In *The IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS 05), Edmonton, Canada, August 2005.

Real-Time Vision on a Mobile Robot Platform*

Mohan Sridharan
Electrical and Computer Engineering
University of Texas at Austin, USA
smohan@ece.utexas.edu

Peter Stone
Department of Computer Sciences
University of Texas at Austin, USA
pstone@cs.utexas.edu

***Abstract*— Computer vision is a broad and significant ongoing research challenge, even when performed on an individual image or on streaming video from a high-quality stationary camera with abundant computational resources. When faced with streaming video from a lower-quality, rapidly moving camera and limited computational resources, the challenge increases. We present our implementation of a vision system on a mobile robot platform that uses a camera image as the primary sensory input. Having to perform all processing, including segmentation and object detection, in real-time on-board the robot, eliminates the possibility of using some state-of-the-art methods that otherwise might apply. We describe the methods that we developed to achieve a practical vision system within these constraints. Our approach is fully implemented and tested on a team of Sony AIBO robots.**

***Index Terms*— Vision and Recognition, Legged Robots.**

I. Introduction

Computer vision is a major area of research with applications in robotics and artificial intelligence. Though significant advances have been made in the use of vision systems on robots (and AI), one of the major drawbacks has been the minimal use of these algorithms for solving practical tasks. Most vision approaches have underlying assumptions (large memory, high computation power or off-line processing) that prevent their use in tasks with significant computational constraints. Our focus is on developing efficient algorithms for solving problems in task-oriented scenarios. One such scenario is the RoboCup Robot Soccer Legged League[1] in which teams of fully autonomous four-legged robots manufactured by SONY (Aibos [1]) play a game of soccer on a $\approx 3m \times 4.5m$ field (see Figure 1).

Fig. 1. An Image of the Aibo and the field. The robot has a limited field-of-view of 56.9^o (hor) and 45.2^o (ver).

Like in real soccer, the robots' goal is to direct a ball into the opponents' goal. The robot's primary sensor is a CMOS camera with a limited view (56.9^o (hor) and 45.2^o (ver)) of its environment, from which it has to extract the information needed for decision-making. Images are captured in the $YCbCr$ format at $30Hz$ and a resolution of 208×160 pixels. The robot has 20 degrees-of-freedom (dof), three in each of its four legs, three in its head, and a total of five in its tail, mouth, and ears. It also has noisy touch sensors, IR sensors, and a wireless LAN card for inter-robot communication. All processing, for vision, localization, locomotion, and decision-making (action-selection), is performed on board the robots, using a 576MHz processor. Currently, games are played under constant and reasonably uniform lighting conditions, but the goal is to enable the robots to play under varying illumination conditions.[2]

The vision problem can then be characterized by the following set of inputs and outputs:

1. *Inputs:*
   * A stream of limited-field-of-view images with defects such as noise and distortion. This reflects the rapid and non-linear changes in the camera position due to the robots' legged locomotion modality.
   * The robots' joint angles over time, particularly the tilt, pan, roll of the camera; and sensor inputs, especially the accelerometer values that can be used to determine the body tilt and roll.
2. *Outputs:*
   * Distances and angles to a fixed set of color-coded objects with known locations, that can be used to *localize* the robot on the field.
   * Distances and angles for a varying set of mobile objects.

Our goal is to generate a reliable mapping from these inputs to outputs with all processing performed on-board the robot, at frame rate, while leaving as much time (and memory) as possible for other modules. Each complete cycle of operation, therefore, can take a maximum of 33msec. Throughout the paper, we provide timing data for the algorithms that we present.

Though motivated by the robot soccer domain, this problem formulation is applicable to general vision-based mobile robots. This paper also serves as a case study demonstrating the practical steps in the process of developing an effective real-time vision system for a mobile robot. A primary distinguishing feature of our task is that the relatively low resolution (and noisy) camera image is the primary sensory

*This work was supported in part by NSF CAREER award IIS-0237699, ONR YIP award N00014-04-1-0545, and DARPA grant HR0011-04-1-0035.

[1] http://www.tzi.de/4legged

[2] The stated goal of the RoboCup initiative is to create a team of humanoid robots that can beat the human soccer champions by the year 2050 on a real, outdoor soccer field [14].

input, unlike many other mobile robots where the focus is mainly on laser/sonar sensors [9].

