Real-Time Vision on a Mobile Robot Platform

Mohan Sridharan and Peter Stone
Department of Computer Sciences
The University of Texas at Austin
Austin, TX, 78731

Under review - not for citation

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, jerkily-moving camera and limited computational resources, the challenge only increases. In this paper we present our implementation of a real-time vision system on a mobile robot platform that uses a camera image as the primary sensory input. The constraints imposed on the problem as a result of having to perform all processing, including segmentation and object detection, in real-time onboard the robot eliminate the possibility of using some state-of-the-art methods that otherwise might apply. We present 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, enabling them to place among the top finishers at an annual international robot soccer competition.

1. Motivation

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 in general), one of the major drawbacks has been the minimal use of these algorithms for solving practical tasks. Most vision approaches have underlying assumptions (such as large memory, high computation power and 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 robotic dogs (Aibos [2]) manufactured by SONY play a game of soccer on a $\approx 3m \times 4.5m$ field (see Figure 1).

Figure 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 while preventing the ball from entering their own goal. The robot's primary sensor is a CMOS camera located in the robot's nose with a field-of-view of 56.9^o (hor) and 45.2^o (ver), providing the robot with a limited view of its environment from which it has to extract the information needed for decision-making. The images are captured in the $YCbCr$ format at a frame rate of $30Hz$ and image 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, and any lag in frame rate places the robot at a severe disadvantage in terms of reaction time, etc. 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]

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

[2] The stated ultimate 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 [15].

The vision problem that concerns us can be characterized by the following set of inputs and outputs:

1. *Inputs:*
   - A $30Hz$ stream of 208×160 limited-field-of-view images in the YCbCr color space. The image stream reflects rapid and non-linear changes in the camera position due to the robots' legged (as opposed to wheeled) locomotion modality and includes many defects such as noise and distortion. The images include many objects of interest, but also many unpredictable elements.
   - The robots' joint angles over time, particularly the tilt, pan, roll of the camera.
   - The sensor inputs, especially the accelerometer values that can be used to determine the body tilt and roll.
2. *Outputs:*
   - Distances and angles (with associated probability measures) to a fixed set (8 in our case) of color-coded objects with known locations that can be used to *localize* the robot on the field.
   - Distance, angles and probability measures of 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, ideally at frame rate, while leaving as much time as possible for localization, locomotion, and decision-making. Another constraint is the memory available on board the robot. In order to proceed at frame rate, each complete cycle of operation can take a maximum of 33msec. Throughout the paper, we provide timing data for our presented algorithms.

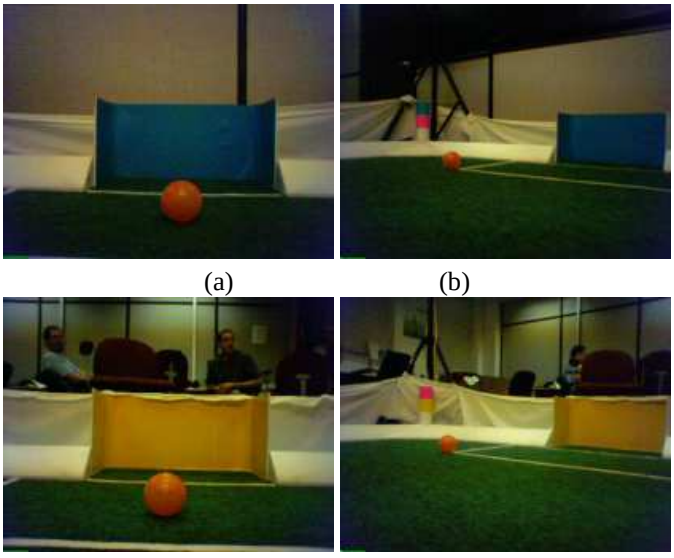
Though motivated by the robot soccer domain, this problem formulation is applicable to general vision-based mobile robots. This paper therefore also serves as a case study demonstrating the practical steps in the process of developing an effective real-time vision system for a mobile robot. A primary distinguishing feature of our task is that the camera image is the primary sensory input, unlike many other mobile robots where the focus is mainly on laser/sonar sensors [10]. The camera jerks around a lot due to the legged (as opposed to wheeled) locomotion modality, and images have a relatively low resolution and possess common defects such as noise and distortion.

