

PERFORMANCE DEVELOPMENT OF A
REAL-TIME VISION SYSTEM

By

Joseph G. Loomis, Jason D. Palmer, Prachi Pandit
(in alphabetical order)

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING (ELECTRICAL)
AT
CORNELL UNIVERSITY
153, RHODES HALL, ITHACA NEW YORK 14850
MAY 2003

© Copyright by Joseph G. Loomis, Jason D. Palmer, Prachi Pandit, 2003

ABSTRACT
MASTER OF ELECTRICAL ENGINEERING PROGRAM
CORNELL UNIVERSITY

Title: Performance Development of a Real-time Vision System

Authors: Joseph G. Loomis, Jason D. Palmer, Prachi Pandit

A real-time vision system used to extract meaningful data from a captured image is very hard to develop. Every piece of the system from the graphical user interface to the image processing code must be designed for performance. Developing such a system requires both fast running functions and careful memory allocation. Additionally, in order for this real-time system to perform efficiently, it is necessary for the system to be calibrated correctly. This means that before the system starts, certain measures must be taken to ensure that any distortion resulting from the curvature of the camera lens and from the mapping of a 3D image to a 2D image be resolved.

The goal of this work is to document the development of the Vision 2003 System. Documentation is critical since no members will be returning and the system must be understandable for future members to use. It was with great reluctance that the Vision 2002 system was used only as a template for the Vision 2003 system. However, whenever possible the Vision 2002 code was used since it has been proven to work. Vision 2003 was built with the ideas of performance and simplicity in an effort to provide a vision system for future RoboCup teams.

Dated: May 2003

Project Advisor:

Professor Raffaello D'Andrea

INDIVIDUAL CONTRIBUTION: JOSEPH G. LOOMIS

As a member of the RoboCup Software Engineering Vision team it was my main goal to improve the Vision 2002 system. This was a very daunting task since the current vision system not only worked extremely well but also contained a considerable amount of code. However, there were a number of things that could be improved. The main problem with the Vision 2002 was the latency of the information it was producing. The system also lacked documentation and explanation of its architecture and code especially its ball vision system, used to locate the ball in the occluded regions. Luckily, all of these problems were solved simultaneously when Vision 2003 was developed.

The first step in improving an existing system is to determine exactly what needs to be improved. For Vision 2002, it was clear that its latency was real a problem. Latency is the time it takes from when an image is first captured until object information from that image is sent to the AI computer. Testing revealed that Vision 2002 had a latency of 4-6 frames at 60 fps. This is a factor of more than 3 times the theoretical value of less than two frames, one to capture the image and one to process the image. Since the time to capture an image is fixed at one frame, the time to process the image was believed to be the cause.

The next step was to develop a way to locate the source of this latency. The Vision 2002 team believed the latency was a hardware problem. The computer was not able to do all of the necessary image processing between frames and that the code could not be changed for this to occur. I created a separate vision test environment that simply captured an image at 60 fps and located the ball. I was then able to use the LED test from previous years to test the latency of this system. This testing revealed that it was not a hardware problem since the LED test returned the theoretical latency of under 2 frames. Since it was possible

to grab an image and process some of it at 60 fps with a latency of less than two frames image processing was the most likely cause of the latency. Of course, locating the ball with two frames latency is not running all of vision with 2 frames latency.

Having determined that the latency problem was not hardware I was faced with a very important decision. The code for the Vision 2002 was very large and was able to run RoboCup Main Vision, a Ball Vision system and RoboFlag Vision. It also had a complex Graphical User Interface with a number of threads running simultaneously. In addition, the documentation of this code was minimal and not very comprehensive. I could spend my time fixing and documenting the Vision 2002 system or I could develop a new system. Vision 2002 did work and with some investigating it might be possible to locate the latency problem if it was in an algorithm. On the other hand, if the problem was because of its architecture or graphical user interface (GUI) it would be very hard to locate and correct without making major changes. It was this uncertainty in the cause of the latency that resulted in my decision to develop a system from the ground up using the Vision 2002 system as much as possible.

I began to add components of the Vision 2002 system to the vision test environment that I wrote to test the latency of the system. I timed each piece of code to see how long it took and also the latency of the complete test environment. With every Vision 2002 component that was added, the processing time began to creep closer to the absolute limit of 16 ms but the latency remained below 2 frames. Finally when I attempted to segment four colors the latency of the system became unstable and the time of the loop was just over 16 ms. The cause the latency was known for certain. Vision 2002 did too much processing of the image; it would have to be reduced to reduce the latency of the system.

Knowing the cause of the problem was just the beginning.

Reducing the processing time of a highly optimized vision system was not going to be easy. Luckily, the test environment did give some clues on how this could be done. Vision 2002 segmented the input color image into 6 binary buffers. One buffer for each segmented color. Each of these buffers had to be loaded and processed, requiring about 2 ms each. Processing involved locating each blob in the binary image. If there was some way to reduce the number of binary buffers from 6 to 2 the overall improvement of the system could be enough to reduce the latency to less than two frames.

The binary buffers are used to locate the center of each of the color markers used to identify the robots and the ball. The results of processing each buffer are the location of each blob and its color. The only way to locate the blob efficiently is to process a binary image. The way to determine the color of the blob can be done in multiple ways. For Vision 2002, it is simply the number of the buffer the blob is located in. Buffer 0 for example is orange while buffer 1 is yellow. One must realize that the color of the blob is also encoded in the location information from the processed buffer. The location of a blob can be used to locate the pixel in the input image. Once the pixel is located its color can be determined by using a Look-up table, just like during segmentation. The 6 binary buffers could be reduced to 2 buffers if there was a way to ensure that the color blobs do not overlap or merge together in the binary buffer. The center markers of the robots and the ball could never overlap or be close enough to be merged together. The peripheral markers used to determine robot orientation are designed to be separate from each other in the binary image even if they are the same color and next to each other. This provided a way to reduce the number of binary buffers from 6 to 2 and test the latency of the overall system.

