Fast and Accurate Environment Modelling using Omnidirectional Vision

Patrick Heinemann, Thomas Rückstieß, and Andreas Zell

Wilhelm-Schickard-Institute, Department of Computer Architecture,
University of Tübingen, Sand 1, 72076 Tübingen, Germany
{heinemann, trueckst, zell}@informatik.uni-tuebingen.de

Abstract. This paper describes an algorithm to detect obstacles and landmarks, using the omnidirectional vision system of a RoboCup robot, to build an internal representation of the robot's environment. The restriction to pixels corresponding to an equally spaced grid on the floor around the robot and a biologically inspired fault-tolerant colour segmentation of this grid result in a fast and robust detection. The performance of the environment modelling concerning computation time and accuracy is addressed by comparing experimental results to object positions given by an absolute positioning system.

Introduction

A major task for a mobile robot that has to localise itself and plan collision free paths avoiding stationary and moving obstacles is to build an internal representation of its environment. In many situations it is sufficient to maintain a two dimensional representation, i.e. a bird's eye view of the floor, on which the robot moves, including obstacles and landmarks for self-localisation. In order to create the environment model from the images obtained with an omnidirectional vision system, two basic steps have to be performed. First, the objects and landmarks in the image are identified and second, the real world position of objects and landmarks is calculated from their pixel coordinates. These steps have to be performed in real time for a robot moving in a dynamic environment. The next section gives a detailed description of the environment modelling algorithm, followed by a description of experimental results concerning computation time and accuracy. The last section concludes this paper and gives an outlook on future work.

The Object Detection Algorithm

The major steps of the environment-modelling algorithm are as follows:
1. Transformation of the image pixels into colour classes
2. Segmentation of the image pixels by colour class
3. Clustering of segments that are believed to belong to the same object
4. Mapping of the cluster to world coordinates

Colour Transformation

As the RoboCup environment is well defined with all relevant objects and landmarks being coded in a distinct colour, the first step to model the environment is to transform the colours of the omnidirectional images into several classes. As our frame grabber supplies 2^5 levels per colour dimension, a colour lookup table (*colour LUT*) is trained. With the 32 values per dimension this colour LUT has a size of 32 KB, which easily fits into the second level cache of a standard PC.

Still, transforming the whole image of 768x576 pixels is too time consuming. Although a transformation of the whole image could give some topological information (e.g. colour regions as obtained by the run length encoding process in Bruce et al. [1]), this algorithm transforms only some pixels to reduce the computational load.

For a partial transformation of the image it has to be decided, which pixels of the image should be transformed. There is obviously a trade-off between an efficient transformation and the coverage of the image. The proposed solution to this problem is a regular grid defined in world coordinates with a grid resolution of 10cm to be sure to cover even the smallest robot on the field with at least one gridline in each direction. This is similar to the *receptor* approach in Bonarini et al. ([2]). Although the white field lines are only 5-12cm wide, they are covered lengthwise at least with some gridlines of the rectangular grid. The resolution along the gridlines was set to 2cm.

The mapping of real world coordinates to pixel coordinates, known as *perspective mapping*, is learned from a set of automatically measured correspondences to retrieve the pixels that correspond to the grid points. After removing grid points that were mapped to the same pixel coordinates, the number of pixels of this grid is approximately 40.000 and thus only 9% of the image pixels are processed. These grid points can be easily stored into a list to serve as a *grid LUT* for every frame.

Colour Segmentation

To find sequences of a given colour class in the colour-transformed grid, a fast and fault-tolerant algorithm is needed. Similar problems are subject to ongoing research in molecular biology, where sequences of amino acids (aa) are compared to retrieve the best matching subsequence. A standard algorithm for this is the Smith-Waterman algorithm for local alignment of common molecular subsequences ([3]). It finds best matching substrings by comparing the aa of two strings one by one, assigning high scores for exact matches and lower or negative scores for unequal pairs. In the end of this process, the substring with the best score is considered as the best matching substring. This algorithm is fault-tolerant, as it allows a certain number of unequal aa in the two strings. This number is based on the scores given to the compared aa, which in turn are based on statistical analysis of how likely two aa may be exchanged without affecting the characteristics of the whole sequence.