Our vision processing algorithm detailed in Section 2 proceeds in a series of stages — color segmentation, blob formation, and object recognition — which, overall, convert the sensory inputs into the desired outputs identified above. Several popular techniques have been developed in vision research for these (or similar) tasks. But given our constraints we had to develop new algorithms or modify existing techniques to achieve the desired results. Throughout, we provide numerical results that justify the trade-offs we made to satisfy our constraints.

2. Approach

In this section, we present a step-by-step description of the vision system that we developed for our legged robot platform.

Figure 2 shows four representative images from the robot soccer environment including shots of both goals, the ball, and two of the four beacons. For the purposes of visualization, we have converted them from their native YCbCr color space to RGB. Throughout the paper we use these same images to illustrate the results of each stage of our vision system (Figures 3–6).[3]



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

Figure 2: Sample Images in the RGB color space.

The vision module consists of four stages: Color cube generation, described in Section 2.1, Blob formation (Section 2.2), Marker detection (Section 2.3) and Line detection (Section 2.4). We conclude this section with an extension of our approach to variable lighting conditions (Section 2.5).

Sample videos showing the robot's view as it attempts to score on the yellow goal, both as raw footage and after each stage of processing, are available on-line. [4] The videos show that the camera moves readily and jerkily as the robot performs its task and that many irrelevant objects and colors appear in the field of view.

2.1. Color Segmentation

The first step in our robot vision system is color segmentation. During the first pass over the image, the robot maps

[3] The images appear in color in the electronic version of the paper.
[4] http://www.cppreference.com/cvpr05-aibovision.html

each pixel in the raw YCbCr input image into a color class label (m_i). In our target domain, the robot needs to recognize ten different colors ($i \in [0, 9]$). A complete mapping identifies a label for each possible point in YCbCr space:

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

Segmentation is a well-researched field in computer vision with several good algorithms, for example mean-shift [6, 22]. But these involve more computation than is feasible to perform on the robots given our constraints (as described in Section 1). A variety of previous approaches have been implemented on the Aibo robots for use in the RoboCup domain including the use of decision trees [23] the creation of axis-parallel rectangles in the color space [24]. Our baseline approach is motivated by the desire to create fully general mappings from the YCbCr values (ranging from $0-255$ in each dimension) to the color labels $(0-9)$ [26].

We represent this mapping as a *color cube* which is created via an off-board training process. A set of images are captured using the robot's camera. These images are then hand-labeled (*painted*) such that the robot learns the range of $\{Y, Cb, Cr\}$ values that map to each desired color. Each pixel that is painted in the training images represents an individual instance of the mapping $\{Y_p, Cb_q, Cr_r\} \mapsto m_i|_{i \in [0,9]}$ and is taken as ground truth. However, after painting 20 images, only about 3% of the color space is labeled. Thus, in order to generalize from this 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 from the cell (a form of *Nearest Neighbor-NNr*). This operation helps remove the *holes* and smooths out the edge effects in the color cube. In the end, we find significant overlap among the labelings of some colors, such as yellow and orange, that could not be represented with axis-parallel labeling.

The painting and NNr computation are both done during an off-board training phase. To reduce memory requirements, we subsample the color space to have values ranging from 0–127 in each dimension. The resulting color cube, taking ≈ 2 Mbytes of memory, is then loaded on the robot for use in segmenting its input images into the colors it has been trained to recognize. The segmented image is the output of this first stage of the vision processing system.

Though the YCbCr color space has been used for most previous work in this domain, we noticed that the segmentation was often sensitive to small changes in illumination, for example leading to yellow being misclassified as orange due to shadows. Previous research in rescue robotics has suggested that a spherically distributed color space known as LAB inherently provides some robustness/invariance to illumination change [13, 18]. In order to take advantage of LAB's properties without incurring the overhead of on-line conversion, we modified our color cube generation algorithm as follows.

1. The initial painting is done in the LAB color space during the off-board training phase. That is, each painted pixel in the training image maps to a cell in the LAB *color cube*.
2. Similarly, the NNr operation is computed 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 static (off-line) conversion.