Colors were segmented into 2 binary buffers and a new function to determine the color of a pixel based on location was added to the test

environment. The latency of this new system was below 2 frames and the vision system was able to determine the location of all the robots and the ball as well as the orientation of our robots. The latency problem was solved.

The ball vision system that was part of Vision 2002 was undocumented and tightly coupled to a system that had latency problems. A new ball vision system needed to be developed from the ground up just like main vision. The most notable differences between main vision and ball vision are that ball vision uses two cameras and only requires the ball to be located each frame. Even with these changes a lot of the main vision code could be used for ball vision without any changes. A simple vision test environment was again used to determine if ball vision could be operated at 60 fps. Grabbing from two cameras at 60 fps really hurt the performance of the system. After segmentation and ball location the system is very close to being stable. Ball vision could probably run at 60 fps if a faster computer or a segmentation algorithm that does not use a LUT were used. Ball vision does operate at 30 fps without any problems. Detailed documentation of ball vision has also been done to ensure that future teams will not have to start over with a ball vision system.

Besides reducing the latency and developing ball vision a number of other improvements were made to the vision system. There is a large reduction in the amount of code needed to perform the same tasks as Vision 2002. The number of objects and threads were each reduced from their original numbers of 8 and 7 to 3. The documentation of Vision 2002 system lacked explanation of their architecture, design decisions and code. Vision 2003 documentation on the other hand strives to explain all three in detail. Changes to the Mil Library version, robot hats, network connection and packet information were all done to improve the overall operation of the vision system.

SUMMARY OF ACCOMPLISHMENTS

1. RoboCup Main Vision is stable at 60 frames per second (fps) and has an image processing latency of less than 2 frames.
2. Main Vision can implement both RGB and HSI segmentation.
3. Automation of distortion calibration was developed.
4. Automation of AI calibration was developed.
5. A multi-camera vision system used to track the ball in occluded regions was developed and documented.
6. Research into intensity normalization and robot recognition was performed.

May you see lots of goals scored against them!

Table of Contents

Table of Contents	ix
1 Requirements and Design Specifications	1
1.1 Robocup Small Size League	1
1.2 Additional System Requirements	2
1.3 Complete Description of Vision 2002	2
2 Setup	4
2.1 Setup Overview	4
2.1.1 Calibration toolkit (CALIB)	5
2.1.2 RoboCup Main Vision (Dice)	5
2.1.3 Ball Vision System (Aux)	6
2.1.4 RoboCup Camera Setup	8
3 Main Vision Algorithm Design (Dice)	11
3.1 Image Capture	12
3.2 Segmentation	12
3.3 Blob Analysis	13
3.4 Ball Identification	13
3.5 Opponent Identification	14
3.6 Robot Identification	14
3.6.1 Robot Markers	14
3.6.2 Robot Marker Ordering	15
3.6.3 Robot Location and Orientation Angle	16
4 Ball Vision Algorithm Design (Aux)	17
4.1 Image Capture	17
4.2 Segmentation	18
4.3 Blob Analysis	18

4.4	Miscellaneous	18
5	Dice Code Specifics	19
5.1	Parameters	19
5.2	Objects and Classes	20
5.3	Graphical User Interface (GUI)	21
5.4	Mil Object Class	22
5.4.1	Mil Object Segmentation	23
5.4.2	Mil Object Blob Analysis	24
5.4.3	Mil Object Identification	25
5.4.4	Merging and Partitioning Blobs	25
5.4.5	Locating the Ball and Opponent Robots	26
5.4.6	Locating Cornell's Robots	26
5.4.7	Vision Packet Dispatch	27
5.5	Robot Class	27
5.5.1	Least Squares Method	28
5.6	Blob Class	28
5.7	The Vision Loop	28
5.8	Debugging Console Window	28
6	Aux Code Specifics	30
6.1	Parameters	30
6.2	Objects and Classes	30
6.3	Graphical User Interface	31
6.4	Mil Object Class	31
6.5	Mil Object Segmentation	31
6.5.1	Mil Object Blob Analysis	32
6.5.2	Mil Object Identification	32
6.5.3	Vision Packet Dispatch	32
6.5.4	Debugging Console Window	32
7	Calibration toolkit (CALIB)	33
7.1	Distortion Calibration	34
7.1.1	Distortion Calibration: Vision 2002	34
7.1.2	Distortion Calibration: Vision 2003	35
7.2	AI Field Calibration	39
7.2.1	Other Considerations - Edge Detection	46

8	Intensity Normalization and Masking	49
8.1	Motivation	49
8.2	High Level Description	49
8.3	Algorithm	51
8.4	Edge Detection	52
8.5	Mask Creation	53
8.6	Algorithm Procedure	55
8.7	Results	57
8.8	Conclusions	58
9	Conclusion	61
9.1	Achieved Requirements	61
9.2	Unresolved Bugs	62
9.2.1	Dice System Bugs	62
9.2.2	Aux System Bugs	62
A	Possible Problems	63
B	Possible Improvements	64
C	General Advice	65
D	Dice and Aux User Manual	66
E	CALIB User Manual	68
E.1	Distortion Calibration	68
E.2	AI Calibration	73
	Bibliography	75

Chapter 1

Requirements and Design Specifications

1.1 Robocup Small Size League

RoboCup is a game where small sized robots compete against other teams from around the world in a game of soccer. The ball in this case is a golf ball and the field is about the size of a ping pong table. The preferred method to identify robots and to track the ball is through the use of a global vision system.

Specifications of the global vision system are described below.

1. Field size is 2900 mm x 2400 mm. There is a 50 mm wall that surrounds the field so vision only needs to work for a 2700mm x 2300 mm field.
2. The size of the goal box is 700 mm x 180 mm.
3. The ball color is orange is about 43 mm in diameter.
4. Recognizes up to 10 robots, 5 from each team.
5. Each team will have a team color (Blue or Yellow) and will be required to place a circle of 40 mm diameter on top of the robot.

