

Object detection in changing environment of middle size RoboCup and some applications

Mansour Jamzad, Ehsan Chiniforushan Esfahani, S.Bashir Sadjad

Sharif University of Technology
Department of Computer Engineering
Azadi Ave. Tehran, Iran jamzad@sharif.edu

Abstract— In this paper, we introduce several novel ideas on robot vision, its implementation on RoboCup and some of its applications. We discuss about a new color model which components are taken from different color models, a fast object detection method using the idea of searching on a few jump points in the perspective view of robot front view CCD camera, a reliable object shape estimation and a fast method to detect a few edge points on field borders for line estimation. Our practical experiments convinced us that in dynamically changing environment such as RoboCup, fast and almost correct solutions to the vision problems are good enough for detecting objects, finding their distance and angle from the robot, collision avoidance and position estimation. This vision system was tested on our robots in RoboCup competitions and very good results were obtained [1], [5].

Keywords: RoboCup, Vision, Object detection, Shape estimation, perspective view, soccer robot, color model.

I. INTRODUCTION

In RoboCup competitions such as Middle size, Small size and AIBO, we have a real time situation where objects are moving around in a known environment. Robots are the moving agents, which should detect all objects in the soccer field with a relatively high speed. Although robots can use different types of sensors such as vision, laser range finder, sonar, infrared, etc, for object detection, but in this paper we describe our experiments in Sharif CE middle size team which only uses the vision sensor. Robots that use vision sensor, will get the best solution if they can detect all necessary objects in their CCD camera frame rate which are usually 25 or 30 frames per second. Although we can find many objects in each frame, but in practice it is not always necessary to detect all of them. Mainly we need to detect ball, robots, wall and the goals. However, in case of a robot, there are many details in it, such as its wheels, body, camera, sensors, its markers, etc. But, we only need to detect the robot as one body. This approach is true for all other objects on RoboCup soccer field as well.

In this research, we propose a novel method for object detection which is based on examining a set of jump points on the image, a new color model and a fast method for object size and shape detection. The mechanics and hardware of our robots are fully described in [1], [2], [3], [4]. The hardware of our vision system consists of a Sony Handycam and a Genius video grabber, which uses BT878 IC. This grabber is mounted on a main board with Pentium, 233 MHz processor. The software is written in C++ under

DJGPP compiler (DJ Gnu Plus Plus) in MS-DOS.

II. PROPOSING A NEW COLOR MODEL

In RoboCup objects are distinguished by their colors. The issue of color coding has been studied for many years, and several color-coding methods have been introduced [6]. Some common approaches are known as *CIE Lab*, *HSI*, *YIQ*, *YUV*, etc. We used to use the *HSI* color model for segmentation, but in practice we found some limitations especially in its *I* and *H* parameters. Since the parameter *I* is the average of R,G and B, it is not a good measure for evaluating the gray level intensity. But the *Y* parameter of *YIQ*, as seen in the following formula, best reflects the gray level intensity of a color image. Therefore it is better to use *Y* than *I* as one of the parameters in color image segmentation.

$$Y = 0.229R + 0.587G + 0.114B$$

After examining several color models to find the one, which gives better segmentation for color itself, we found that the \hat{H} component in *CIE Lab* that is defined as follows, gives a good measure for detecting regions matching a given color [7]. This selection of \hat{H} suits well in RoboCup environment where we have large regions covered with a single color:

$$\hat{H} = \tan^{-1} \frac{b^*}{a^*}$$

Where a^* denotes relative redness-greenness and b^* shows yellowness-blueness [6]. To measure the color saturation, the component *S* in *HSI* gave satisfactory results. Therefore, we propose a new color model named $\hat{H}SY$, where \hat{H} is taken from *CIE Lab*, *S* from *HSI* and *Y* from *YIQ* color model. We implemented this new color model in RoboCup environment and reliable results were obtained.

III. CONSTRUCTING A COLOR LOOK UP TABLE

To detect colors in real time speed, we do a color training stage for our robots. Before the start of game, we put the robot on the field in different angles and locations, grab frames by its CCD camera, freeze them and construct a look up table (LUT) manually. This (LUT) can assign a color code to a given RGB value. We define nine color codes, eight of which are for standard colors in RoboCup such as red, green, blue, yellow, black, white, light blue, purple and one unknown color for all the rest.

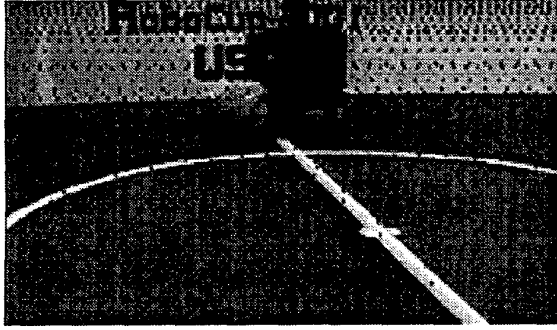


Fig. 1. Position of jump points in a perspective view of robot.