As such, on-line segmentation incurs no extra overhead over the baseline approach. Whereas for a given illumination, it is possible to tune the color cube in the YCbCr color space such that the segmentation is almost the same as it is with the cube in the LAB color space, we found that that using LAB as the underlying color space helped reduce the amount of misclassification with minor changes in illumination, especially during competitions when there are several objects around the field (e.g. clothing worn by the spectators) that are similar to the colors the robots have been trained to recognize.

Figure 3 shows the the result of segmentation, using the new approach in the LAB color space, on the sample set of images (as in Figure 2).

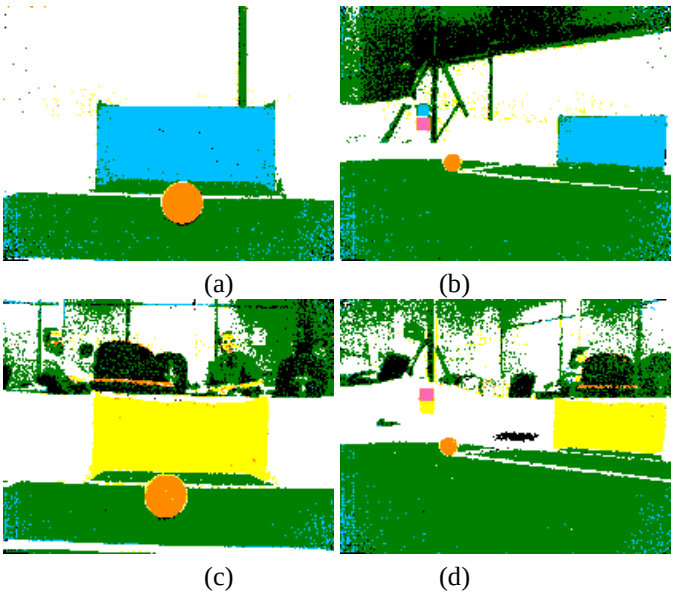


Figure 3: Sample Segmented Images.

From a computational perspective, the on-line pixel-level segmentation process is reduced to that of a table lookup and takes ≈ 0.120msec per image.

2.2. Blob Formation

Once the input images have been successfully segmented, the next step is to find contiguous *blobs* of constant colors. That is, we need to extract useful information from the color coded image by *clustering* pixels of the same color into meaningful groups. This again is a well-researched area in computer vision [11, 14]. However making this process happen both efficiently and accurately is particularly challenging due to the fact that the reasoning is still at the pixel-level. Computationally, this process is by far the most expensive component of the vision system that the robot executes.

Our approach to blob formation is modeled closely after previous approaches on the Aibo [26], though we add features to optimize the process. As the pixels in the image are being segmented (during the first pass over the image) they are organized into run-lengths represented as the start point and length in pixels of a contiguous color strip.[5] Thus, after this process, we need to consider only a few run-lengths instead of having to deal with the images at the pixel-level. As an optimization, we only encode the run-lengths corresponding to colors that identify objects of interest in the domain. In the robot soccer case, we also omit the colors of the field (green) and the borders (white). Though these colors are extremely useful in detecting the field borders and lines, we achieve that by incorporating a separate and efficient line-detection algorithm (Section 2.4).

To determine merged blobs from these run-lengths, we utilize a procedure that is a variation of the Union-Find algorithm [7]. We merge each run-length with another of the same color as long as they are within a threshold distance from each other. This region-merging operation results in a set of blobs, each of constant color. During the process of region-merging, we also progressively build *bounding boxes* around the merged run-lengths. That is, we develop a rectangular boundary around the regions. This abstraction enables the categorization of each region on the basis of the four vertices of the bounding rectangle. At the end of this stage, we end up with a set of bounding boxes, one for each blob in the current image. In addition to the bounding boxes, we also store a set of properties corresponding to each candidate blob, such as the number of pixels (of the blob color) and run-lengths it envelopes. These properties are used in the object recognition phase (see Section 2.3).

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 very 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.

Our blob formation algorithm serves the dual objectives of incorporating a fast-region growing algorithm while at the same time ensuring that the performance is not compromised. An alternative to this procedure would be to pose this as a pattern recognition problem and extract features (in different feature spaces) corresponding to the desired objects. The objects in the test images would then be detected based on matching the same set of features extracted from the test image. Though this process does provide good performance, it involves computation that is not feasible on our robots.