Regarding the colour transformed gridlines as an aa string, with the colour class of the pixels being an identifier similar to the character of an aa, substrings (segments) of a given colour class can be found. The algorithm assigns positive scores to pixels belonging to the examined colour class and a negative score to pixels of a different

colour class. Pixels of a different colour class would usually end a segment without consideration of errors in the colour transformation step due to noise in the image. Using the Smith-Waterman algorithm, however, segments are extended over small gaps of pixels of a different colour class.

While parsing through the grid lines, the algorithm processes the different colour classes in parallel. A new segment is started, if there is currently no unfinished segment of the pixel's colour class. The algorithm adds a score of +2 for pixels of the same colour class, -1 for pixels of a different colour class, which is known to occur frequently in segments of that colour class, and -3 for pixels of a colour class that is unlikely to occur in a segment of that colour class to a grand total for each unfinished segment. A segment is finished, when its score drops below zero, or when the current gridline is finished. However, the end of the segment is determined as the last position which reached the maximum score in the segment, as this is the point of the highest number of matching pixels and only some small errors in a row.

The number of non-matching pixels that can be skipped without finishing a segment depends on the number of matching pixels that occur before and after. Therefore the fault-tolerance would be higher for long segments, probably connecting two long segments, which do not belong together. To enforce a common number of pixels that can be skipped, the score is limited to a maximum, which cannot be exceeded. This maximum score depends on the size of the objects and the grid resolution. If the segment never reached the maximum score, it is considered as too small and is dropped.

Fig. 1 gives an example of a gridline being parsed by the segmentation algorithm.

Clustering of Segments

The clustering step clusters the segments based on their centre points in pixel coordinates. This is only done for segments of black colour (robots) and orange colour (ball) as these are the only objects that appear on a RoboCup field. For the white line markings no further processing is needed after the segmentation step.

Prior to the clustering, the segments are sorted by the distance r_p to the image centre[1]. For each segment the clustering algorithm then tests if there already exists a cluster in the list of clusters. If this list is empty or the angle $\varphi_{p,sc}$ of the segment's centre point does not lie inside the minimum and maximum angle of any cluster, a new cluster is started. The cluster centre is initialised with the centre of the current segment and a minimum/maximum angle is calculated as $\varphi_{p,sc}$ minus/plus the opening angle ψ of the object. This opening angle is determined as

$$\psi = \arctan\left(\frac{R_{\max}}{f\left(r_{p,cc}\right) + R_{\max}}\right),$$

[1] Coordinates are given as polar coordinates $\left(r_p, \varphi_p\right)$ and $\left(r_w, \varphi_w\right)$ originating in the image centre and the robot centre, respectively.

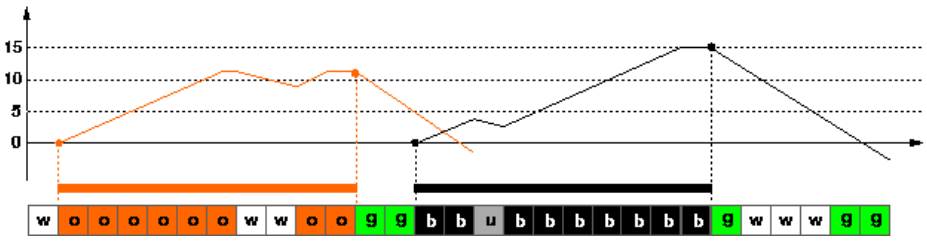

Fig. 1. Example of a gridline being parsed by the segmentation algorithm for orange and black segments. The computed segments are marked with a coloured bar over the gridline. Both segments reach their maximum scores at least once.

where R_{max}=35cm is the maximum radius of a robot in RoboCup MSL (R_{max}=15cm for orange segments which show the ball), and $f(r_{p,cc})$ is the inverted *perspective mapping function* of step 1, known as *inverse perspective mapping function* ([4]).
If the current centre point's angle lies inside the angular range of an existing cluster, the centre point is added to that cluster. The new centre of the cluster is finally updated as the mean of all centre points contained in this cluster.