II. Approach

Our vision algorithm proceeds in a series of stages which convert the sensory inputs into the desired outputs. As a result of the domain constraints we had to develop new algorithms or modify existing techniques to achieve the desired results. Throughout, we provide numerical results to justify the trade-offs made.

Figure 2 shows two representative images from the robot soccer environment. We shall use the same images to illustrate the results of each stage of our vision system (Figures 3–6).[3]

(a) (b)

Fig. 2. Sample Images in the RGB color space.

The vision module consists of four stages: Color cube generation and blob formation (Section III, Marker detection (Section IV) and Line detection (Section V). The paper is organized such that each section isolates and empirically analyzes a component of the vision process. We also present an extension of our approach to variable lighting conditions (Section VI).

Sample videos showing the robot's view as it attempts to score on the yellow goal, after each stage of processing, are available on-line.[4]

III. Color Segmentation and Blob Formation

The first step in our robot vision system is color segmentation. During the first pass over the image, the robot maps each pixel in the raw YCbCr input image into a color class label (m_i), one of ten different colors ($i \in [0, 9]$) in our domain. A complete mapping identifies a label for each possible point in YCbCr space:

$$\forall p, q, r \in [0, 255] \qquad (1)$$
$$\{Y_p, Cb_q, Cr_r\} \mapsto m_i|_{i \in [0,9]}$$

Previous research in the field of segmentation has produced several good algorithms, for example mean-shift [3] and gradient-descent based cost-function minimization [23]. But these involve more computation than is feasible to perform on the robots. A variety of previous approaches have been implemented on the Aibos in the RoboCup domain, including the use of decision trees [24] and the creation of axis-parallel rectangles in the color space [25]. Our approach is motivated by the desire to create fully general mappings for each YCbCr value [6].

We represent this mapping as a *color cube* which is created via an off-board training process. A set of images, captured using the robot's camera, are hand-labeled (*painted*) such that the robot learns the range of $\{Y, Cb, Cr\}$ values that map to each desired color. But, after painting 20 images, only $\approx 3\%$ of the color space is labeled. In order to generalize from the hand-labeled data, the color label assigned to each cell in the color cube is modified to be the weighted average of the cells a certain *Manhattan distance* away (a variant of *NearestNeighbor-NNr*). This operation helps remove the *holes* and edge effects in the color cube, providing a better representation for colors with significant overlap, such as yellow and orange.

To reduce memory requirements (on the robot), we subsample the color space to have values ranging from 0 to 127 in each dimension. The resulting color cube, taking ≈ 2 Mbytes of memory, is loaded on the robot for use in segmenting its input images. The segmented image is the output of this first stage of the vision processing system.

We noticed that the segmentation in the YCbCr space was often sensitive to small changes in illumination, for example yellow being misclassified as orange due to shadows or highlights. Research in rescue robotics has suggested that a spherically distributed color space known as LAB inherently provides some robustness/invariance to illumination changes [12], [17]. To utilize LAB's properties without incurring the overhead of on-line conversion, we modified our color cube generation algorithm.

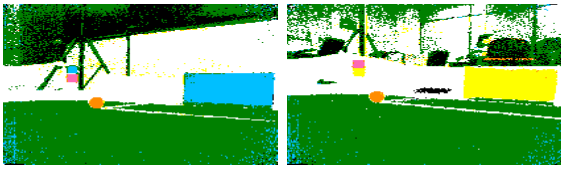
1. The initial painting (off-board training phase) is done in the LAB color space – each painted pixel in the training image maps to a cell in the LAB *color cube*.
2. The NNr operation is performed in the LAB color space.
3. Each cell in the output YCbCr color cube is labeled based on the value in the corresponding cell in the LAB color cube, as determined by a static transformation.

Thus, on-line segmentation incurs no extra overhead over the baseline approach. The pixel-level segmentation process is reduced to that of a table lookup and takes ≈ 0.120msec per image.