6. Robots may use cyan, magenta and green as defined by the competition organizers. The colors black and white may be used without restriction.
7. Cameras will be mounted 3000 mm above the playing field and may not protrude more than 150 mm.
8. Referees may use a grey hockey stick and vision must be able to ignore it.
9. The lighting conditions may vary from 700 LUX to 1000 LUX in the entire field.

1.2 Additional System Requirements

In addition to the above vision system requirements Vision 2003 needs to improve and correct the problems with Vision 2002. Latency, the main problem with Vision 2002, is the time from when the picture begins to be captured until that information is sent to the AI computer. The theoretical latency of a vision system is under 2 frames. One frame is used to grab the image and another fraction of a frame to process it. The actual latency for the Vision 2002 system at 60 Hz on Antares (Dual Xeon 2 GHz processors) was 4 to 6 frames. It was the primary goal of this project to reduce the latency to the theoretical latency of less than 2 frames.

1.3 Complete Description of Vision 2002

Since a working version of Vision 2002 was given to us in the fall it was possible to extract both good and bad stuff from it.

Features:

1. Operates at both 30 and 60 Hz.

2. Works for both RoboCup and RoboFlag
3. Custom coded segmentation using Look-up Table (LUT).
4. Uses a LUT for color segmentation.
5. Includes multiple camera system for ball location.
6. Calibration units are separate with field calibration in AI.

Flaws:

1. Latency of 4 to 6 frames.
2. Documentation is lacking.
3. Operation at 60 Hz is unstable.
4. System crashes for unknown reason.
5. Not a development system because of the complex code and object system.
6. Poor use of threads.
7. Look-up table (LUT) is hard to create.

Chapter 2

Setup

The main vision system (Dice) requires a camera, a frame grabber, computer and the necessary cables to connect each component for operation. Dice is explained in detail in chapters 3 and 5. For ball vision (Aux) requires one computer separate from main vision with two frame grabbers and two cameras. The Aux system will be explained in detail later in chapters 4 and 6.

2.1 Setup Overview

The vision system needs to work in a field the size of 2900 mm by 2700 mm. This means that the main camera needs to have the correct lens to cover the whole field. The requirements for ball vision are slightly less, each of the ball vision cameras can cover a field half to a quarter of a field since the region of interest is only a quarter of the field. Frame grabbers for each camera should be able operate at 60 fps.

2.1.1 Calibration toolkit (CALIB)

CALIB is implemented solely in MATLAB. It initially uses the Sony DCX-9000 camera and the MIL image capture tools to obtain the field images. These are then used in MATLAB to carry out the following calibrations.

- Distortion Calibration
- Field Calibration

The details of the algorithms are explained in chapter 7.

2.1.2 RoboCup Main Vision (Dice)

For RoboCup main vision we use the same Sony DCX-9000 camera as Vision 2002 but a faster computer Antares running Mil 7.0 Library instead of the Mil 6.1 Library. This setup requires the usage of the following equipment.

Hardware for Dice:

1. 1 x PC workstation (Antares)
2. 1 x long cable to connect camera to the frame grabber.
3. 1 x frame grabber (Matrox Meteor Multi-Channel)
4. 1 x power cable for the camera
5. 1 x high resolution camera (Sony DCX-9000)
6. 1 x camera mounting gear

This setup is able to meet all the design goals of the new vision system. This system is capable of segmenting 6 colors out of a 640 x 480 field size at 60 fps.

Main Computer Setup

Dice runs on Antares, a Dual Xeon 2GHz machine with 1 Gb of RAM memory. The Mil Library 7.0 must be installed as well as the hardware dongle for the software to work. A Matrox Meteor II Multi-Channel card and a MAP 950 network card must be installed into PCI slots on Antares.

Camera and Lens

Sony DCX-9000 camera with a zoom lens is used to capture the whole field. It is setup with the front of the camera facing the door in the RoboCup lab.

Frame Grabber

Matrox Meteor II Multi-Channel frame grabber is used with the Mil 7.0 Library. The Mil Library provides the drivers to work with the Matrox Meteor II MC frame grabber.

Network Card

A MAP 950 network card with a RS-232 cable is used to connect the Antares to the AI computer.

2.1.3 Ball Vision System (Aux)

For RoboCup ball vision we use the two RoboFlag cameras fitted with the smaller 6mm lens with Hercules (Dual P-IV Xeon 1700) computer. This setup requires the usage of the following equipment.

Hardware for Aux:

1. 1 x PC workstation (Hercules)
2. 2 x long cable to connect each camera to a frame grabber
3. 2 x frame grabber (Matrox Meteor II Multi-Channel)
4. 2 x power cable for the camera
5. 2 x high resolution camera (Pulnix TCM-6700)
6. 2 x camera mounting gear

This setup is able to meet all the design goals of the new vision system. This system is capable of segmenting 1 color out of two 480 x 320 field size at 60Hz.

Computer Setup

Aux runs on Hercules, a Dual Xeon 1700 machine with 480 Mb of RAM memories. The Mil Library 6.0 must be installed as well as the hardware dongle for the software to work.

Two Matrox Meteor II Multi-Channel cards and a MAP 950 network card must be installed into PCI slots on Hercules.

Camera and Lens

Two Pulnix TCM-6700 cameras with 6mm lens are used. Each camera is placed above a goal with their fronts facing the goal.

Frame Grabber

Two Matrox Meteor II MC frame grabbers are used with the Mil 6.0 Library. The Mil Library provides the drivers to work with the Matrox Meteor II MC frame grabber.

2.1.4 RoboCup Camera Setup

The setup of the camera system is a multi-step process that involves the setup and testing of multiple cameras to get the correct field views. Please refer to the trouble shooting section of this paper for help with possible problems.

Required equipment for Dice camera setup:

1. 1 x camera (Sony DCX-9000)
2. 1 x power supply cord
3. 1 x camera cord
4. 1 x magic arm mount
5. 1 x lens
6. 1 x frame grabber (Matrox Meteor II MC)
7. 1 x computer with Matrox Intellicam (Installed with the Mil Library)

Required equipment for Aux camera setup:

1. 2 x camera (Pulnix TMC-6700)
2. 2 x power supply cord

3. 2 x camera cord
4. 2 x magic arm mount
5. 2 x lens
6. 2 x frame grabber (Matrox Meteor II MC)
7. 1 x computer with Matrox Intellicam (Installed with the Mil Library)

Before a camera is setup the cables and the correct Matrox meteor card should be tested. The Matrox computer program Intellicam works the best for this.

