Real-time image processing algorithms for object and distances identification in mobile robot trajectory planning

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

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

Abstract: In this paper we present a suite of algorithms for determining the possible trajectories of an autonomous robot while navigating through obstacles. For this, we first identify the obstacle orientation and then determine the distance between each two obstacles and compare it with the robot width. The results of these algorithms are intended to be used by the robot path planning, for selecting the best trajectory. The algorithms are relatively simple and accurate and can work successfully even on robots with medium computational resources.
*Keywords***:**. autonomous robot, robot vision, image processing algorithms, Canny edge detection, K-Means clustering.

1. INTRODUCTION

Image processing applications have become extremely important in robotics over the last 15 years. Image processing algorithms have been introduced for a multitude of tasks, such as navigation, orientation, object and surrounding identification (see Forsyth, Florczyk, Browning).

In many such algorithms it is important to extract the essential image information, such as the presence of objects, their properties, including their size and position. There are several approaches for this (see Florczyk). In camera calibration, the goal is to determine the parameters of a camera model, in order to explain the projection of the 3D image to the 2D plane and to allow object reconstruction, including determining info on their size and position (see Forsyth, Florczyk, Vincent, Ji, Kolesnik, Ma). Calibration itself can be split in two main categories: test-area calibration, based on using a known, test environment and self-calibration, which uses images obtained from different positions in order to determine the camera model and to allow the object reconstruction (see Florczyk). Of these, especially the latter has attracted the interest (see Vincent, Horaud, Ji, Kolesnik, Ma, Mirzaei). Many of these deal also with full determination and correction of image distortion (see Vincent).

However, when dealing with robot navigation, very often the problem can be solved with less information on the image and less accurate measurements. For instance, the focus is not on computing the object size and position, but rather on determining the distances between two obstacles, and possibly the length of alternative path segments, in order to choose the optimal path. In both cases we argue that a very precise determination of these distances is not the primary goal, while making a good decision on the path in the early movement stages is. This decision will be based first of all on determining the available trajectory options among the obstacles (by eliminating the corridors that are too narrow for the robot) and then by selecting the one that is most suitable. Since in many cases the robot camera is not able to see the entire movement area, up to the target, this decision will be made on incomplete information.

In this paper we present a practical approach for extracting from the camera image the essential information that is needed for making good decisions during the navigation. In order to simplify the algorithm, there is no self-calibration during the robot navigation. For best results, the camera has to be pre-calibrated, using a test image. The path decisions and planning will then be done in the image 2D plane, without any transformation to the 3D real space.

2. OBJECT ORIENTATION IDENTIFICATION

It is often the case that the objects orientation in the (x,y) coordinate plane is needed in order to build a more accurate environment representation. In order to extract this feature from a 2D image we developed an algorithm based on Canny edge detection and K-Means Clustering, formulated in terms of Expectation Maximization algorithm. The main idea of algorithm is to adequately represent the lines formed by objects edge pixels in the Hough space. In this space a straight line is represented by the pair (r,θ), where r is the distance between the line and the origin and θ is the angle of the vector to the origin for the closest point as depicted in Fig. 1.

In the first step of our algorithm we apply the Canny edge detector on the image obtained from the camera and extract the individual lines. Because edge lines are formed by individual pixels, in the first step of our algorithm we compute the (r,θ) coordinates between each individual pixel and all other pixels, using the equations below.

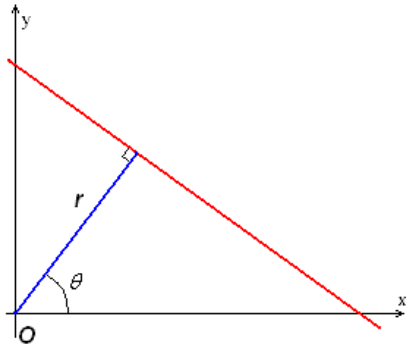


Fig. 1. The *(r,θ)* coordinates line parameterization used for object edge representation.

In these equations d represents the Euclidian distance between two points on a line and the pairs (x_1, y_1) and (x_2, y_2) represent their polar coordinates. All distances are represented in pixels.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (1)$$

$$\theta = \arccos\left(\frac{y_2 - y_1}{d}\right) \qquad (2)$$

$$r = \left|\frac{y_2 \cdot x_1 - y_1 \cdot x_2}{d}\right| \qquad (3)$$

The results obtained from this computation are two matrices, R and Θ, containing all the vector angles and lengths between each pair of individual pixels.