Though it is possible, for a given illumination, to tune the YCbCr color cube such that the segmentation is almost as good as it is with the LAB cube, using LAB helps reduce the amount of misclassification with minor changes in illumination, especially during competitions when there are several objects around the field (e.g. clothing of spectators) that are similar to the colors the robots have been trained to recognize. We compared the segmentation accuracy of the two color spaces over a set of images (≈ 15) that were captured to reflect small changes in illumination. The *Ground Truth* was provided by hand-labeling specific regions of the images (on average $\approx 20\%$ of the pixels were hand-labeled). The entire image is not considered because we are interested only in the colors of the markers and other objects on the field and/or below the horizon, and the *correct* classification result is unknown for several background pixels in the image. The classification accuracies (%) were 81.2 ± 4.4 and 92.7 ± 2.5 for YCbCr and LAB respectively (statistically significant at 95% confidence).

[3]The images appear in color in the electronic version of the paper.
[4]www.cs.utexas.edu/users/AustinVilla/?p=research/robust_vision

Figure 3 shows the the result of segmentation, using the new approach in the LAB color space.



(a) (b)

Fig. 3. Sample Segmented Images.

A. Blob Formation

The next step in vision processing is to find contiguous *blobs* of constant colors, i.e., we need to *cluster* pixels of the same color into meaningful groups. Though past research in this area has resulted in some good methods [10], [13], doing this efficiently and accurately is challenging since the reasoning is still at the pixel-level. Computationally, this process is the most expensive component of the vision module that the robot executes.

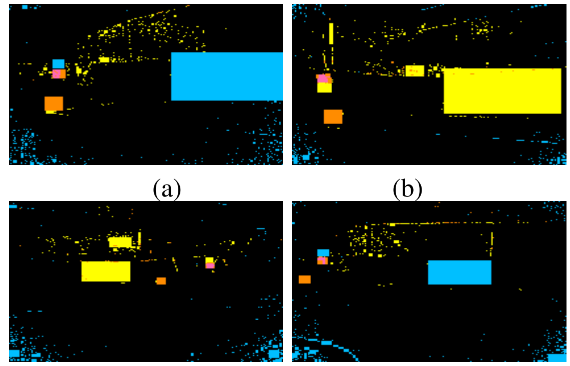
Our approach to blob formation is modeled closely after previous approaches on the Aibo [6], though we add features to optimize the process. As the pixels in the image are segmented they are organized into run-lengths represented as the start point and length in pixels of a contiguous color strip.[5] As an optimization, we only encode the run-lengths corresponding to colors that identify objects of interest – we omit the colors of the field (green) and the borders (white). Though these colors are useful in detecting the field borders and lines, we achieve that by incorporating an efficient line-detection algorithm (Section V).

Next, we use an implementation of the Union-Find algorithm [4] to merge run-lengths of the same color that are within a threshold *Euclidean* distance from each other. This results in a set of blobs, each of constant color. During this process, we also progressively build *bounding boxes* i.e. rectangular boundaries around the regions. This abstraction helps categorize each region by the four vertices of the bounding rectangle. We then end up with a set of bounding boxes, one for each blob in the current image, and a set of properties corresponding to each blob, such as the number of pixels (of the blob color) it envelopes. These properties are used in the object recognition phase (Section IV). Our technical report [5] has complete details.

Errors in the segmentation phase due to noise and/or irrelevant objects in the image can lead to the formation of spurious blobs and make object recognition challenging. In Figure 4 we show the results of blob formation on the sample set of images and a couple of additional images that lead to spurious blobs.

The vision module, including segmentation and blob formation, takes $\approx$ 20msec per image.

[5]details on run-length encoding can be found in image processing books [11].



(a) (b)

(c) (d)

Fig. 4. Sample Blobs.

IV. Object Recognition

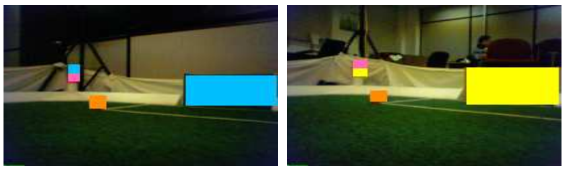
Once we have candidate blobs, the next step is to recognize the relevant objects in the image. Object recognition is a major area of research in computer vision and several different approaches have been presented, depending on the application domain [2], [20], [26]. Most of these approaches either involve extensive computation of object features or large amounts of storage in the form of object templates corresponding to different views, making them infeasible in our domain. Further they are not very effective for rapidly changing camera positions. We determine the objects of interest in the image using domain knowledge rather than trying to extract additional features from the image.