Steps for camera setup:

1. Attach the correct lens to the camera while the camera is off.
2. Attach the one end of the camera cable to the Matrox Meteor II Multi-Channel card being careful that the pins line up correctly and the other end to the camera.
3. Connect power to the camera and then plug it into the wall outlet.
4. Open up Matrox Intellicam program and do File → Open → Sony_Camera.dcf
5. An image should appear when the camera button is depressed on the main setup.
6. The Camera should be unplugged and disconnected so that the cables can be routed to the correct field location using the ladder.
7. The ladder should be used to install the camera arm in the correct location.

8. Attach the camera mount to the camera and install the camera into the magic arm.
9. Run the Matrox Intellicam software again and use it to focus the camera and adjust the brightness. Both of these adjustments are done using the lens.
10. Adjust the camera into the correct position using the camera view.
11. Readjust the camera focus and brightness.

The steps are the same for the ball vision cameras. The Pulnix_Camera.dcf should be used instead of the Sony_Camera.dcf connected to the same computer. To switch between cameras all windows must be closed in Matrox Intellicam and the Device must be toggled from Dev_0 to Dev_1.

Chapter 3

Main Vision Algorithm Design (Dice)

At first the vision system appears very complex but it is based on a simple system that sequentially reduces the input image into locations and orientation of objects. Dice is composed of four main parts. The first part captures/grabs an image of the playing field at 60 fps. This color image is then segmented using a LUT into two binary images. The first image contains the binary representation of the colors dark blue, yellow and orange. The second image contains the binary representation of the colors cyan, magenta, and bright green. The locations of the binary blobs are then found using blob analysis on each of the images. Once a blob is located its color is determined and it is then used to identify the ball, opponent robots and Cornell's robots. Identification of the ball and opponent robots consists of just a position in the x/y plane. Identification of our robots consists of a position and an orientation angle. Dice is described in detail below while its code implementation is described in Chapter 5.

3.1 Image Capture

Image capture is the process where a color image is grabbed from the camera and placed into a memory location (buffer) on the host computer. This image must be transformed into a packed RGB image before it can be processed. The Mil Library provides a function to grab and copy the image into the correct format. Double buffering is used with an asynchronous grab to improve performance. Double buffering is where two buffers are allocated to the grab process. This allows an image to be grabbed into one buffer while the image in the second buffer is being processed.

3.2 Segmentation

Segmentation is done using a 16 MB LUT to map a packed RGB image buffer into two packed binary image buffers. Two buffers are used for a number of reasons. The processing time is reduced by running two threads simultaneously to process the two buffers. The main thread processes the binary representation of dark blue, yellow and orange. The second thread processes the binary representation of cyan, magenta and bright green. The two buffers provide the maximum distance between blobs of different colors in the x/y plane. This ensures that two different color blobs are not merged together during blob analysis. In addition, the two binary buffers take up a third of the space in memory as six buffers, one for each color. Once the images are processed using blob analysis their color is determined by reading the color of a pixel location in the center of the blob. The binary images that are created using segmentation are in packed binary mode. This is where every pixel is represented as a bit in the final image. This allows 8 color pixels to be compressed into one byte in

the packed binary image. It should be noted that the order of the 8 color pixels is reversed.

3.3 Blob Analysis

Blob analysis is a multi-step process that determines the location and orientation of objects in the input image. A blob is just a region of the input image that has a single color from the LUT. During blob analysis, a blob is just a region that has a value of 1 (or true). This is why the colors during segmentation are split. There is no way for the different color blobs to overlap in the binary image and therefore be considered the same blob even though they have different colors. There is no way for blob analysis to take blob color into account since it only operates on binary images. The first step in the blob analysis process is erosion. Erosion gets rid of most of the image noise and reduces the blobs to those used to identify the robots and the ball. Each of the two binary images is eroded and then the location, size and color of each blob is determined and placed in a vector. Each thread generates a vector of blobs that is merged into one vector after blob analysis. The distance of each blob to the other blobs is checked to ensure that if two blobs are close enough and of the same color they are merged into one blob. The Mil Library erosion and blob analysis functions are used to process the packed binary images.

3.4 Ball Identification

The ball is easily identified as the largest blob with the color orange. If no ball is located it is considered occluded or missing and no location is sent to the AI computer. The Aux system should be able to locate the ball if the main camera cannot. The

Aux system is explained in chapters 4 and 6.

3.5 Opponent Identification

The opponent's robot is located just as easily as a ball since its location is determined by a team color marker located in the center of the robot. If no team color marker is found the robot is not located. Team colors are currently dark blue or yellow and are required to be 40 mm in diameter placed in the center of the robot.

3.6 Robot Identification

Identifying Cornell's robots is quite complicated. Dice uses a method of least squares to determine both the location as well as orientation. This is the same method that Vision 2002 used and that was proposed by Professor Raffaello D'Andrea. The steps for robot identification are explained in greater detail below while its code implementation is explained in Chapter 5.

3.6.1 Robot Markers

Determining the general location of our robots is done by locating the center marker. The color of Cornell's robots is known a priori and thus we are able to get a general location of our robot. The location of the center marker is referred to as the general location of the robot because we also know the configuration of four orientation markers on our robots. This means that the center marker as well as the four orientation markers can be used to determine the center of the robots more precisely than just the one center marker. Once the center blob is located the marker blobs around it are found by finding the distance from the center blob to them. This forms a combination

of markers that can be used to determine the robot identification. A hash table that maps the number of cyan, magenta and bright green markers to robot identification is used to determine the real world number of a robot. The hash table is generated from the input file which calculates the possible color combinations. Combinations of all four markers and when any one of the four markers is missing are generated. Finding the robot identification is essential because it is the only way to track a robot for AI and to determine a robot's orientation. Robot identification is a number from zero to four of the robot based solely on a robots marker pattern. It should be noted that a robots number is just the number that the robot is in the vector of robots and has no real world significance. A robot's identification however does represent the number of a robot in the real world. Therefore, a robot could have a robot number of 3 and have a robot identification of 0. This robot would be robot 0 in the real world and have marker pattern of consisting of one magenta and three cyan markers.