In the second step we first use a modified version of K-Means clustering algorithm that we developed to group angles that have a deviation smaller than 3 degrees. Then we select the clusters that have a number of elements bigger than a specified threshold. This is done because real angles of the edge pixel lines that are prominent yield bigger clusters in terms of number of elements.

In the third step, for each previously selected cluster we again apply K-Means in respect to the distances matrix R. Thus, each angle cluster is "divided" into several clusters based on distance from the origin. For each angle cluster we select only the distance clusters that are bigger than a specified threshold. These resulting clusters represent the object edge lines for which we determined the angles. In the final we determine the object relative orientation to the robot by examining the edge lines for each detected object. An object orientation is expressed by the angle of a primary edge line and its perpendicular. Because some of the object edge lines are partially detected or are not detected at all, in order to determine an object relative position to the robot, we analyze the actual object edges in three cases:

- Case 1, if an edge line is present between two edge lines at 90 degree, then that line is selected as the primary line in the 2D image plane.
- Case 2, if no two 90 degree edge lines are found, then the edge object lines are analyzed and the primary line is the one with the smallest *r* distance in the 2D image plane (r is the perpendicular line from the image origin to the line.)
- Case 3, if the primary edge line is not found as in case 1 or 2, then the principal line is selected to be a horizontal line (at 0 degree angle) that connects the *x* right and *x* left extreme coordinates of the object in the 2D image plane.

The algorithm for obstacle orientation identification is presented below:

Object Orientation Identification Algorithm

1. Detect objects edges
2. Localize objects in the 2D image plane
3. Compute distance (R) and angle (Θ) matrices
4. Compute true angles of object edge lines
5. Compute distances in respect to true edge lines
6. Compute relative object orientation to the robot from the edge lines angles and vector distances

Since in the standard version of the K-Means clustering algorithm one needs to specify the number *k* of clusters prior to the clustering process, we developed an enhanced version to automatically select *k*. In our modified version, the number *k* of clusters increases based on the magnitude of the deviation. Thus, if the deviation of a cluster has larger magnitude than a specified threshold, the cluster is divided and the *k* number of total clusters is increases. In the division process the centroids of the new clusters are chosen to be the old cluster centroid and the value of an element for which the deviation is maximum. The algorithm is presented below.

The Modified K-Means Algorithm

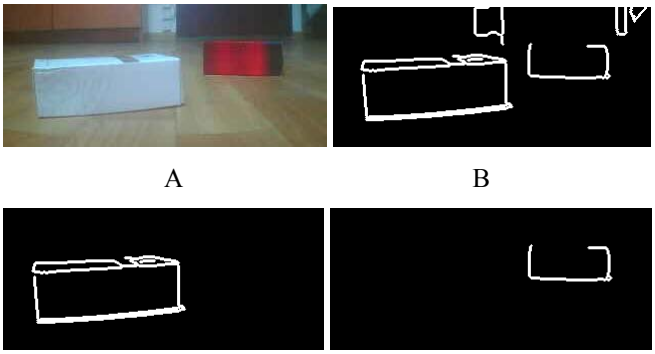
1. Start with one cluster ($k = 1$)
2. noPointsMove = false
3. Assign each point to the cluster
4. While (! noPointsMove)
   a. If(! $k = 1$)
      i. Assign each point to closest cluster
   b. For each cluster
      i. Compute new centroid
      ii. Compute the new deviation

iii. If(noPointsMove && deviation > threshold)

1. noPointsMove = false;
2. Divide cluster

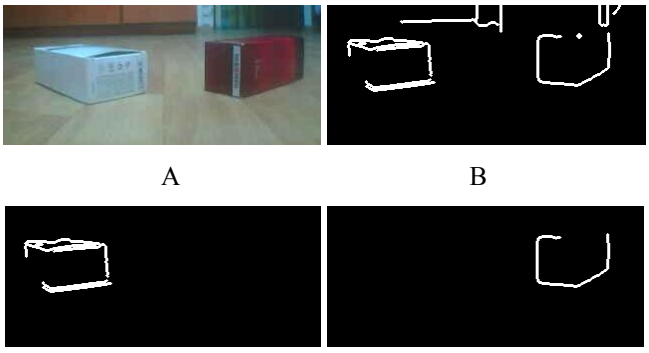
5. Remove small clusters

In Fig. 2 and Fig. 3 we present the processed image, as resulted after applying the object orientation identification algorithm, in two distinct situations.
In the first situation (Fig. 2), the algorithm identified a line connecting two other lines, with which it creates a 90° angle, thus it was selected as the primary line, based on which the object orientation is computed. In the figure, this is the line that is closest to the horizontal. In the second case (Fig. 3), the algorithm did not identified one of the rectangular edges in either two objects, so the primary line is selected to be the one with the smallest distance r from the image origin (the bottom right corner) to the line (i.e. the extension of the segment edges).