By looking at a freezed image on the monitor, for any object, for example, the green carpet on the field, we manually draw one or more rectangles on it. For all pixels that lie inside these rectangles, the range of $\hat{H}SY$ parameters are determined and all pixels in the image which $\hat{H}SY$ value lie in that range are painted with green color and displayed on monitor. If we find this segmentation satisfactory, then these ranges are saved for green color code assignment in LUT. If not, we will select rectangles from areas that should be segmented green but are not. This procedure is repeated, until we get a satisfactory segmentation result. The above routine is repeated for all eight colors.

At the end of this step, we have a LUT such that given the $\hat{H}SY$ values of any pixel, it returns its color code. During the game, this LUT is used to determine the color of pixels.

A. Jump points in a perspective view

In a CCD camera all objects are seen in perspective, which is, the far objects are seen small and those closer are seen larger. Figure 1 shows a set of points displayed in perspective on an image. These points are called "jump points". They have equal horizontal spacing on each row. The vertical distance between these jump points is set in such a way that, for the smallest object in the field, which is the ball, at least 5 jump points (3 horizontal and 2 vertical) is located on it, no matter how close or far the ball is located on the soccer field.

In general, selection of the number of jump points and their spacing on an image plane depends on the physical size of the smallest object to be detected and also the depth of camera view, such that, independent of object distance from camera, always only a few jump points can be located on the object. However, the overall speed of object detection depends on the complexity of calculation need to be performed on each jump point.

IV. OBJECT DETECTION IN A NON SEGMENTED IMAGE

In RoboCup, we think it is worth getting fast and almost correct results rather than slow but exact. Therefore, unlike the common image processing routines that define a segment to have a well defined border with its neighboring

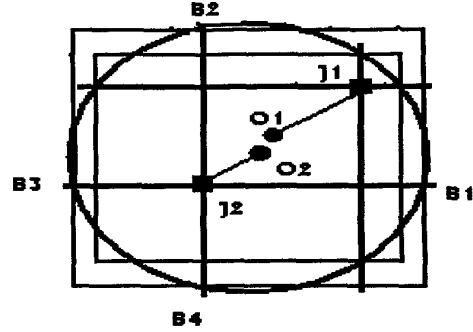


Fig. 2. Estimating a rectangle around the ball.

areas [8], the segments that we define always have rectangular shapes. We think there is no need to know the exact shape of a robot during the game, because a robot shall always keep a distance from other robots, therefore, an estimation of their shape would be sufficient. The simplest shape estimate is a rectangle bounding box drawn around the robot. This idea is implemented for all other objects such as ball, goals, etc.

Therefore, our vision system assumes there are always 2D rectangular objects moving around on the soccer field and performs its calculations accordingly. For any rectangular object, we return its size, color, the coordinate of its upper left, lower right and the coordinate of a point Q which is located on the middle of the lower side of the rectangle. This point Q is used for calculating the distance and angle of the robot from this rectangle [5].

The idea of jump points helped us to find these rectangles without doing any color segmentation on the entire image frames, but only by checking the color code of a few pixels, some of which were jump points.

A. Rectangular shape object detection

Assume we want to find the ball, that is the only red object in the field. We scan the jump points from lower right point towards upper left corner. At each jump point, the $\hat{H}SY$ values of that pixel is passed to the LUT and its corresponding color code is returned. If this color is red, then this jump point is located on the ball. From this jump point, we move toward right, left, up and down, checking each pixel for its color. As long as the color of pixel being checked is red, we are within the ball area. This search stops in each direction, when we reach a border or limit point, which is a non red color pixel. In checking the pixels on the ball, we perform a noise estimation to make sure the point has the right color.

An illustration of this method is shown in figure 2. Two jump points (i.e. j_1, j_2) are shown on the ball. From each jump point, four straight lines are drawn toward the border point of ball. A rectangle is drawn from these four border points, for example, points B_1, B_2, B_3 and B_4 . The length of line segment connecting that jump point and center of this rectangle is calculated. Line segments O_1J_1 and O_2J_2 are two such examples. At a jump point J_i , if the length

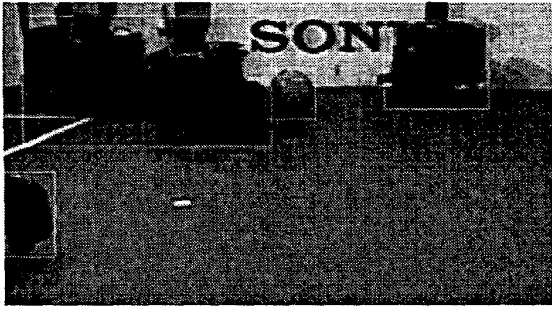


Fig. 3. Objects estimated by rectangles.

of $O_i J_i$ is less than a threshold T , then its corresponding rectangle is accepted as an estimate for the ball. However, as soon as such rectangle is found at a jump point, we will terminate the search for ball. In figure 2, for simplicity of drawing we showed the procedures with only two jump points. The larger rectangle is taken as the final estimate.