3.6.2 Robot Marker Ordering

A robot's markers must be placed into the correct order to determine the orientation of a robot. The angle a robots maker makes with the x-axis is first determined and then they are ordered based on the size of the angle. The markers are then permuted to the correct order of markers using the previously found robot identification and a parameter file provided marker order.

3.6.3 Robot Location and Orientation Angle

Once the location of the center of each robot marker and the angle of the marker relative to the playing field x-axis it is possible to determine its location and orientation. Both the orientation and location are found using the least squares method. The angle is then normalized so that it is between plus and minus pi.

Chapter 4

Ball Vision Algorithm Design (Aux)

Ball vision is used to locate the ball when it has been occluded from the view of the main camera. This can occur when the ball is in a corner of the field or a number of robots encircle the ball away from the center of the field. By using two cameras and an additional computer we are able to locate the ball much more effectively in the occluded regions. Two cameras require twice the information to be captured and processed in the same 60 fps time. Two threads are used like in Dice except each thread processes an image from one of the two cameras. The results from each thread are combined and the largest blob over a given size is sent to the AI computer using a MAP network card.

4.1 Image Capture

In order for image capture to work at 60 fps for two frame grabbers on the same computer the images must be half the size or at a resolution of 320 x 240. This does not reduce the overall location accuracy of the ball since each camera covers half the

field. Mil Library functions are used to grab the images and to copy them to packed RGB format. Double buffering is also used since the performance gain is significant.

4.2 Segmentation

A LUT table is used as in Dice. Each image is reduced to a binary image of orange and not-orange. Only half of the field is captured in each image and only half of each image is processed. This helps improve performance so that Aux is able to run at 60 fps.

4.3 Blob Analysis

Blob analysis with erosion takes too long to run at 60 fps, so no erosion is done on the captured images. All blobs are located even noise ones and then the largest blob is located. If the largest blob is not of a minimum size it is considered noise and not used. This could result in an actual ball location being lost since it is still possible to partially block the ball in the corner of the field with a robot.

4.4 Miscellaneous

The two camera system really hurts performance and does not allow much post processing once an image is grabbed and copied. Aux is still being developed and might require operation at 30 fps instead of 60 fps if a faster computer does not improve its performance.

Chapter 5

Dice Code Specifics

In order to get the real-time vision system to operate at 60 fps a number of changes had to be implemented to the process flow. This chapter provides a detailed explanation of the code and how it was implemented. The simple class structure is maintained for both performance and development.

5.1 Parameters

Dice is built with the assumption that a parameter file will be included at start up. This parameter file contains all the information that is necessary for vision to run. Variables such as the number of robots and robot encodings are included in this parameter file. The parameter file also contains links to three important files. The first file is the camera dcf file which is required for the Matrox Meteor II MC frame grabber to connect to the camera and capture images from it. The second file is a look-up table which maps 3 byte RGB pixel values to an integer color value ranging from 0 to 5 depending on the color. Refer to the table below for color mapping information. All other colors are ignored.

Color	Integer Value
Orange	0
Yellow	1
Blue	2
Green	3
Cyan	4
Magenta	5
solid	3

Table 5.1: LUT mapping of real word colors to an integer value.

5.2 Objects and Classes

All the vision code resides in Mil Object, Robot and Blob classes. The GUI interface is a way to run the functions of the Mil Object class. The GUI is the only file that has threads which are used to create a start and stop buttons as well as improve the performance of blob analysis. The class structure is shown in figure 5.1.

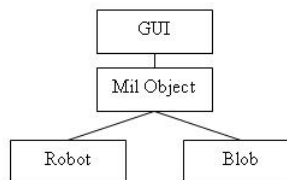


Figure 5.1: Class structure for Dice.

The last file is a distortion calibration file which maps the captured image field values to a real world location. The captured image is distorted because of the camera lens and therefore the 3D curved image that the camera captures must be flattened

into the 2D real world image. The distortion calibration file provides a grid of points from the 3D curved image and their real world equivalent points. If a segmented point does not fall on a grid point the Mil Library software interpolates the point location based on the distortion file.

5.3 Graphical User Interface (GUI)

The GUI is minimal because of the processing constraints of the system. Testing revealed that outputting to a console window or the GUI while the vision loop is running would cause the system to be unstable and result in abnormal behavior.

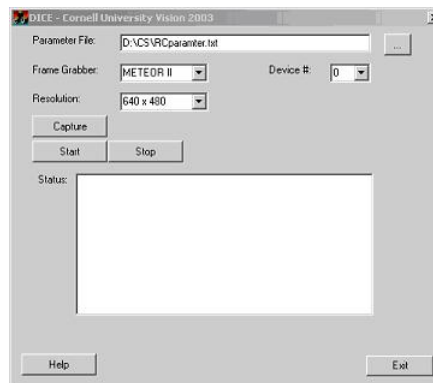


Figure 5.2: Picture of Dice GUI.

The GUI was specifically designed to be separate from the image processing code. This means that the GUI can be replaced or modified without rewriting any of the image processing code. The GUI is also the only file to have threads. Threads are used to provide a start and stop button and to increase the performance of blob analysis. When the start button is pressed the GUI starts a thread which runs the vision code in a continuous loop. Each time it loops it checks a flag which is set by

the stop button to see if it should continue.

It is strongly advised that the GUI be kept simple since it is believed to be one of the primary causes of the latency problems in Vision 2002. It is also advised that the number of threads be kept at a minimum since it is believed to be the cause of the Vision 2002 crashes.