All the objects in the robot's environment (including the fixed markers and moving objects) are color-coded and thus we can use the blobs determined previously to recognize the objects. Even so, the task is non-trivial as there are generally several objects in and around the field that could be segmented as the same color as the objects of interest – note for example, the people, chairs, walls, and computers in our sample images.

To recognize objects we first eliminate blobs that do not correspond to strict constraints of size, density and position in the image, based on domain knowledge. We filter the blobs through a set of heuristics designed to detect blobs that are too small to correspond to objects, or that are not *dense* enough (measured as the ratio of appropriately colored pixels within the bounding box). For example, all objects of interest to the robots are either on the ground or a certain distance above the ground and have bounding boxes with high densities. Also, the ball is mostly enveloped in a square bounding rectangle except when it is partly occluded. Full details of the heuristics are available in our team technical report [5]. These heuristics are easy to apply since the required properties were stored in the blob formation stage (Section III-A). These properties are also used to determine the probability of occurrence of each object (based on the degree of correspondence with the expected values of the properties).

We analyzed the performance of the object recognition system by calculating the ratio of images over which the objects were correctly detected, over eight sequences of $\approx$ 200 images each. We performed this first with the robot stationary (and looking at the objects in view) and then with the robot moving. The corresponding classification accuracies were 100% and 92.7% respectively. Though the motion, as expected, causes a decrease in the accuracy of object recognition (due to the motion-based distortion of the image), there are no false positives in either case.

Figure 5 shows the blobs detected as objects superimposed on the original (RGB) images.



(a) (b)

Fig. 5. Sample Object Recognition.

By eliminated the spurious blobs, this process ensures that we recognize all the objects in an image while at the same time making the object recognition phase highly efficient and computationally inexpensive. The vision module, up to the object recognition phase takes ≈ 28msec per frame, enabling us to process images at frame rate.

This object recognition algorithm does not recognize the lines in the environment, a great source of information. Next, we shall describe the algorithm that we use to detect the lines.

V. Line/Line Intersection Detection

Lines with known locations can be important sources of information for the robots, particularly in the robot soccer domain, where the robots' main focus (during a game) is the ball and other robots may occlude the beacons and goals.

Previous research on detecting lines/edges in an image has resulted in methods such as Hough Transforms and edge detectors such as Canny, Sobel [11]. Most popular methods determine the edge pixels by convolving a suitable mask across the image, an operation that is too time-consuming for our purposes.

Our line-detection method, motivated by a previous approach in the RoboCup environment [18], utilizes environmental knowledge to detect edge pixels: edges of interest on the Robot Soccer field involve a white-green or green-white-green transition corresponding to the borders and the field lines respectively. We do not determine the bounding boxes corresponding to the white blobs and green field and use heuristics to determine the actual lines/edges (as mentioned in Section III-A) because of the computation involved.

In our line-detection algorithm, we begin, as in [18], by performing a series of vertical scans on the segmented image with the scan lines spaced $4-5$ pixels apart. In addition to making the scan faster, this ensures that we incorporate noise filtering – noisy lines that extend only a few pixels across are automatically eliminated. When processing a scan line, the robot checks for the occurrence of candidate edge pixels by looking for the green-white and white-green transitions. Over this baseline approach, we add additional features as follows. To bias the scan procedure towards detecting edge pixels that are closer to the robots, our scan lines proceed from the bottom of the image to the top, i.e. the border edge pixels now correspond to green-white transitions. Once an edge pixel is detected along a scan line, we do not process the remainder of the scan line and proceed directly to the next scan line. Though this excludes the possibility of detecting, for example, a border line above a field line, the procedure is based on the assumption that the observation of lines closer to the robot provides more reliable information. Candidate edge pixels in the image are accepted *iff* they also have a significant amount of green below them. Once we have a set of candidate edge pixels, we incorporate a set of heuristic filters whose parameters were determined experimentally. For example, we reject pixels that do not project to a point (on the ground plane) within a threshold distance in front of the robot [5].