Fig. 2 visualises the results of the first three steps of the environment modelling algorithm applied to an example image.

Transformation to Real World Coordinates

As the inverse perspective mapping function was learned in respect to the floor plane, only coordinates on the ground level are mapped correctly. Thus, from all segment points of a cluster only the point nearest to the image centre is mapped to its world coordinates as follows:

$$\varphi_w = \varphi_p,\ r_w = f(r_p)\,.$$

Most likely this is a contact point of the object to the floor. The radius R of the object is determined as

$$R = r_w \tan\left(\left(\varphi_{\max} - \varphi_{\min}\right)/2\right),$$

where $\varphi_{\max}$ and $\varphi_{\min}$ are the maximum and minimum angles based on the start and endpoints of the segments. The centre point of the object in world coordinates is then simply $\left(r_w + R, \varphi_w\right)$.

Results

To give exact results of the estimation errors of the environment modelling algorithm, the estimations were compared to measured positions of the highly accurate absolute positioning system W-CAPS ([5]).

A second robot drove to different positions and several measurements were taken with the W-CAPS and the robot vision system at each point. The position estimates from the W-CAPS were averaged to serve as the ground truth, and the mean of the algorithm's estimates was compared to this value resulting in an estimation error for

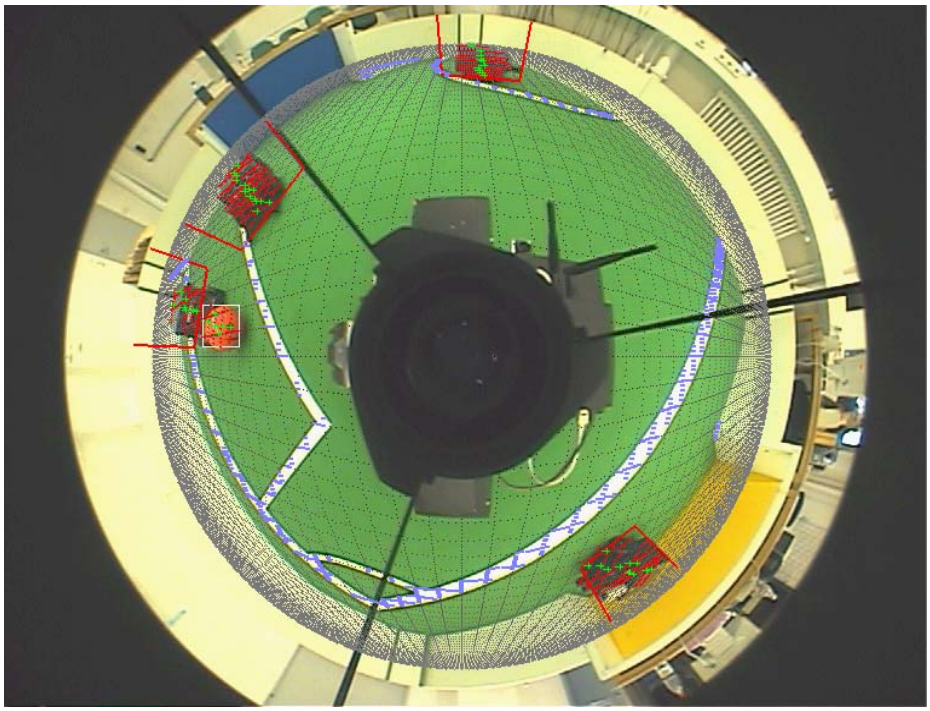

Fig. 2. The grey grid points are transformed into colour classes and used as input to the segmentation step. The computed segments are visualised as blue and red grid points. The green crosses show the centre points of the segments which are clustered into the different objects.

each position. The linearly interpolated error at these points is shown in the error surface in Fig. 3. The mean error over all positions is 8.16cm, with values ranging from 0.77cm to 60.11cm. As can be seen, the system is very accurate in a range of 0.7m to 3m around the robot. The low image resolution for distances of more than 3m, result in a higher estimation error at those distances. Near the robot the contact point between the object and the floor was occluded by the robot's body. Thus, the estimated distance was overestimated.