Nevertheless, we can implement a much faster linear search to detect the border points of an object. For example, in moving toward right, instead of jumping in step of one pixel, we can jump in step of 5 pixels or more, depending on real size of object. The larger the object the bigger the jump size. However, in this way most of the time, in the last jump we end up entering outside object region. To overcome this problem, we move in the opposite direction, in step of one pixel, until reaching the border point. Similarly, the bounding box around all other objects are calculated. Figure 3. shows how a few objects are estimated in this method.

In this method of search for objects, in one scan of all jump points, we can find all objects such as robots (our team or opponent team robots), yellow goal, blue goal, etc., in the image. For each object we return its descriptive data such as, its color, size and the coordinate of its lower left and upper right corner of its surrounding rectangle and a point Q on the middle of its lower side. Point Q is used to find the distance and angle of the robot from that object [5]. Our software can extract this information from an image in about $\frac{1}{50}$ of a second.

V. STRAIGHT LINE DETECTION

During the game, there are many times that the robot needs to know its distance from walls or border lines. But, the goal keeper in all times, needs to know its distance and angle from the field borders and also from the straight white lines of penalty area.

Hough transform [8] is a common approach to estimate a straight line passing through a few edges. However, the usual methods of filtering for detecting edge points is very time consuming and may not be practical. In RoboCup where most robots use commercially available PC processors.

To solve this speed problem, we propose a fast method for straight line detection. To detect the edge points on

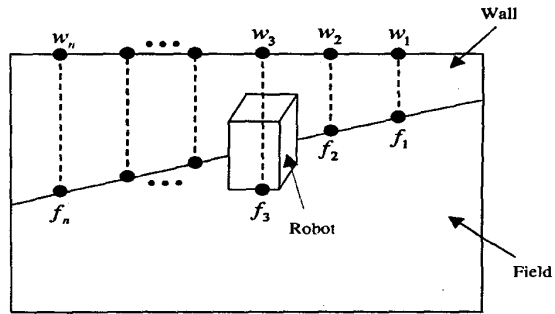


Fig. 4. Wall and field edge point detection. Straight lines w_i, f_i show the drilling path.

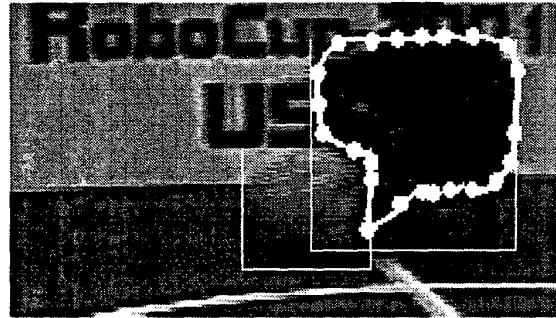


Fig. 5. Detecting border points in clockwise direction radiuses.

the border of wall and field, we assume to drill the wall downward in a few locations until hitting the green field. These points on the field are the edge points passed to the Hough transform. As it is seen in figure 4, in our algorithm, we select a few equally spaced points on top of an image w_i , and move down, pixel by pixel or by jumping in a few pixel steps, until hitting the green field, f_i . A green point such as f_1 or f_2 is accepted as an edge point, if it is below a white point (wall). By this criteria, point f_3 is considered to be noise.

VI. APPLICATIONS

The idea of searching for objects on a few jump points in perspective can be applied to AIBO (Sony legged league of RoboCup) and also to AGV (Automatic Guided Vehicles) systems. In addition, in small size league of RoboCup, where we have centralized top view with no perspective, we can distribute these jump points in equally spaced positions in vertical and horizontal directions, as if they are located on the corner points of a mesh. The size of this mesh can be selected such that at least 2 or 3 jump points will be located on the ball, that is the smallest object in the field. Object search criteria can be the same as described in this paper.