Instead of using these pixels directly as localization inputs, as in [18], we find the lines that these edge pixels represent. We *cluster* the candidate edge pixels into lines in the image plane using the Least Square Estimation procedure [15]. This is an efficient method that can be performed incrementally, i.e. lines can be fit to the candidate edge pixels as they are found and new edge pixels can be either merged with existing lines or they can be used to generate new lines. We introduce filters for suppressing noise and false positives — at the line detection level we remove outliers (candidate edge pixels that are not close to any of the known edges) and also consider lines *iff* they can account for more than a threshold number of pixels.

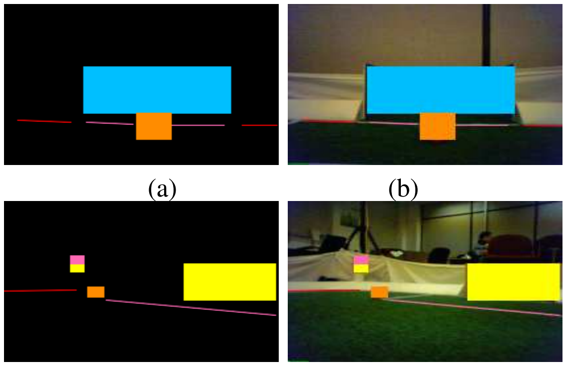
Although line pixels (or lines) provide useful information, the intersection of lines are more meaningful (and less noisy) since they involve much less ambiguity. They are not unique because of the symmetry of the field, but the ambiguity can be resolved to some extent based on the knowledge of the previous known position. To determine the line intersections we just consider a pair of lines at a time. Line intersections are accepted only if the angles between the lines are within experimentally determined heuristic thresholds. For more information on how lines are used in localization, see [5].

Figure 6 shows a set of images with the lines detected: field lines are drawn in pink and are distinct from the border lines which are red.

We also analyzed the performance over image sequences by determining the ratio of images where the line intersections were correctly identified.

Over ≈ 2000 images, half of which had line intersections in them, we obtained accuracies of 100% and 93.3% respectively for the stationary and moving robot cases. The motion, once again, causes a decrease in the accuracy (again due to the motion-based distortion of objects in the image). As in the case of object recognition there are no false positives even when the robot is in motion.

In fact, we noticed a significant difference in our localization accuracy once we incorporated the information regarding the lines/line intersections as inputs [22]. Also, the process



(a) (b) (c) (d)

Fig. 6. Sample Line Recognition.

is not computationally expensive and we are able to perform the entire visual processing in $\approx$ 31msec per frame so that the robot is able to operate at frame-rate with $\approx$ 2msec per frame to spare for other computations.

VI. Illumination Invariance

To this point, the approach we have described has assumed that the lighting conditions are relatively constant. Though using the LAB color space enables robustness to small changes, our eventual goal is to enable the robots to operate in a broad range of, and changing, lighting conditions. Here, we present our approach to move towards that goal.

Color constancy (illumination invariance) is a major research focus in the field of computer vision. In the past, color constancy has been studied primarily on static cameras with relatively loose computational limitations [7], [8], [19]. Lenser and Veloso [16] presented a tree-based state description/identification technique for this same platform that uses a time-series of average screen illuminance to distinguish between illumination conditions: the absolute value distance metric being used as the similarity measure. We explore an alternative approach based on color space distributions.

In our lab, the intensity on the field varies from the *dark* ($\approx$ 400lux with just the fluorescent ceiling lamps) to the *bright* ($\approx$ 1500 lux with the additional lamps turned on). We can gradually vary the illumination between these two levels. The quality of the camera image makes it infeasible for the robot to work at illuminations lower than $\approx$ 400lux.

The robot was trained to recognize and distinguish between three different illumination conditions: *dark*, *interm* and *bright* – the *interm* illumination being in between the extreme lighting conditions. The initial training phase equipped the robot with a color cube and a set of training distributions for each of the three illumination conditions. The training distributions were obtained by allowing the robot to capture a set ($\approx$ 20) of images under each illumination and generating histograms after transforming from the YCbCr space to the normalized RGB (*rgb*) color space since it is more robust to minor illumination changes. Also, as $r + g + b = 1$, any two of the three features are a sufficient statistic for the pixel values, thereby lowering the storage requirements. We stored the distributions in the (r, g) space, quantized into 64 bins along each dimension.

