

Visual Object Recognition in RoboCup

Horia Vlad Balan

May 6, 2004

Abstract

This paper constitutes a presentation of the methods used by the IUB RoboCup team for visual recognition, comprising a description of the system architecture and of the image processing algorithms. The problem is first formulated and then a state of the art implementation is presented.

New innovative methods are discussed together with their results.

Contents

1	Introduction	2
1.1	Machine Vision - tracking and recognition	2
1.2	Vision in the RoboCup Small Size League	2
2	System Design	4
2.1	Hardware Architecture	4
2.2	Software Architecture	4
2.3	Processed Information	5
2.4	Modular Software Design	5
3	The Recognition Algorithm	8
3.1	The Data Flow in Recognition	8
3.2	Blob Recognition	8
3.3	Light Maps	10
3.4	Presence Maps	11
3.5	Object Identification	11
3.6	Coordinate Change	12
3.7	Industrial Applications	13
4	Results	16

1 Introduction

1.1 Machine Vision - tracking and recognition

The continuous increase in computing power in the last decades caused the emergence of machine vision as a subfield of increasing importance of computer science. Motivated by industrial or military interest, the research in the subject was encouraged, and results appeared.

The history of machine vision begins in the 1970s when mainframe computers were first used in conjunction with image capture and display devices, making possible the development and testing of the first algorithms. In the next decade, dedicated workstations for image manipulation entered the scene, featuring hardware dedicated to accelerated image processing. In the 90s, it appears that generally available hardware started to include most of these characteristics, rendering specialized solutions obsolete. It became thus possible to work on visual recognition tasks using standard, relatively inexpensive components.

The main purpose of a machine vision application is the visual inspection of a domain, in a quite narrow setting. Typical examples include industrial functions such as bar-code reading, optical character recognition, counting objects, directing tools through sensors. As opposed to computer vision, a machine vision application generally does not try to obtain a model of the surrounding environment, but acts under a very restrictive set of assumptions. Industrial activities such as semiconductor device fabrication, industrial robotics and automated surveillance are some of its main beneficiaries.

One of the methods traditionally used is the marking of objects for visual identification. In this way a rather complicated arbitrary pattern recognition activity is replaced with a search for the most convenient available pattern, with the option of increasing the quantity of information presented and/or the probability of correct detection. Such implementations do not require complicated algorithms and can be achieved using simple, high speed hardware such as programmable logic devices linked to CCD sensors, obtaining speeds of thousand of frames per second.

The main challenge present in the field today is trying to cope with difficult or varying environmental conditions, while providing a good rate of correct detection. The efforts of numerous research groups are concentrated on creating robust systems, as well as generally applicable methods.

1.2 Vision in the RoboCup Small Size League

RoboCup started in 1997 as an educational and research initiative targeting the development of the fields of artificial intelligence and cooperative robotics. It offers a specific problem, namely playing the football game, in the solving of which significant advances for both fields are expected to be made, due to its characteristics such as a dynamic environment, real-time decisions, incomplete information access, ambiguous sensor readings, and the necessity for distributed control. Its main event is the annual Robot Soccer World Cup, completed by regional events, conferences and educational programs. The proclaimed objective of this initiative is to achieve an expertise level making it possible that by the year 2050 a humanoid robot team will be able to successfully challenge the

best human football team.

Due to the variety of approaches and possibilities in constructing a solution system, and in order to provide objective differentiation during competitions, a system of leagues was developed. Each robot league fixes the dimensions of the robots, the ones of the field, the degree of autonomy and the perception methods to be used, as well as sometimes more advanced traits of the players. There are five football leagues and one rescue robots league.

The small size league games take place on a field of 4.9m * 3.4m between teams of five robots having a maximal diameter of 180mm, having color markers on their top area. The rules are inspired by the official FIFA Rules of the Game, with modifications due to the nature of the event, and with an accent on preserving the autonomy of the players.