A B C D

Fig. 2. Processed Images (principal line selected as the one between lines at 90 degrees): A – Real Image, B – Edge Line Image, C – First Object (at 21 degrees orientation), D – Second Object (at 6 degrees orientation)



A B C D

Fig. 3. Processed Images (two connected lines with smallest *r* distance): A – Real Image, B – Edge Line Image, C – First Object (at 39 degrees orientation), D – Second Object (at 46 degrees orientation)

One of the algorithm limitations is the way it interprets objects that have a circular (curve) shape, or a cylindrical shape. For instance, in the later case, the line connecting the two edges at 90° is a curve. In this case the algorithm considers the primary line as the horizontal that connects the two vertical edges.

3. COLISION FREE PATHS IDENTIFICATION

A collision free path is a path which the robot could follow in order to reach its final goal. For determining the available collision free paths we analyze the obstacles location and their orientation. In each case we associate with each collision free trajectory a distance and weight value that depends on the accuracy with which objects are detected.

In the first case the detected objects are perpendicular to the robot trajectory and parallel to each other. This situation is presented in Fig. 4 and Fig. 5. The distance between the objects and between each object and possible walls is compared to the robot width. In Fig. 4 and Fig. 5, there are no lateral walls and the distance between the two objects that are closer is larger than the robot width, so the robot could pass through. Thus, the possible collision free paths that are determined in this situation are either on the left side of the first object, the right side of the second or between the two objects. For each trajectory marked as free from collisions we may compute the distance that the robot has to travel and assign a weight value of 50%. If the final goal is straight ahead, the smallest travel distance corresponds to the path which is closest to this straight line. The weight of 50% means that there is a 50% probability that this path would be blocked further on, by an object that is undetected yet (e.g. is hidden by the front ones). If there is a more distant object on the path that is detected (as O3 in Fig. 4), there are two situations.

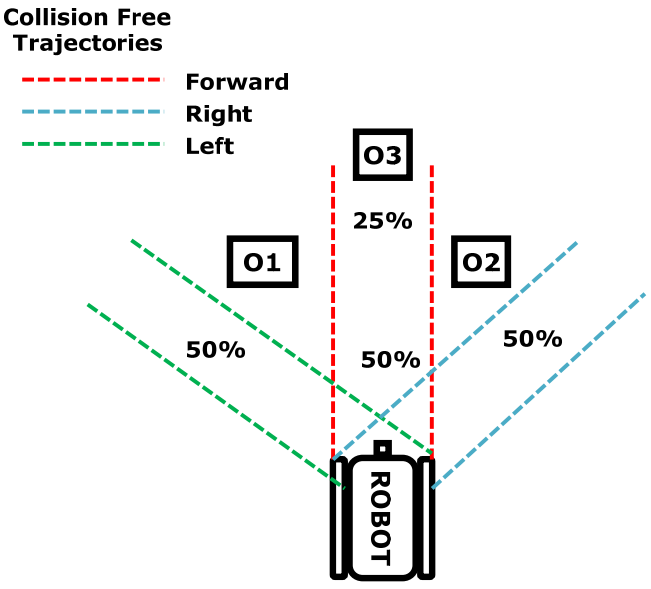


Fig 4. Case 1 – Detected objects are parallel to each other and perpendicular to the robot trajectory. The distance O1-O2 is computed and found larger than robot width, but distances O3-O1 and O3-O2 are assumed that cannot be computed.

In the first, the distances between that object and its neighbours (such as O3-O1 and O3-O2 in Fig. 4) can be determined (details on how to compute it will be shown

below) and hence the collision free paths around that obstacle are found. They would have a 50% weight, for the reasons explained above. In the second situation, the distances cannot be determined. Consequently, the collision free path that contains this next obstacle is assigned a 25% weight. This situation is shown in Fig. 4.

The way that the obstacles are detected by the robot can be seen in Fig. 5.