During the online testing phase, the robot generated the distribution in the r-g space corresponding to the test image. This distribution was compared to the previously saved distributions and was assigned the illumination class of the distribution that was most *similar* to the test distribution. As a similarity measure, we use *KL divergence*: given two 2D (r, g) distributions A and B (with $N = 64$, the no. of bins along each dimension),

$$KL(A,B) = -\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}(A_{i,j} \cdot ln\frac{B_{i,j}}{A_{i,j}}) \qquad (2)$$

The more similar two distributions are, the smaller is the KL-divergence. Since the KL-divergence is a function of the *log* of the color distributions, it is less sensitive to large peaks in the observed color distributions in comparison to other measures that do not have this property. Using this measure, the robot was able to correctly classify the test distributions and identify and distinguish between the three illumination conditions. The only restriction, based on computational constraints, was to test for illumination change no more than twice a second.

We tested the performance of the robot in illumination conditions in between the three illuminations it was explicitly trained for and observed that the robot picked the illumination condition (among the three it was trained for) that was *closest* to the test condition (see [21] for full details). We designed a *find-and-walk-to-ball* task where the robot starts from the center and has to find and walk to the ball, which is placed a certain distance in front of the yellow goal. With a single color cube, when the illumination is drastically changed, it is unable to perform the task. Now with the three illumination models, it is able to perform the task even for illuminations that it is not explicitly trained for. During the experiments, the robot started off in the bright illumination and a change in illumination was made $\approx$ 1.5sec into the task. The robot waits for several test frames before it accepts a change in illumination (a filtering procedure). Thus we would not expect the robot to perform the task as quickly as in constant lighting. Table I shows the corresponding results, averaged over ten trials.

The fact that the robot is still able to perform the task demonstrates that the switching among color cubes is working. The first row in Table I refers to the case where the robot performs the task in the bright (normal) illumination and no change in illumination occurs. The other two rows correspond to the case where a change in illumination occurs.

Lighting	Time (sec)
Bright-Constant	6.7 ($\pm$0.6)
bet. bright and interm	12.27 $\pm$0.5
bet. interm and dark	13.3 $\pm$2.0

TABLE I

TIME TAKEN (IN SECONDS) TO *find-and-walk-to-ball*

Though this procedure is not required during the normal game, it can still be performed in real-time on the robot.

Addition of this procedure causes just a slight decrease in frame rate from 30fps to $\approx$ 25fps.

VII. Summary and Conclusions

Significant advances have been made in the field of computer vision algorithms leading to its increasing use in related fields such as AI and robotics. Still, the use of these methods in practical tasks with computational constraints has been minimal, primarily because it is infeasible to run many algorithms in real-time with limited computational resources.

In this paper, we have described the development of an entire vision system on a mobile robot platform with a camera as the primary sensor. The camera has limited field-of-view and image resolution, and the legged locomotion introduces dynamic changes in camera position. In addition, all computation has to occur on-board in real-time with limited computational resources. These constraints in the RoboCup legged robot domain, representative of current directions in mobile robotics, enforce the need for efficient algorithms. We have shown that with innovative algorithms and modifications to existing ones it is possible to build a real-time vision system without compromising on the desired quality of performance. In the process, we have been able to efficiently tackle hard vision problems such as segmentation, object recognition and color constancy.

This paper has presented detailed empirical results in isolated test scenarios of several components of our vision system, including the choice of color space, the overall effectiveness of object detection and line recognition, timing data about the various phases of the cycle, and the robustness of our approach to illumination changes. Given the many-faceted task we have attempted to solve during the development of our robot vision system, it is difficult to directly compare our end result with other vision modules, even within the RoboCup community: the performance of each complete system on the overall task is a result of localization, locomotion, and decision-making modules in addition to the vision processing. However anecdotal comparisons have shown our vision system to be at least as efficient and robust as those of other RoboCup teams and it has allowed us to achieve good overall performance in the soccer task, enabling our third place finish at the RoboCup 2004 US open and quarter-finals appearance at RoboCup 2004.