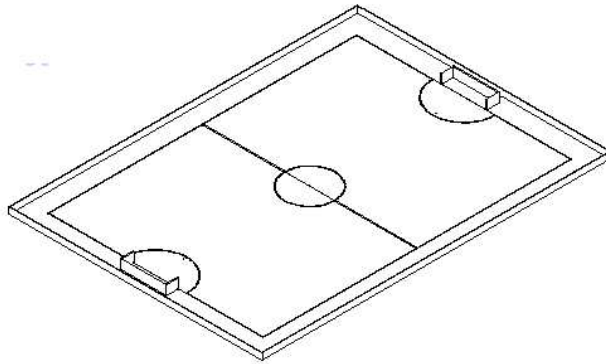


Figure 1: Field sketch (taken from www.robocup.org)

The small size of the robots makes it very hard to have localized visual sensing, calling therefore for an external perception method, namely a camera placed by each team above the field surface. This in turn is connected to a processing computer, which sends the data to the players.

Our interest lies in the realization of this visual recognition. The main characteristics of a successful recognition system are robustness and performance, both of which we tried to achieve.

2 System Design

2.1 Hardware Architecture

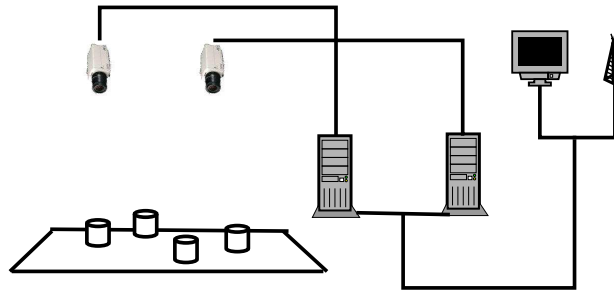


Figure 2: Hardware diagram

The hardware cycle is comprised of:

- camera
- image acquisition
- image processing
- primary data transmission
- data merging
- data presentation
- secondary data transmission

We are using a pair of *no-name* cameras, designed for industrial applications, which provide an S-VHS analog image output. Each image serves as input to a Philips BTB848 *exact model* frame grabber, installed on a PC platform. The two PCs are 1GHz Pentium III machines, with 256MB of RAM, each running FreeBSD version 5.1. They perform the image processing and transmit the resulting primary data through a 100Mbps network as UDP packets to a single computer, running a program responsible for data merging, visualization and secondary data transmission. The secondary data is sent through a serial connection to a RoboCube equipped with a wireless transmission board, capable of sending data through a serial protocol to the robot team.

The main difficulty that persuaded us to separate the recognition cycles on two distinct machines was the impossibility to guarantee independent access to multiple video grabbers on the same machine. The source of this problem could be a limitation in the operating system kernel, or one of a hardware nature.

2.2 Software Architecture

The software has been developed under FreeBSD 5.1 using the GNU compiler tool-chain for C++. It is organized in a modular fashion, which allows it to profit completely of the advantages of object-oriented design. The code has been documented using the Doxygen utility while the present report serves as primary

reference source. The main components of the software and documentation are as follows:

Component	Description
RVLib	objects library
Camera	image processing
Beamer	data visualization
Documentation	reference articles code reference report

Components table

2.3 Processed Information

The primary data consists of blob information indicating the important zones of color on the field and it has the following structure:

Frame	frame_start					
	<table border="1"> <tbody> <tr> <td>Blob</td> <td>int lower_margin int upper_margin int left_margin int right_margin int identifier</td> </tr> <tr> <td>...</td> <td></td> </tr> <tr> <td>frame_end</td> <td></td> </tr> </tbody> </table>	Blob	int lower_margin int upper_margin int left_margin int right_margin int identifier	...		frame_end
Blob	int lower_margin int upper_margin int left_margin int right_margin int identifier					
...						
frame_end						
...						

The secondary data consists of robot position and orientation values and it has the following structure:

Frame	frame_start					
	<table border="1"> <tbody> <tr> <td>Robot</td> <td>int position_x int position_y float orientation int identifier</td> </tr> <tr> <td>...</td> <td></td> </tr> <tr> <td>frame_end</td> <td></td> </tr> </tbody> </table>	Robot	int position_x int position_y float orientation int identifier	...		frame_end
Robot	int position_x int position_y float orientation int identifier					
...						
frame_end						
...						

Due to their correspondence to the physical entities searched for, these data structures are a natural choice for our recognition algorithm.

2.4 Modular Software Design

In our latest implementation, we opted for a modular architecture which can approximate the data flow of a processing scheme, using modular components interconnected only in the last steps of the design. We obtain thus very simple final applications, and a very good separation between the generic objects offered and the way in which they are used. The main source of inspiration for this approach was the Microsoft Direct-X standard which provides a so-called filter system for the Direct-Show component, based on inter-connectable

distinct objects offering limited just by the interfaces provided. While we did not have at our disposition a marshaling mechanism such as COM or CORBA, nor the intention to use any, the solution chosen was to develop a templated plugin class, in such a manner as to insure through compile-time checks direct interconnection.