We can extend the idea of rectangular shape estimation to a curved shape estimation. As it is seen in figure 5, let's have a rectangle around a robot, from the center point of

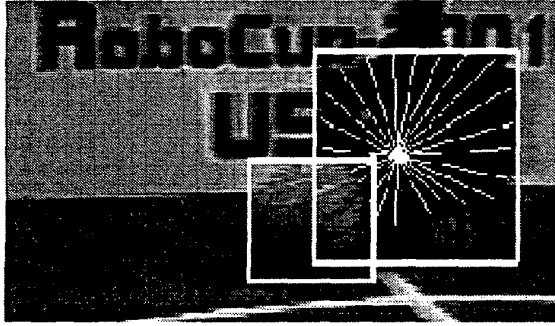


Fig. 6. Shape estimation by drawing a Bezier spline through control points.

this rectangle, on radiuses spaced in 45 or less degrees, (depending on the application and the exactness of shape that we want to estimate) we move towards the rectangle edge, until hitting a border point. These border points are taken as control points and passed to a spline curve drawing algorithm which draws a closed spline on these control points, as shown in figure 6. This closed spline curve is an estimate for object shape and can be used in related applications. However, to have a correct sequence of control points for spline curve, we shall select the radiuses in clockwise or counter-clockwise order.

In addition, we can think of many applications for our drilling idea in detecting edge points. For example to detect the border of mountains and the sky, as it is shown in figure 7, we have passed a Bezier spline curve between the drilled edge points to estimate the border. Another application can be to estimate the border of sea and the sky when we want to find the horizon line. Also, it can be used to find border area between two regions with relatively distinguished colors.

VII. DISCUSSION

We know that mechanical and hardware control stability and reliability are the essential factors in having a reliable robot. However, in RoboCup and many other mobile robot applications, we need to have fast and accurate software routines to make the robot react properly in a rapidly changing environment. In our robots, we only use vision sensor to extract all necessary data needed for our software. Therefore, our vision programs should be able to provide these data to software decision making routines at least in real time rate.

Our successful experiments in RoboCup games in the past years, convinced us that, fast and almost correct solutions to the vision problems, works well compared to the exact but time consuming routines. After all, in real world, when we recognize objects, in the first look, we recognize them as a whole, but not with all details. Of course we will examine the details only if we want to. Let's take the case in real soccer. The human player, when looking at a distance, only needs to know the position of ball, goals and players with a relatively accurate estimate, and does not

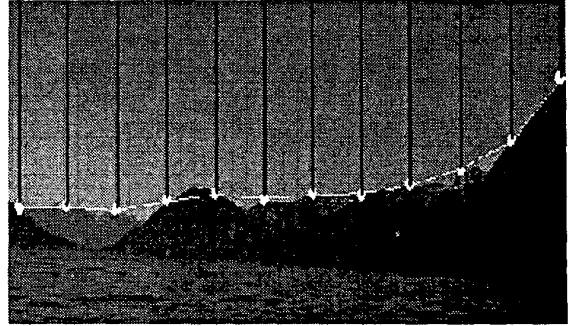


Fig. 7. Passing a spline curve through the border points of a mountain and sky.

go for much details in shape, color, etc. Therefore, the idea of using jump points in a perspective view to estimate object shapes by rectangles, provides the necessary accuracy in relatively high speed. As a result, the PC processor has enough time to run the decision making routines and the overall vision and decision making software could work in real time speed.

VIII. ACKNOWLEDGMENTS

I would like to thank a lot, my students research team in mechanics, hardware and software, who have worked hard in the past few years to construct the robots. In particular, I would like to name students in software group A.Hadjkhodabakshi, S.V.Mirrokn, A.Heydarnoori, T.Hadjjaghahi, M.Ebraahimi Moghadam and S.Rastkar, who helped to implement some ideas presented in this paper on our robots. I am very grateful to all our sponsors and also I should give my sincere gratitude to all top ranking managers of Sharif University of Technology whose serious support made this work possible.

REFERENCES

- [1] M. Jamzad, et al, *Middle sized Robots: ARVAND*, RoboCup-99: Robot Soccer world Cup III, Springer (2000), pp 74-84 .
- [2] M. Jamzad, et al, *Design and construction of a soccer player robot ARVAND*, RoboCup-99: Robot Soccer world Cup III, Springer (2000), pp 745-749 .
- [3] M. Jamzad, et al, *Basic requirements for a teamwork in middle size RoboCup*, To appear in RoboCup-2001: Robot Soccer world Cup V, Springer (2002).
- [4] M. Jamzad, et al, *ARVAND A Soccer Player Robot*, AI Magazine, Volume 21, No.3, Fall 2000, pp 47-51.
- [5] M. Jamzad, et al, *A fast vision system for middle size RoboCup*, To appear in RoboCup-2001: Robot Soccer world Cup V, Springer (2002).
- [6] S.J Sangwine and R.E.N Horne, *The color image processing handbook*, Chapman and Hall (1998).
- [7] Y. Gong, and M. Sakauchi, *Detection of regions matching specified chromatic features* , Computer vision and Image Understanding, 61(2), pp163-269, 1995.
- [8] R.C. Gonzalez, and R.E. Woods, *Digital Image Processing*, Addison-Wesley, 1993.