A main contribution of this paper is a case study of practical steps in the process of developing an effective real-time vision system for a mobile robot, thereby making it suitable for other related robot vision tasks (surveillance, rescue). Further details about all stages of our vision processing algorithms are available in our technical report [5] and image sequences illustrating the process are available from: www.cs.utexas.edu/users/AustinVilla/?p=research/robust_vision

Acknowledgments

The authors would like to thank the UT Austin Villa team for their efforts in developing the software used as a basis for the work reported in this paper.

References

[1] The Sony Aibo robots, 2004. `http://www.us.aibo.com`.

[2] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *Pattern Analysis and Machine Intelligence*, April 2002.

[3] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (Second Edition)*. MIT Press, September, 2001.

[5] P. Stone et al. UT Austin Villa 2004: Coming of Age, AI Technical Report 04-313. Technical report, Department of Computer Sciences, University of Texas at Austin, October 2004.

[6] William Uther et al. Cm-pack'01: Fast legged robot walking, robust localization, and team behaviors. In *The Fifth International RoboCup Symposium*, Seattle, USA, 2001.

[7] G. Finlayson, S. Hordley, and P. Hubel. Color by correlation: A simple, unifying framework for color constancy. *In IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11), November 2001.

[8] D. Forsyth. A novel algorithm for color constancy. *In International Journal of Computer Vision*, 5(1):5–36, 1990.

[9] D. Fox. Adapting the sample size in particle filters through kld-sampling. *International Journal of Robotics Research*, 2003.

[10] A. L. N. Fred and A. K. Jain. Robust data clustering. In *The International Conference of Computer Vision and Pattern Recognition*, pages 128–136, June 2003.

[11] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, 2002.

[12] Jeff Hyams, Mark W. Powell, and Robin R. Murphy. Cooperative navigation of micro-rovers using color segmentation. *In Journal of Autonomous Robots*, 9(1):7–16, 2000.

[13] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[14] Hiroaki Kitano, Minoru Asada, Itsuki Noda, and Hitoshi Matsubara. Robocup: Robot world cup. *IEEE Robotics and Automation Magazine*, 5(3):30–36, 1998.

[15] Least Square Principle for Line Fitting. At URL http://mathworld.wolfram.com/LeastSquaresFitting.html.

[16] S. Lenser and M. Veloso. Automatic detection and response to environmental change. In *The International Conference of Robotics and Automation (ICRA)*, May 2003.

[17] B. W. Minten, R. R. Murphy, J. Hyams, and M. Micire. Low-order-complexity vision-based docking. *IEEE Transactions on Robotics and Automation*, 17(6):922–930, 2001.

[18] T. Rofer and M. Jungel. Vision-based fast and reactive monte-carlo localization. In *The IEEE International Conference on Robotics and Automation*, pages 856–861, Taipei, Taiwan, 2003.

[19] C. Rosenberg, M. Hebert, and S. Thrun. Color constancy using kl-divergence. In *In IEEE International Conference on Computer Vision*, 2001.

[20] A. Selinger and R. C. Nelson. A perceptual grouping hierarchy for appearance-based 3d object recognition. *Computer Vision and Image Understanding*, 76(1):83–92, October 1999.

[21] M. Sridharan and P. Stone. Towards illumination invariance in the legged league. In *The Eighth International RoboCup Symposium*, Lisbon, Portugal, 2004.

[22] Mohan Sridharan, Gregory Kuhlmann, and Peter Stone. Practical vision-based monte carlo localization on a legged robot. In *The International Conference on Robotics and Automation*, To Appear 2005.

[23] B. Sumengen, B. S. Manjunath, and C. Kenney. Image segmentation using multi-region stability and edge strength. In *The IEEE International Conference on Image Processing (ICIP)*, September 2003.

[24] The UNSW Robocup 2001 Sony Legged League Team. *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002.

[25] The UPennalizers Robocup 2003 Sony Legged League Team. *RoboCup-2003: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2004.

[26] A. Torralba, K. P. Murphy, and W. T. Freeman. Sharing visual features for multiclass and multiview object detection. In *The IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Washington D.C., 2004.