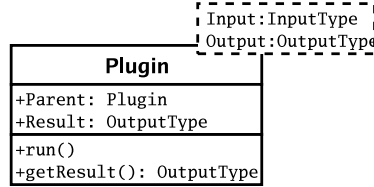


Figure 3: Plugin Structure

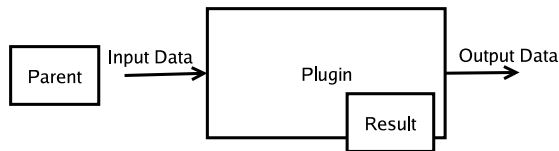


Figure 4: Plugin schematic

A plugin derived from the Plugin class is parameterized by its input data and output data types. Presumably it's accesses it's parent plugin's (the provider of it's input data) *getResult()* function, which is constrained to the type of the input data, processes it and stores the result locally in an output type variable, accessible through it's output type *getResult()* function. Therefore type checks for this mechanism are enforced during compilation and none of the template mechanism's type uncertainty is felt. There are however limitations, such as the fact that one plugin can only access data from one source. However, a more general approach would have increased dramatically the overall complexity of this approach.

The plugins developed were:

CGrabber - A plugin responsible for effecting the frame grabbing operations and returning a memory pointer to the bitmap containing it. The grabbing is done through device operation (*ioctl()* calls) while the access to the image is through direct memory access. The main source of inspiration in writing this plugin were the documentation and examples provided in [5] and [6]. Our earlier Windows version used the Microsoft Vision SDK and an intermediate DLL created in Visual C++ for the same purpose.

CBlobRec - A blob recognition plugin working after the algorithm described in the next section and returning a list of candidate blobs

CSender - A plugin responsible for sending the blob information through the network using the UDP protocol.

CReceiver - A plugin responsible for receiving blob information from several hosts on different ports.

CDisplay - A plugin responsible for displaying the data in a composite form

These components were compiled as object files which were added to a single library (RVLib) making it possible for the different applications running to have

a common base. We organized the software project's makefiles in such a manner that each modification to a base object would determine the rebuilding of the library and recompilation of all applications using it. In this manner we obtained a fast development/testing cycle, almost as good as the one specific to Rapid Application Development (RAD) environments.

For recognition, three separated processes running in parallel are needed, namely two instances of the blob recognition program (Camera), which occupy most of the processor power, and one instance of the data integration and presentation program (Beamer). Since lost packets were not of the highest concern in designing the communication system, we decided on UDP to be the fastest available network protocol over which these processes could communicate (pipes were not an option, since the processes are running on different hosts).

3 The Recognition Algorithm

In our implementation we chose one of the most used marker recognition algorithms in the RoboCup community, namely the one based on blob identification and enlargement. Our object identification algorithm is based on the geometry of the markers and it bears similarities to the one discussed in [3].