Even so, blob formation is the most expensive phase of our visual processing system, which along with segmentation takes $\approx$ 20msec per image.

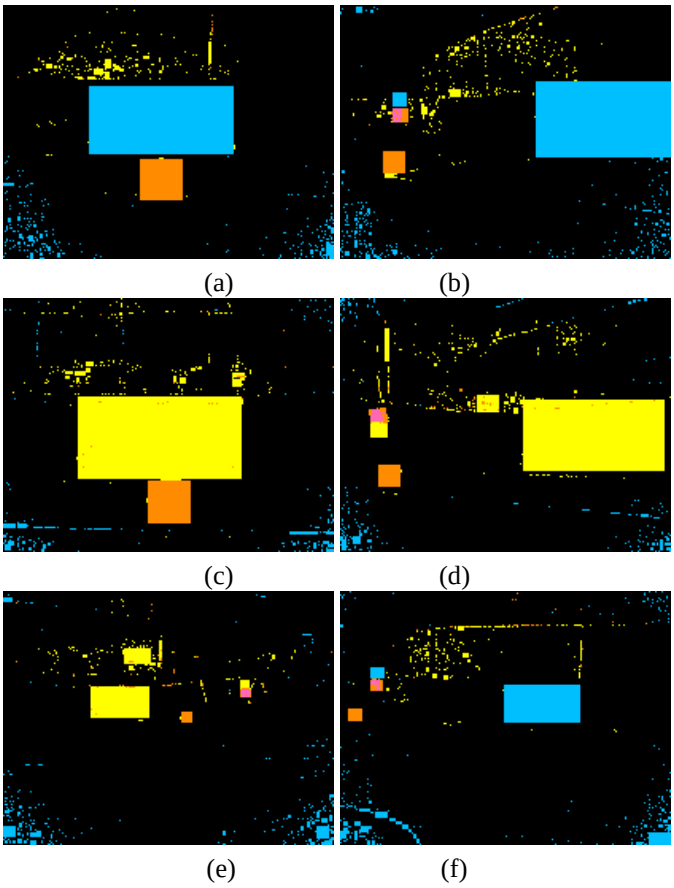


Figure 4: Sample Blobs.

2.3. Object Recognition

Once we have candidate blobs, the next step is to recognize the relevant objects in the image. Object recognition is a well-researched area in computer vision and numerous algorithms have been developed to recognize objects in an image, depending on the application domain [4, 21, 25].

Most of these approaches either involve extensive computation of object features or large amounts of storage in

[5] Further information on run-length encoding can be found in popular image processing textbooks [12].

the form object templates corresponding to different views. Further they are not very effective for rapidly changing camera positions. Constraints on the computational resources and memory render several of these approaches infeasible in our problem domain. We decided to determine the objects of interest in the image from the blobs using domain knowledge rather than trying to extract additional features from the image. This saved a lot of additional computation (and memory).

The objects of interest include the fixed markers which the robot uses to localize itself and the moving objects that the robot has to track. All the objects in the robot's environment 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 the knowledge of the environment. Thus, 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. Also, the ball is mostly enveloped in a square bounding rectangle except when it is partly occluded. All objects — ball, beacons and goals — have bounding boxes with high densities. Due to space constraints, we omit the details of these heuristics. Full details are available in our team technical report [1]. These heuristics are easy to apply since the required properties were stored in the blob formation stage (Section 2.2). Also note that the same properties, such as density and size, are used to determine the probability of occurrence of each object, once it is recognized.

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

As can be seen in the images, we have 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 let us recognize the lines in the environment which are a great source of information. In the next section we shall describe the algorithm that we use to detect the lines.