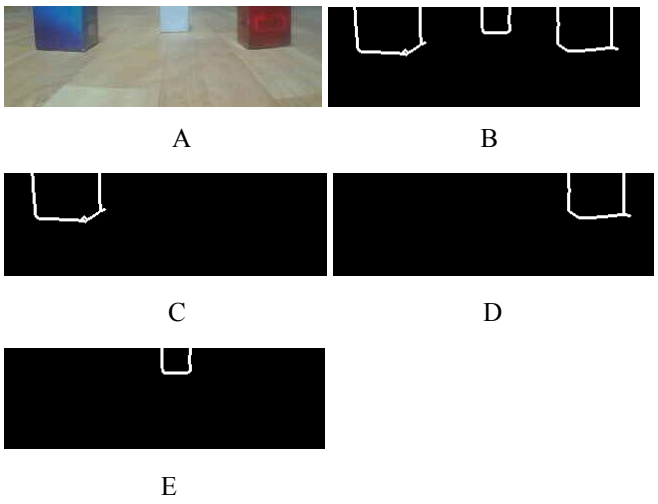


Fig. 5. Processed Images: A – Real Image, B – Edge Line Image, C – Left Object, D – Right Object, E – Centre Object

In the second case obstacles are parallel to each other, but not perpendicular to the robot current orientation. This situation is shown in Fig. 6. The possible collision free trajectories are the same as in the first case. In order to mark the trajectory passing between two objects as a collision free trajectory, we compute the minimum passing distance. This minimum passing distance in this case represents the perpendicular on each object sides. Because we represent each object orientation through the primary line and because we know the edges that are perpendicular, the computation of the minimum distance becomes trivial. The assignment of the weight values for each collision free trajectory follow the same logic as in the first case.

The way that the obstacles are detected by the robot can be seen in Fig. 7.

In the third case the detected objects form an angle to each other and also to the robot current trajectory. This situation is depicted in Fig. 8 – Fig. 11. The possible collision free paths are identical to the previous cases. The minimum distance between the two objects is computed by determining the (x, y) coordinates in the 2D image plane of the point where the primary lines of the objects intersect each other. If the y coordinate of this point is larger than the maximum y coordinate of the two primary lines, then the minimum distance is computed as the horizontal distance between the objects, in the corner with the lower y.

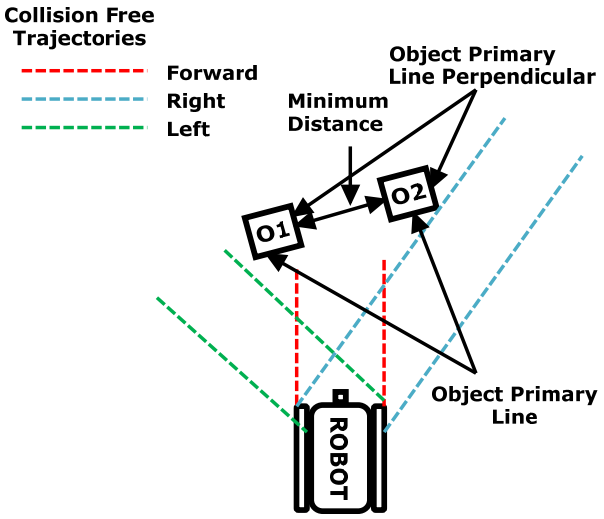


Fig 6. Case 2 – Detected objects are parallel to each other and form an angle with robot current orientation. The objects are O1 and O2.

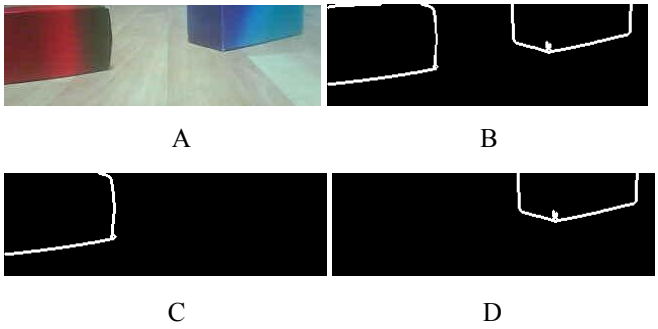


Fig. 7. Processed Images: A – Real Image, B – Edge Line Image, C – Left Object, D – Right Object, E – Centre Object