3.1 The Data Flow in Recognition

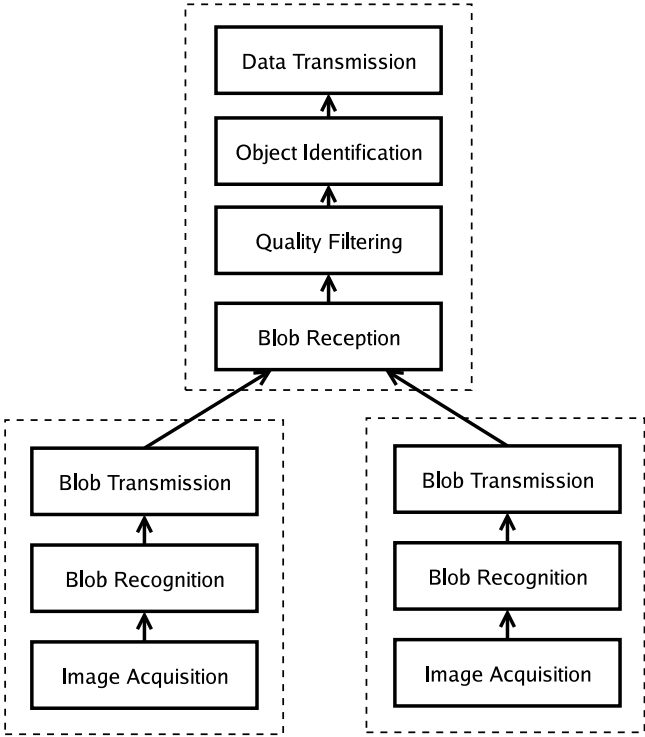


Figure 5: The flow of data in the recognition algorithm

The flow of data takes place as indicated in the above diagram. Each dotted box represents an independent computer process, two of which are clients (recognition processes) and one of which is server (data processing, displaying and communication process).

3.2 Blob Recognition

The simplest and one of the most efficient methods of recognizing visual markers in an images is the so-called Blob Recognition. Since in RoboCup the markers appear in the images as contiguous zones of the same color and a certain dimension and shape (blobs), and we can simply try to identify such zones for all colors of interest.

This implementation relies on a distance function for comparing the colors of different points, for example the sum of the absolute differences in the red,

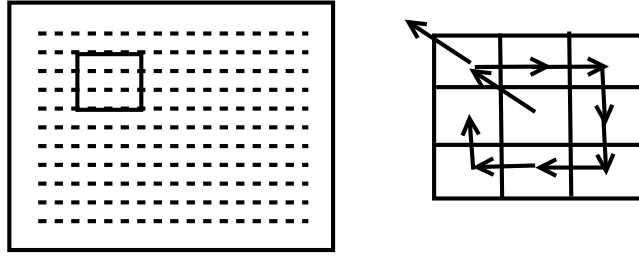


Figure 6: Blob Recognition

green and blue components.

The identification comprises just two steps:

Identifying the presence of a color on the image

Obtaining the dimensions of the color zone

In order to find the occurrence of specific colors in the image, we traverse it from top to bottom left to right at a step comparable to the assumed size of the color zone. When the distance between the color of a certain point and one of the searched colors falls under a certain threshold, we examine the image starting with that point using a blob enlarging algorithm. This is a recursive search through the neighbors of the current point, conditioned by the color threshold and using an image mask in order to avoid repetition. Finally we obtain a list of blobs, which can be sorted according to some quality criteria, in our case the size correspondence and from which the best candidates can be extracted.



Figure 7: Blob Recognition example

In other implementations (see [4]) a segmentation algorithm is applied in order to divide the image in a number of disjoint planes corresponding to the searched colors. Afterwards, a recognition algorithm based on the proximity of a number of color zones can be applied in order to identify marker patterns.

3.3 Light Maps

The most frequent difficulty encountered while using such a recognition model arises from the lack of uniformity of illumination on the field. We can expect that the same marker, in different positions on the field, will receive different quantities of light, and therefore will be perceived differently. One solution to this problem would be using a color model which separates the intensity component of the color from the other ones (for example the HSI color model), and ignoring the intensity value. However, it appears that the saturation and hue components also change under different intensities, and therefore we cannot achieve the desired results.

Our solution to the above mentioned problem consists in splitting the field area into squares sufficiently small in order to consider the illumination uniform on their surface, measure the value of the color markers when situated in that region, and in this way construct a so-called color map, which can be later used as reference in the process of blob recognition.

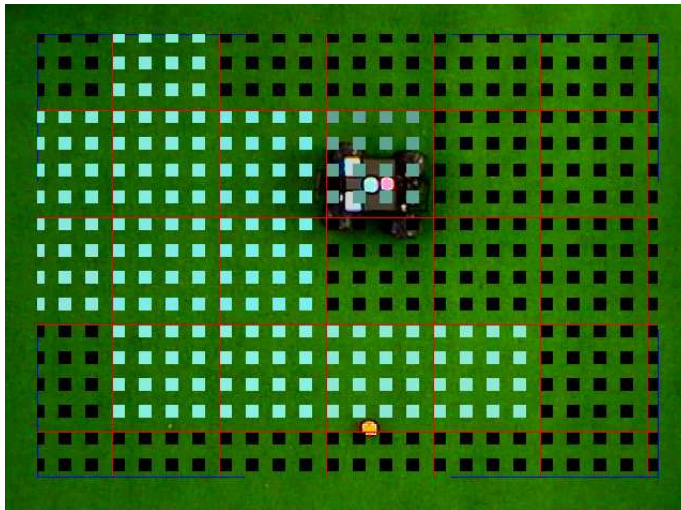


Figure 8: Light Maps example

This method can be used with good results in practical conditions, however our implementation did not focus on creating a fast color to light adaptation scheme, or achieving in-game calibration. From our knowledge, none of the competing team has achieved the level of stability necessary for the realization of this second goal. Nevertheless, the problem remains of practical importance.