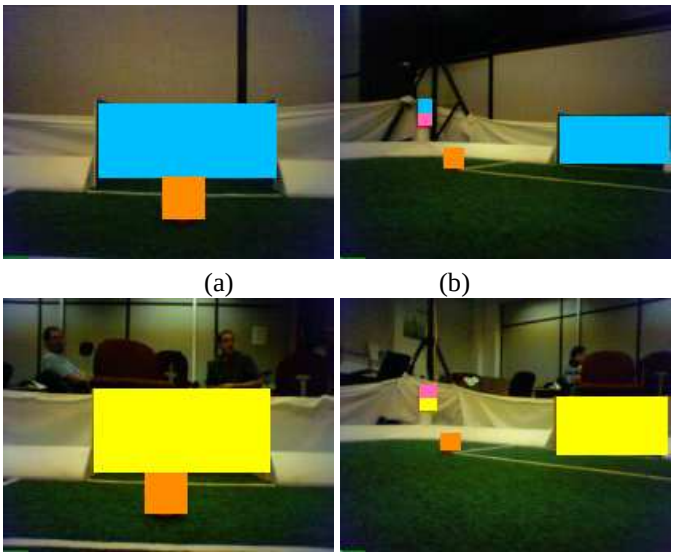


Figure 5: Sample Object Recognition.

2.4. Line/Line Intersection Detection

In addition to the objects that can be represented as fitting in rectangular bounding boxes, lines with known locations can be important sources of information for the robots. Particularly in the robot soccer domain, when the robots are in the process of playing a game, the main focus is the ball and other robots may occlude the beacons and goals. As a result, lines on the field become crucial for localization.

Detecting lines/edges is a very heavily-researched field in computer vision, with methods such as Hough Transforms and edge detectors such as Canny, Sobel [12]. 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 is motivated by a previous approach in the RoboCup environment [19]. To find the candidate edge pixels, we utilize environmental knowledge: 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. Given that information, we could have determined the bounding boxes corresponding to the white blobs and green field and then used heuristics to determine the actual lines/edges. But, as mentioned in the section on blob formation (Section 2.2) we do not even calculate the run-lengths much less store the statistics corresponding to these two colors primarily in an attempt to reduce the computation involved. Our approach is still able to efficiently detect the edge pixels and thereby the lines of interest in the image.

In our line-detection algorithm, we begin, as in [19], by performing a series of vertical scans on the segmented im-

age with the scan lines spaced $4-5$ pixels apart. In addition to making the scan faster, this ensures that we incorporate noise filtering into the process and eventually consider only the lines that extend beyond a few pixels–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 edges pixels by looking for the green-white and white-green transitions. Over this baseline approach, we add features to suit our purposes. 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 plane are accepted *iff* they also have a significant amount of green below the 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 [1]. Then, instead of using these pixels directly as localization inputs, as in [19], we find the lines that these edge pixels represent. Given a set of candidate edge pixels, we *cluster* them into lines in the image plane using the Least Square Estimation procedure [16]. This is an efficient line-fitting 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) on the field 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 they can be very useful in localizing the robot on the field – 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 heuristic thresholds, determined experimentally. For more information on how lines are used in localization, see [1].

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.

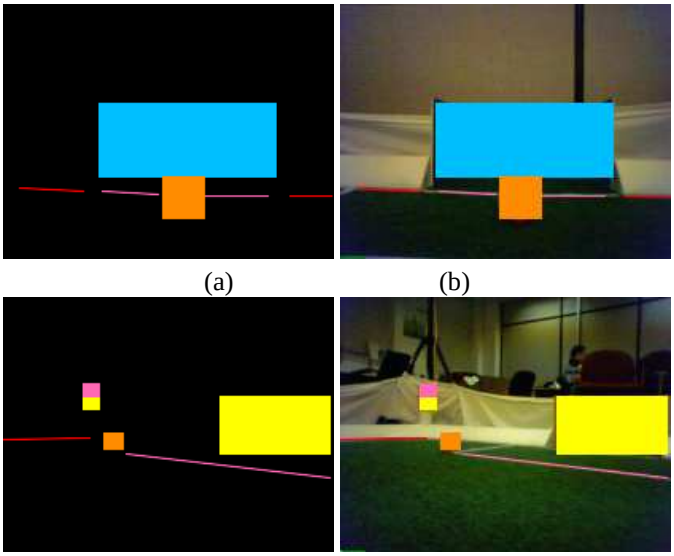


Figure 6: Sample Line Recognition.

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

2.5. Illumination Invariance

To this point, the approach we have described has assumed that the environmental lighting conditions are relatively constant, particularly in the segmentation stage. 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. In this section we present evidence that our approach can be adopted for this purpose.