5.4 Mil Object Class

This class is where all the image processing is done. It primarily interacts with the Mil Library functions and is used to create the buffers that store images. It includes the important functions of segmentation, blob analysis, identification and vision packet dispatch. It reduces an image of the field into the location and sometimes orientation of objects on the field.

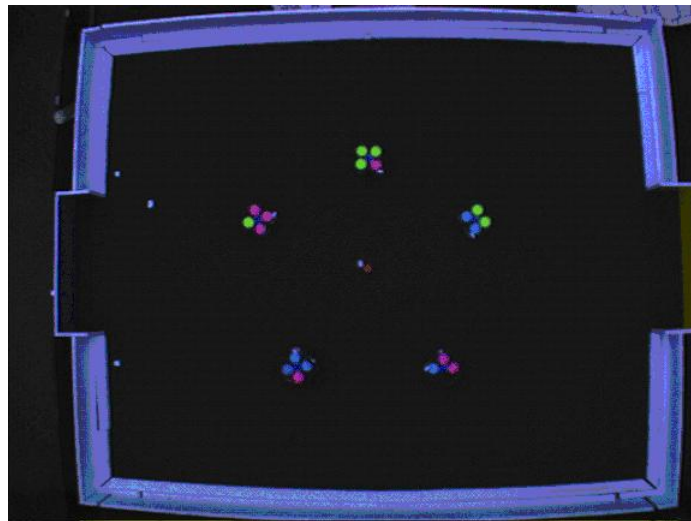


Figure 5.3: Image before Segmentation.

5.4.1 Mil Object Segmentation

The segmentation code that is currently used is very similar to Vision 2002 except 6 colors are segmented into 2 binary buffers instead of 6 binary buffers. This reduces the number of buffers that need to be processed in blob analysis. The first binary buffer contains the segmented colors of dark blue, yellow and orange. The second binary buffer contains the segmented colors of bright green, cyan and magenta. The color of a blob is determined after it is located by checking the pixel value in the original image. The segmentation file takes a packed RGB image buffer and the RGB look-up table and outputs two packed binary images. The segmentation code consists of two 'for' loops. The first 'for' loop increments the row or y value while the second 'for' loop increments the column or x value. This is done so that the part of the image that contains the field is processed. The minimum and maximum values of the 'for' loops represent a square that encloses the region to be processed.

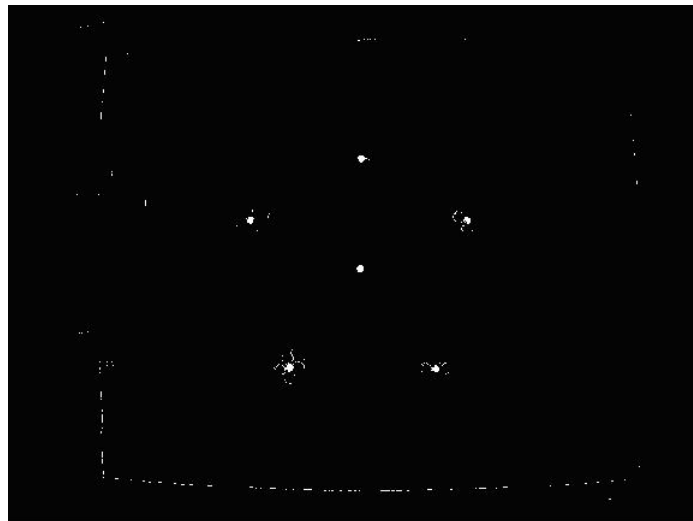


Figure 5.4: Segmented binary image of yellow, dark blue and orange.

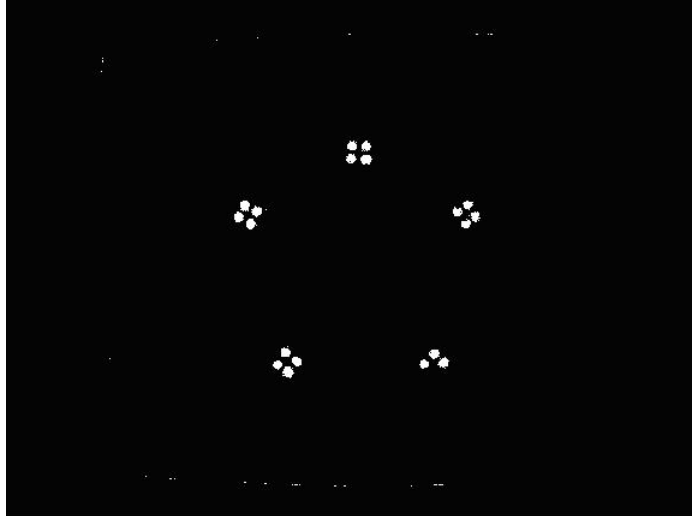


Figure 5.5: Segmented binary image of bright green, magenta and cyan.

Segmentation code that uses a YUV (HSI) look-up table has been investigated with promising results. YUV encoding represents colors as a hue, saturation and intensity value. Since one of the variables is intensity it can be ignored and therefore the brightness of the field has less of an effect on the segmentation of colored markers.

Currently, the white field boundary and dark blue appear to have the same hue and saturation but this is a very preliminary result. The LUT used to do this segmentation was generated by hand and requires a range of hue and saturation values instead of the traditional one to one mapping of a LUT file.

5.4.2 Mil Object Blob Analysis

Blob analysis is a multi-threaded process where each of the two binary images is processed by a thread. The Mil Library functions which are optimized to run on packed binary buffers are used to erode and to locate the center and size of each blob

in the images. Once the center of the blob is located the pixel value at that location is determined and used as the blob color. Two vectors of blobs are generated using blob analysis. The vectors of blobs are merged into one vector before identification is done.