Another idea presented was the usage of a combination of several color models, such as RGB and HSI in measuring color distance. While HSI can be theoretically less sensible to changing light conditions, we found that in practice the behaviour of the measured color's Hue and Saturation components under different light intensities is far from constant.

The paper [3] describes a similar implementation, with the mention that their team has implemented semi-automatic calibration, i.e. the vision system tracks a single robot moving through the field in a random pattern and locally changes the searched color values under the supervision of an operator.

3.4 Presence Maps

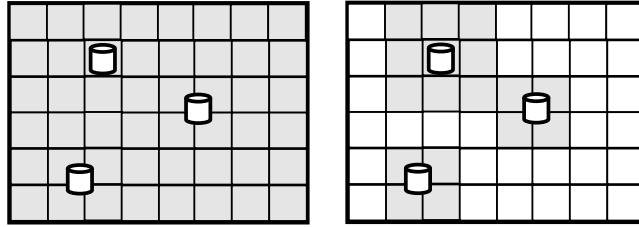


Figure 9: Presence Maps

While we did not use it in our implementation, we feel obliged to discuss a performance acceleration method commonly used in the league, especially due to the impact that it can present on the subject of pattern recognition.

Consider dividing the field in square areas, as in the case of light maps, and assigning to these areas a bit map. After processing a whole frame, we can mark the areas in which objects were found, and in the processing of the next frames limit ourselves to the processing of these areas and of their immediate neighbors, for as long as the recognition performance satisfies us.

In RoboCup, the gain in performance can be computed as the percent of squares in which robots are situated, together with the neighboring squares. For example, when dividing the field area into $40 * 20 = 800$ zones, the computation time decreases to at most $9(\text{neighbours}) * 11(\text{objects}) / 800(\text{zones}) = 12.5\%$. Each of these zones would have a side length of 16 pixels, sufficing to comprise a whole object.

The paper [3] discusses these technique and presents very good performance results (30% CPU time on a 300MHz Pentium machine, running at 30fps).

3.5 Object Identification

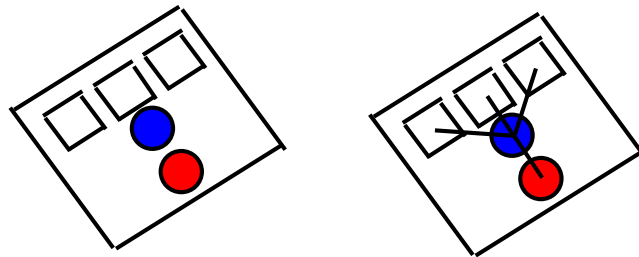


Figure 10: Object Recognition

We use our a priori knowledge of an object characteristics in order to achieve efficient object recognition. Basically we try to match central markers to orientation markers and then verify the identity of the player. In order to simplify our algorithms, it was decided to use both in the exposition and in the implementation the notation of vectors identified to variables in the complex plane. The meaning of the notations used should be immediate.



Figure 11: Object Identification example

To each central marker we assign the closest orientation marker and compute a vector v from the it's center to the center of the orientation marker. We compute the vectors $-v$, $(-1-i)*v$ and $(-1+i)*v$ in order to obtain the positions of the identification markers. The next step is to compare the intensity value of the color of these points to a certain threshold, in order to assign them a value of 1 or 0. A 3-bit code is thus obtained, allowing us to assign to each player its number. We can therefore complete the player data structure with all the informations.

An important advantage in using vector calculations is that they are, to a certain extent, “tilt invariant”. We have tested our recognition process in the case of video filmed with a camera mounted on the side of the field, and we obtained correct identification even with the camera at a height of only one meter. Blob recognition, of course requires some change in order to cope with the change of perspective.

Different error correction methods have been proposed in order to compensate for the occasional “gaps” in the blob recognition process. While we have not used it ourselves we mention the method discussed in [3], namely using a line of team identification/orientation markers in three colors, therefore obtaining recognition in a two out of three detection scenario. The paper [4] describes a four color marker pattern placed around a central identification marker. This pattern is used for orientation and numeric identification.