The computation time for the whole algorithm was 10ms on a Pentium III 850MHz that is used on our robots, leaving enough time for other tasks while processing the full 25 frames per second of the camera.

Conclusion and Future Work

In this paper a fast and accurate environment modelling algorithm based on the images of an omnidirectional vision system was presented. The image is transformed into colour classes based on a trained colour LUT and segmented with a biologically inspired segmentation algorithm. These segments are clustered into objects, which are

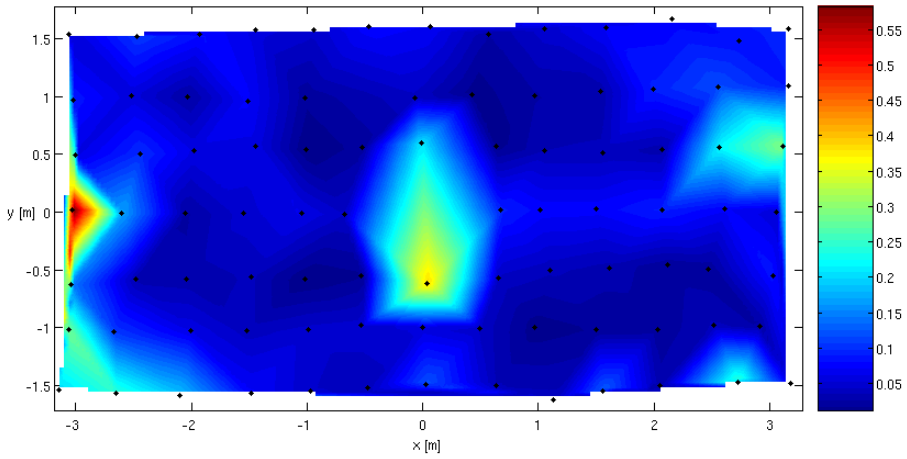

Fig. 3. The Euclidian distance of the mean of the W-CAPS measurements (plotted as black dots) to the mean of the algorithm's estimates is shown in this figure. The robot is situated in the origin of the coordinate system.

then mapped to their real world coordinates with a learned inverse perspective mapping function.

The comparison of the results with an absolute positioning system shows that the object detection algorithm is able to accurately detect the position of objects. With a local search in the region of an object in the image could result in even more precise distance estimations. It will be easy to extend the object detection to detect team colours distinguishing team mates from opponents by simply learning the colour classes and assigning the segments to the same clusters as the black object segments. This algorithm can be easily applied to a perspective camera, too.

References

[1] J. Bruce, T. Balch, and M.Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proc. of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '00)*, volume 3, pages 2061-2066, 2000.

[2] A. Bonarini, P. Aliverti, and M. Lucioni. An omnidirectional vision sensor for fast tracking for mobile robots. In *Proceedings of the IEEE IMTC99*, volume 1, pages 151-155, 1999.

[3] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. In *Journal of Molecular Biology*, volume 147(1), pages 195-197, 1981.

[4] H.A. Mallot, H.H. Bülthoff, J.J. Little, and S. Bohrer. Inverse perspective mapping simplifies optical flow computation and obstacle detection. In *Biological Cybernetics*, volume 64, pages 177-185, 1991.

[5] A. Lilienthal and T. Duckett. An Absolute Positioning System for 100 Euros. In *Proceedings of the IEEE International Workshop on Robotic Sensing (ROSE 2003)*.