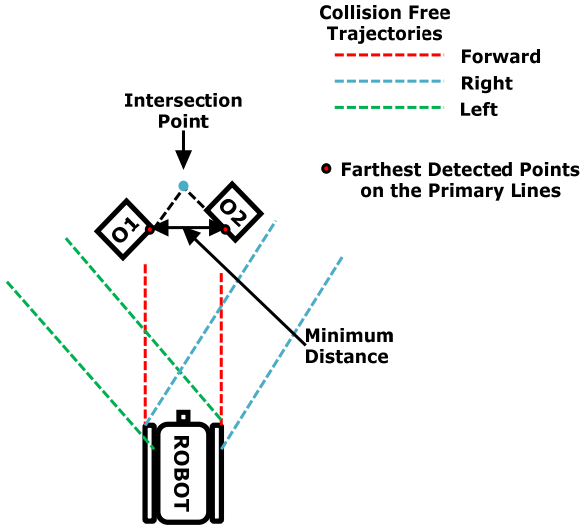
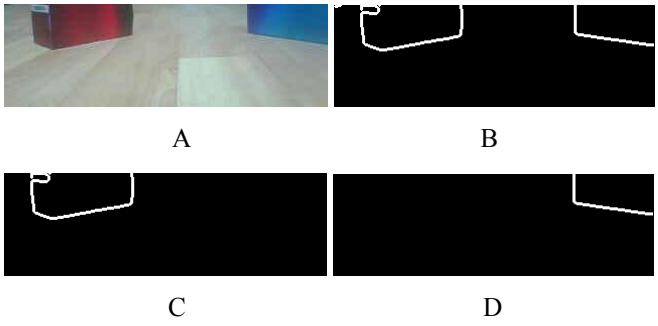


Fig. 8. Case 3A - Detected objects (O1, O2) form an angle with robot orientation and to each other and the intersection point is located above the maximum y coordinate of the primary lines.

This situation is called 3A and can be seen in Fig. 8 and in Fig. 9. In this situation the weight that we assign to this trajectory is 25% because the distance between the two objects could narrow further.

If the *y* coordinate of the intersection point is smaller than the minimum *y* coordinate of the primary lines, the minimum distance is the horizontal distance between the closest points of the object in the *y* direction. This is referred to as case 3B and can be seen in Fig. 10 and Fig. 11. In this situation the weight value that we assign to this trajectory is 50%.



A B

C D

Fig. 9 Processed Images: A – Real Image, B – Edge Line Image, C – Left Object, D – Right Object

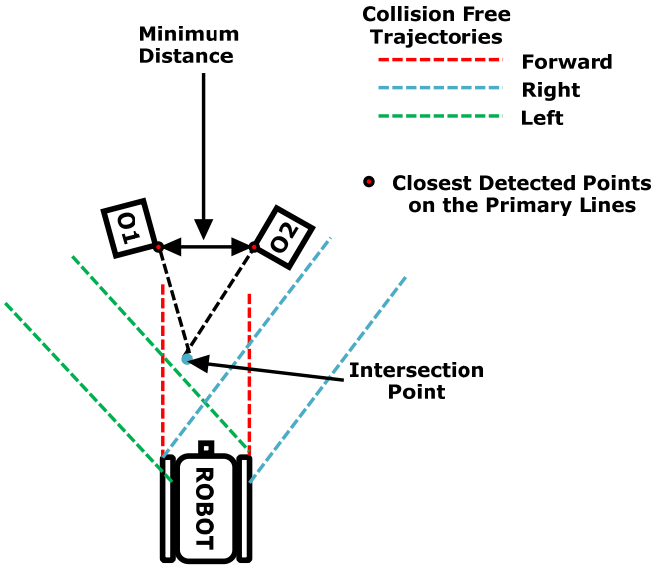


Fig. 10. Case 3B - Detected objects (O1, O2) form an angle with robot orientation and to each other. The intersection point is below the minimum *y* coordinate of the primary lines.

The algorithm for computing the collision free trajectories is presented below:

Collision Free Trajectories Algorithm

For (each pair of obstacles)

If (objects orientation = 90°)

Compute horizontal distance between objects

Compute collision free trajectories

Assign weights

Else If (object 1 orientation = object 2 orientation)

Compute angle of the object interior sides

Compute the length of the perpendicular to each interior side

Compute collision free trajectories

Assign weights

Else

Compute y coordinate of the primary lines intersection point

If(y intersection point > max(y primary lines))