3.6 Coordinate Change

The coordinates that we obtain following image processing are relative to the pixel dimensions of the image. In order to obtain physical coordinated an affine transform is necessary and useful prior to sending the resulting data to the robot team. The coordinate system of the final set of data is usually relative to the center of the field.

The second step involved is compensating for the distortion of the lens,

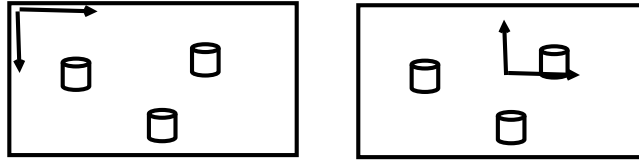


Figure 12: Coordinate change

especially important when we are using two cameras and we need to harmonize their data. This can be done in a variety of ways, from inverting the distortion curve of the lens to constructing a set of points with known physical values in the image space and interpolating from them the physical location of the blobs. A common method of achieving this is covering the surface of the field with a certain repetitive pattern and running an identification algorithm in order to find a set of edge points.

A calibration of the coordinate transformations used is necessary every time when the camera is repositioned. In practice, we encountered a number of problems due to the fact that it is not possible to ensure that the camera will be in center field position during every match. An automatic calibration method based on markers and known positions has been considered.

3.7 Industrial Applications

The relative simplicity of the algorithm used makes it possible to consider a deeper integration into hardware, and the creation of some industrial devices based on it. In what follows we will give a description of the main modifications necessary for this purpose.



Figure 13: CCD, FPGA chip and embedded computer chips (www.hwwtech.com and www.beck-ipc.com)

A basic industrial device for visual inspection consists of a CCD connected to a FPGA programmed as to perform the work of a number of image processing filters, and giving output information. The whole ensemble can run at the speed of several MHz, giving this as the frame per seconds processing rate. In order to achieve this speed, we must use the intrinsic parallelism of the device. The algorithms which we have presented so far are not parallel, but we will show the necessary modifications in order to localize them and obtain corresponding local filters.

The first step in the recognition process is thresholding against the value of each color, and segmentating the image into six color planes, as described in [4]. This operation can be done independently for each pixel, and after it we obtain an associated image map which contains the pixels for which the color falls under a certain threshold from a searched color, as well as the identifier of that color. The next operation is the blob enlargement which can be done in the following way: consider initially each identified pixel as a blob of size one and assign it an unique blob identifier based on it's height and width position (the pair (width,height)). Order these blob identifiers first after the width, then after the length. Repeatedly take all pairs of neighboring blobs and unite them when they have the same color identifier, also assigning them both the lowest color identifier in one variable and the highest one in another. In this way the identifier of the lowest and of the highest point and from a blob propagates throughout the whole blob.

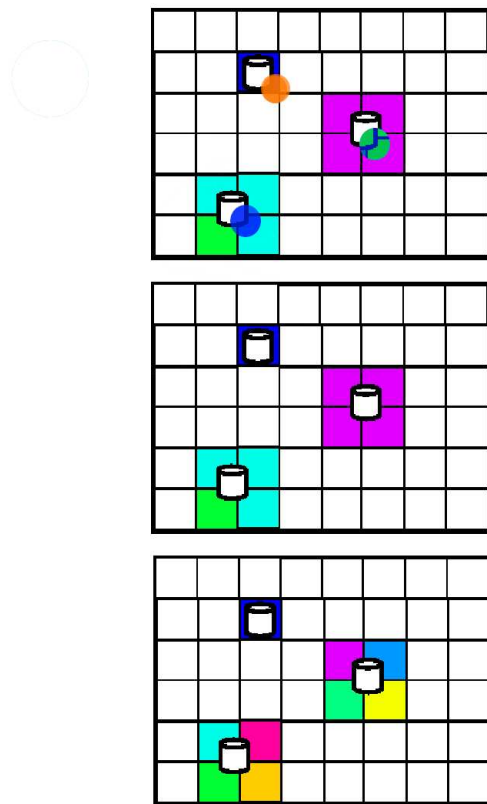


Figure 14: The industrial version of the algorithm in a step-wise, bottom-up presentation

In the last step by comparing each pixel's low blob identifier with the high one, we can determine the size of the blob to which it corresponds. Since all pixels in the blobs have the same high and low identifiers, we can finally obtain a list of all blobs, and of their sizes, which can be used for latter processing.