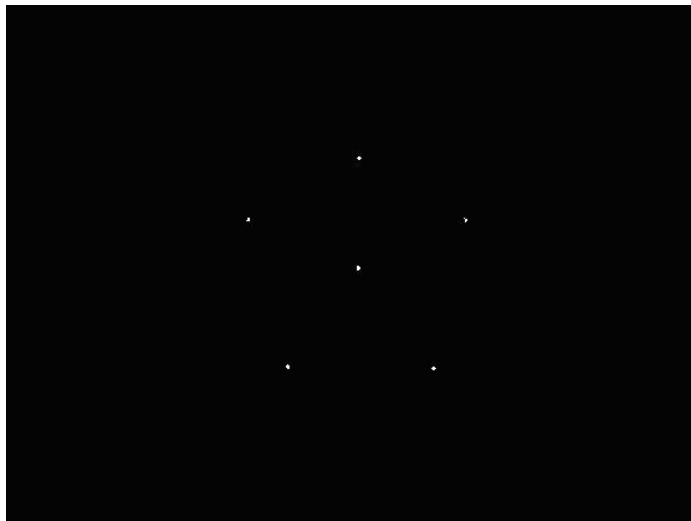


Figure 5.6: Eroded binary image of yellow, dark blue and orange.

5.4.3 Mil Object Identification

Identification takes a vector of blobs and generates the locations of the ball and the opponent robots. It also determines the location and orientation of our robots.

5.4.4 Merging and Partitioning Blobs

Blobs that are the same color and are close to each other are first merged into one blob. This helps reduce some of the burden of having a perfect LUT. Blobs are then partitioned by color into 6 vectors of blobs, one for each color.

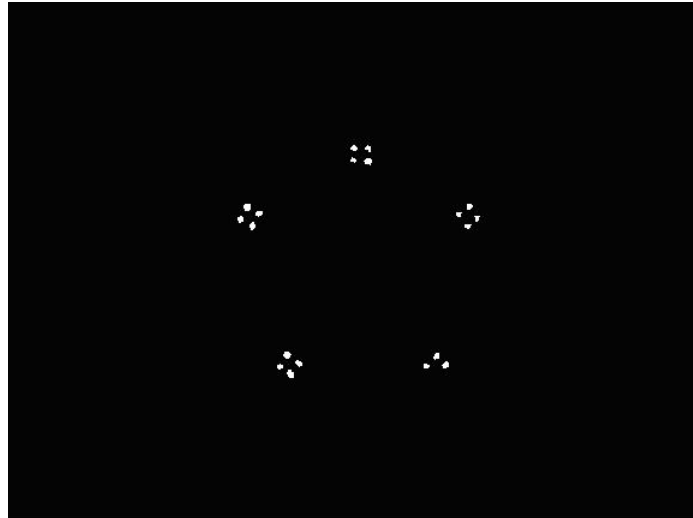


Figure 5.7: Eroded binary image of bright green, magenta and cyan.

5.4.5 Locating the Ball and Opponent Robots

Once the colors have been partitioned into color vectors of blobs locating the ball and the opponent robots is very easy. The ball is the largest blob of color orange. The opponent robots are the 5, or the opponent team size, largest blobs of the opponent team color which is currently yellow or dark blue.

5.4.6 Locating Cornell's Robots

The Robot class explained in section 5.5 goes into detail on how Cornell's Robots are identified. A robots center marker is found by using the 5 largest blobs of the team color. A robot is added to a vector of robots for each team color blob found. The blob vector is then looped through for each robot and a marker is added to the robot object for each blob that is bright green, cyan or magenta within certain distance from the center blob. The location of the center blob and three or four of the orientation

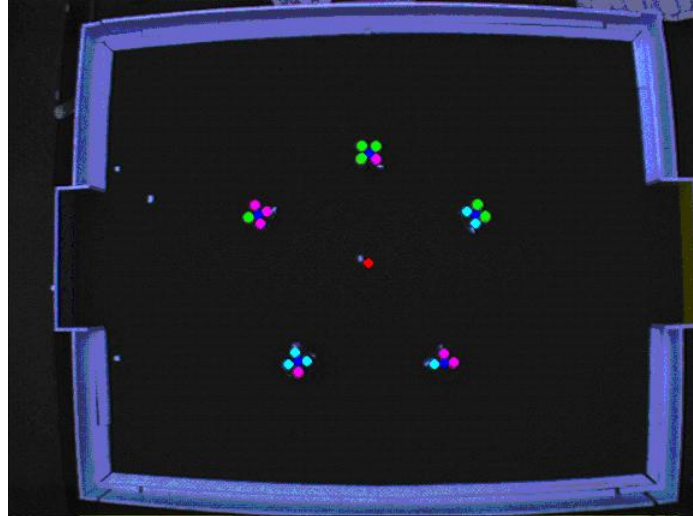


Figure 5.8: Input image with the located blobs identified.

markers is all that is needed to correctly find a robot's identification, orientation and location. Distortion correction is done once a robot's location is determined.

5.4.7 Vision Packet Dispatch

The information from Dice must be sent to the AI computer so that it can be used. The Mil Object has a function that creates a vision packet and another one that sends it. The location of the ball and opponent robots as well as our robot location and orientation are sent in the vision packet.

5.5 Robot Class

The robot class is a class used to represent Cornell's robots. A robot has a center marker with up to 4 orientation color markers. All markers have a location and color associated with them. It also finds the angles that a color orientation marker forms

with the x-axis of the field. It then uses the robot encodings provided in the parameter file to determine the robot identity or real world robot number. The least squares method is then used to determine the orientation and location of a robot based on the location of the center marker and the three or four orientation markers.

5.5.1 Least Squares Method

This is the same method used by the Vision 2002 team and recommend by Professor Raffealo D'Andrea.

5.6 Blob Class

The blob class is used for the blobs that are generated from blob analysis. A blob has six pieces of information associated with it. It has an image location and a real world location as well as size and color information. This class was slightly modified from the Vision 2002 blob class.

5.7 The Vision Loop

The Vision Loop is the piece of code that runs when the start button is pressed. It resides in the GUI class function Process Thread. The Vision Loop implements double buffering as well as multi-threaded processing.