Color constancy (illumination invariance) is currently a major research focus in the field of computer vision. It represents [5] the ability of a visual system to recognize an object's true color across a range of variations in factors extrinsic to the object (such as lighting conditions). In the past, color constancy has been studied primarily on static cameras with relatively loose computational limitations [9, 8, 20]. Again, our focus is on efficiency. In particular, the overhead required to adjust to dynamic lighting conditions should not impede performance under constant lighting. Lenser and Veloso [17] presented a tree-based state description/identification technique for this same platform. They incorporate a time-series of average screen il-

luminance to distinguish between illumination conditions using the absolute value distance metric to determine the similarity between distributions. We explore an alternative similarity measure based on color space distributions.

In our lab, the intensity on the field varies from the *dark* ($\approx$ 400lux with all lamps except the fluorescent ceiling lamps turned off) to the *bright* (≈ 1500 lux with the additional lamps turned on). We have stage lighting equipment arranged along the edges of the field that allows for gradual variation in illumination between these two levels. Note that given the quality of the camera and image resolution, it is not feasible for the robot to work well 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. As mentioned below, we found that being able to work under these three conditions enabled the robot to function under all conditions in between too. The initial training phase equipped the robot with a color cube and a set of training distributions (drawn from sample images) for each of the three illumination conditions.

The color cube for each illumination was trained by hand-segmenting a set of images captured by the robot under each illumination. The training distributions were obtained by allowing the robot to capture a set (≈ 20) of images under each illumination and generating histograms after transforming from the YCbCr space to the normalized RGB (*rgb*) color space. This is inherently 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 therefore 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 number 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 between them. Since the KL-divergence measure is essentially a function of the log of the observed color distributions, it is less sensitive to large peaks in the observed color distributions in comparison to measures that do not have a similar effect of flattening the peaks. Hence, it is less affected by images with large amounts of a single color. Using this measure the robot was able to correctly classify the test distributions and hence was able to identify and distinguish between the three illumination conditions. The only restriction we had to impose, based on computational constraints, was to test for illumination change no more than twice a second.

We then tested the performance of the robot in illumination conditions in between the three illuminations it was explicitly trained for. Experimental results showed that under these illumination conditions for which the robot had not been trained before, the robot picked the illumination condition (among the three it was trained for) that was *closest* to the test condition (see [3] for full details). This was good enough for the robot to efficiently play a game on the field. For example, 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 ≈ 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. Figure 7 shows the corresponding results, averaged over ten trials.

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

Figure 7: Time taken (in seconds) to *find-and-walk-to-ball*

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 Figure 7 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 after ≈ 1.5sec.

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.

3. 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 not feasible to run

many algorithms in real-time with limited computational resources. Our focus is on developing efficient algorithms.

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. As opposed to previous research using other sensors such as lasers and sonar, the camera has limited field-of-view and image resolution. Furthermore our robots' 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, drawn from the RoboCup legged robot domain and representative of current directions in mobile robotics, enforce the need for efficient algorithms that run at real-time. We have shown that with innovative algorithms and modifications to existing ones it is possible to build an efficient 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.

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 quarterfinals appearance at RoboCup 2004.

Though we have shown results in the robot soccer domain, the techniques developed here can be used in other related robot vision tasks such as surveillance, rescue, and human-machine interaction. The 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. Further details about all stages of our vision processing algorithms are available in our technical [1] and video steams illustrating the process are available from http://www.cppreference.com/cvpr05-aibovision.html.

References

[1] Technical report.

[2] The Sony Aibo robots. `http://www.us.aibo.com`.

[3] I. M. Anonymous and M. Y. Coauthor. Conference paper.

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

[5] D. H. Brainard and W. T. Freeman. Bayesian color constancy. *Journal of Optical Soceity of America A*, 14(7):1393–1411, 1997.

[6] 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.

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

[8] 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.

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

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

[11] 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.

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

[13] 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.

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

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

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

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

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

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

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

[25] 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.

[26] William Uther, Scott Lenser, James Bruce, Martin Hock, and Manuela Veloso. Cm-pack'01: Fast legged robot walking, robust localization, and team behaviors. In *The Fifth International RoboCup Symposium*, Seattle, USA, 2001.