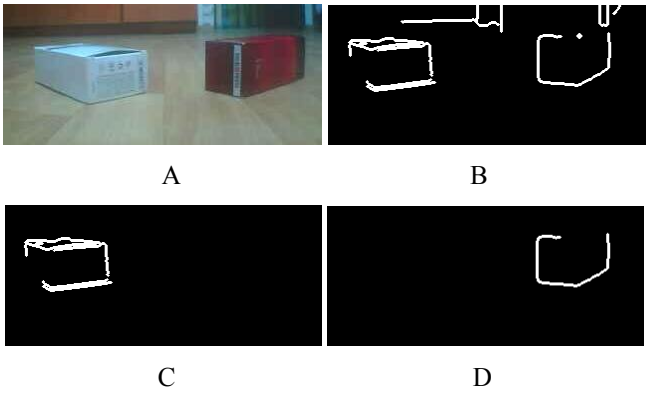
Compute the horizontal distance between *y* maximum coordinates of the primary lines

Else

Compute the horizontal distance between minimum y coordinates of the primary lines

Compute collision free trajectories

Assign weights



A B

C D

Fig. 11. Processed Images: A – Real Image, B – Edge Line Image, C – Left Object, D – Right Object

CONCLUSIONS

In this paper we presented and analysed an algorithm for identifying the object orientation and an algorithm for determining the collision free paths.

The first algorithm is a K-Means algorithm, with the ability of automatically setting the number of clusters.

The outputs of this algorithm are used by a second one, in order to compare the distances between obstacles and thus to determine the possible paths (collision-free paths). There are 3 possible cases for two obstacles positions and we discussed each of them and the way in which the minimum distance is computed. If the robot final goal cannot be seen, the path is determined based on the available information. Thus, each

collision-free path is given a weight, which depends on the probability of being a collision-free path up to the final goal. In the situation in which we cannot determine the minimum distances accurately, the weight assigned reflects this.

Both algorithms are relatively simple and accurate and can work successfully even on robots with medium computational resources.

These algorithms have been implemented and tested using a Blackfin SRV-1 robot.

REFERENCES

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.

Dickmanns, E.D. (2007) Dynamic Vision for Perception and Control of Motion, Springer-Verlag London Limited 2007, ISBN 978-1-84628-637-7

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.

Florczyk, S. (2005) Robot Vision: Video-based Indoor Exploration with Autonomous and Mobile Robots, WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim, ISBN 3-527-40544-5.

Vincent, C.Y; Tjahjadi, T. (2005) Multiview Camera-Calibration Framework for Nonparametric Distortions Removal, *Robotics, IEEE Transactions on,* Vol. 21, No. 5, Oct. 2005, pp.1004-1009.

Horaud, R., Mohr, R., Lorecki, B. (1993) On Single-Scanline Camera Calibration, *Robotics and Automation, IEEE Transactions On,* Vol. 9, No. 1, Feb. 1993, pp.71-75.

Ji, Q. (2004) Self-Calibration of a Rotating Camera With a Translational Offset, *Robotics and Automation, IEEE Transactions On,* Vol. 20, No. 1, Feb. 2004, pp.1-14.

Kolesnik, M.; Baratoff, G. (2000) 3-D Interpretation of Sewer Circular Structures. *Robotics and Automation. Proceedings of the 2000 IEEE International Conference on, San Francisco*. pp: 1453 - 1458.

Ma, S.D.(1996) A Self-Calibration Technique for Active Vision Systems, *Robotics and Automation, IEEE Transactions On,* Vol. 12, No. 1, Feb. 1996, pp.114-120.

Malm, H.; Heyden, A. (2006) Extensions of Plane-Based Calibration to the Case of Translational Motion in a Robot Vision Setting, *Robotics, IEEE Transactions on,* Vol. 22, No. 2, Apr. 2006, pp.322-332.

Mezouar, Y., Chaumette, F. (2002) Path Planning for Robust Image-Based Control, *Robotics and Automation, IEEE Transactions On,* Vol. 18, No. 4, Aug. 2002, pp.534-549.

Mirzaei, F.M., Roumeliotis, S.I. (2008) A Kalman Filter-Based Algorithm for IMU-Camera Calibration: Observability Analysis and Performance Evaluation, *Robotics, IEEE Transactions on,* Vol. 24, No. 5, Oct. 2008, pp.1143-1156.

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.

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.

Zhengyou Z; Quang-Tuan L; Faugeras, O (1996) Motion of an Uncalibrated Stereo Rig: Self-Calibration and Metric Reconstruction. *Robotics and Automation, IEEE Transactions On,* Vol. 12, No. 1, Feb. 1996, pp.103-113.