The rest of our algorithm does not offer advantages in a parallelized form, but it's complexity is negligible in comparison to the one of blob recognition.

Due to the digital nature of our processing some very general signal detection techniques can be applied. It can be useful to examine a set of measurements instead of individual ones, obtaining thus a dramatic decrease of the probability of false alarm.

The proposed solution is both economical and easy to implement. A 1024*1024 pixels CCD can be combined with an FPGA and a small embedded computer such as the BECK IPC 186-chip (www.beck-ipc.com) in order to provide a full implementation for a very small cost. Moreover, the IPC chip offers several networking interfaces, and can be programmed using a generic C compiler for a DOS-like operating system, giving thus the option of connecting to a larger system and of extension. The possibilities of finding applications in areas such as robot direction, robot arm positioning or even object detection are evident.

Our interest in industrial solutions is by no means singular. For an overview of the state of the art in this domain, we invite the reader to conspect the datasheet of the vision system described in [7].

4 Results

The results presented in this section refer mainly to the system constructed for RoboCup 2002, which was intended for usage in a competition.

We obtained a visual recognition system offering, after calibration, a reliable rate of identification and a fast processing rate. Our calibration sessions were on average of ten minutes length, and they involved measuring the color response of the markers on the whole field. After this lengthy process, we would expect that we would have a misidentification of a robot only once every few seconds, so only in a percentage of the total number of frames. Our peak value for processing was at 25fps on a fast machine.

The robots move with a velocity of approximately $1m/s$, giving us an interval of at most $4cm$ between the positions in two successive frames. Moreover, our camera's $640*480$ resolution only allows covering about $1/4cm^2/pixel$. Vision cannot therefore be used for small scale movement direction, but only as a completion of the sensors on the players. Localization information coming from vision can be used in order to correct the time-additive error specific to the robot's dead-reckoning system.

Although vision cannot be used as the only mean of orientation in a game, it has been used successfully in experiments involving robots moving at slow speeds, such as solving motion planning problems. For this purpose an independent data communication system was developed by one of the graduate students at IUB.



Figure 15: Team Recognition example

Our system is capable of successfully tracking a set of eleven object on the field (two teams and one ball), for the period of one game. We surpassed the difficulties related to changing light conditions, high speed movement and stability of identification.

We have to point out that the methods described are some of the simplest and most efficient to be encountered in the field of visual analysis. This is to be explained by the hunger for speed of our application. To our knowledge, these

methods are largely applied by most RoboCup teams, and certainly by those producing team descriptions and scientific articles every year. Since the high performance achieved by a number of implementations is recognized by now, the RoboCup federation tries to change the focus of scientific research by ending the life time of the RoboCup F180 league and introducing a local vision-based league.

We can infer that the vision system that we built is competitive with the ones of the main challengers for the title, and is using state of the art methods developed in this field. We are aware that improvement are still possible and desirable, and it has been always our aim to improve the practical performance of our system without losing it's simplicity. Even if the current vision system will not be furthermore extended, it can be used, together with the existing documentation, as a basis for understanding the nature of the obstacles encountered and of the efficient methods available. We hope that we provided a paved way to anyone willing to deepen this subject.

References

- [1] Stuart J. Russel, Peter Norvig - Artificial Intelligence: a modern approach, Prentice Hall 1995
- [2] J.R. Parker - Algorithms for Image Processing and Computer Vision, Wiley Computer Publishing 1995
- [3] Mark Simon, Sven Behnke, and Raul Rojas: Robust Real Time Color Tracking, in: Proceedings of: The Fourth International Workshop on RoboCup, pp. 62-71, Melbourne, Australia, 2000.
- [4] Performance Development of a Real Time Vision System By Joseph G. Loomis, Jason D. Palmer, Prachi Pandit - Cornell RoboCup Documentation
- [5] The BTB848 FreeBSD Kernel Driver Manual Page (man bktr)
- [6] The Matrox Meteor FreeBSD Kernel Driver Manual Page (man meteor)
- [7] <http://www.mvd-fpga.com> - The Multi Vision Design web page
- [8] The BECK IPC web site - www.beck-ipc.com