEEE International Conference on.* Volume: 3. pp: 2279 – 2284.

Zhao, Y.; Cheah, C.C.; Slotine, J.J.E.(2007) Adaptive Vision and Force Tracking Control of Constrained Robots with Structural Uncertainties. *Robotics and Automation, 2007 IEEE International Conference on*. pp: 2349 – 2354.