5.8 Debugging Console Window

A console window has been included in the code to allow for easy debugging. All 'cout' statements are routed to the console window from any class in the system. It

should be noted that Dice cannot output to the console and run at 60 fps. Vision can however place information into a buffer which can be printed to a GUI 'listbox' or console once the vision loop has been stopped at 60 fps.

Chapter 6

Aux Code Specifics

The Aux system was developed using Dice as a template. The major difference between the systems is that two cameras are used for the Aux system while only one is used by Dice.

6.1 Parameters

The Aux system requires a parameter file at start up. This parameter file contains all the information that is necessary for the Aux system to run. The parameter file also contains links to six files. The first two files are the camera dcf files which are required for the Matrox Meteor II MC frame grabbers to work with the Pulnix cameras. The third and fourth files are look up tables which maps 3 byte RGB pixel values to orange and not orange. The last two are for distortion calibration.

6.2 Objects and Classes

All the vision code for the Aux system resides in Mil Object and Blob and are controlled by the GUI. The GUI is the only file that has threads which are used to create

a start and stop buttons as well as improve the performance of the whole Aux system by running each camera in its own thread. The class structure for the Aux system is shown below.



Figure 6.1: Class structure for Aux.

6.3 Graphical User Interface

The GUI runs the same as in Dice.

6.4 Mil Object Class

This class is basically the same except segmentation and identification have both been modified. Segmentation only needs to remove one color while identification only needs to locate one ball.

6.5 Mil Object Segmentation

The segmentation code is only designed to segment out the ball color. Instead of operating on the whole image it has been reduced to a quarter of the field or half of the captured image. This helped to reduce the processing time.

6.5.1 Mil Object Blob Analysis

Blob analysis does not use an erosion operator because it would take too long to run. The noise which is reduced by erosion is very small since only one color is being segmented. A vector of blobs is produced by each camera.

6.5.2 Mil Object Identification

Identification merges the two vectors from blob analysis into one vector and then selects the largest blob. If this blob is larger than a pre-determined size it is selected as the ball. We realize that a ball could be identified and not used if it is considered noise. Testing is being done to see if running at 30 fps results in better ball identification.

6.5.3 Vision Packet Dispatch

Aux is connected to the AI computer since Dice is unable to process the information while operating at 60 fps. The AI computer can easily decide which ball to use if both Dice and Aux locate the ball.

6.5.4 Debugging Console Window

Aux contains the same debug capabilities as Dice.

Chapter 7

Calibration toolkit (CALIB)

One of the most important aspects of the vision system is the calibration toolkit. Whenever a picture is taken by a camera, several events occur. One of these events is the mapping from the 3-dimensional real world space to the 2-dimensional image space. The second of these is the deformation of the real world image, due to the curvature of the camera lens, that results in a distorted image. Each of these divergences must be corrected. This is the goal of the calibration toolkit. The former of the aforementioned problems is corrected via the AI calibration, and the former is fixed via the distortion corrected. Both of these procedures are essential to the functioning of the entire vision system and are described in further detail below.

In Vision 2002 the various calibration tools are located in different places. The distortion calibration is in Matlab, the field calibration is implemented in AI in VC++, and LUT calibration is implemented in Vision but written in both VC++ and JAVA. Vision 2003 has a separate calibration toolkit which aggregates all these tools implemented on the common Matlab platform. At the moment it consists of only the distortion calibration and the field calibration, but eventually it could include the LUT calibration as well.

The only interface between CALIB and the main vision loop are the parameter files generated by the calibrations. This makes a clean cut between the two parts of Vision 2003 thus enabling each to be developed separately without necessitating modifications in the other.

7.1 Distortion Calibration

One of the essential tasks that the Robocup vision system must perform is the conversion of pixel coordinates into real world coordinates. This is because the AI module is not concerned with pixel values and runs entirely in real world coordinates to make better location and distance approximations. This translation from pixel to real world coordinates is what is known as distortion calibration.

When a picture is taken by a camera, a physical point in 3-dimensional space is transformed into a pixel in the 2-dimensional image space. Due to the curvature of the camera lens, these transformations tend to get warped so that there is no linear transformation from the image world back into the real world. This inverse transformation is not an easy task, however for our purposes a very simplified distortion calibration proves sufficient. For instructions on how to perform the distortion calibration, please see Appendix E, User Manual.

7.1.1 Distortion Calibration: Vision 2002

The distortion calibration tool for Vision 2002 was based on code available at the website http://www.vision.caltech.edu/bougetj/calib_doc. This code is based on homography calculations that are used to locate the corners of a black and white checkerboard pattern. As such, the Vision 2002 distortion calibration employed a black and

white checkerboard matt that is laid out on the entire field, including the goal boxes.

A picture of the field with this matt covering it is then taken and used for the duration of the calibration. Subsequently, the checkered field is broken up into four sections (our side, their side, our goal box, and their goal box) by user defined mouse clicks. Each of these four sections is fed into the homography code above, resulting in an output of an approximate position of all of the checker corners within the section.

The user must then fine tune the location of any of these corners (in all, totaling 728 points) that did not get calculated accurately by the homography. While the initial calculation of the corners is not too time intensive, the tuning of all of these points is, in addition to being quite tedious. It can take up to forty-five minutes for one person to perform a distortion calibration, depending on the number of points that need to be corrected. However, especially in competition, it is imperative that procedures such as distortion calibration be performed with the utmost speed while still maintaining a high degree of accuracy.

7.1.2 Distortion Calibration: Vision 2003

The 2003 distortion calibration tool draws heavily on the 2002 design. The principle of homography is still used to determine the exact location of the checker corners, however the sections that the field is broken up into is different. An earlier version of the Vision 2003 distortion calibration utilized colored dots to mark the field section boundaries. The purpose of these dots was to make the calibration process as automated as possible. This meant that the user did not have to specify the location of the section boundaries.

However, this version of the calibration did not prove sufficient for the following

reason. The dots were located within the image via a simple color segmentation process. It is so simple, in fact, that it can be written in the following one line of code in matlab:

```
[x,y]=find(((I(:,:,1)>=100)&I(:,:,1)<=230)&I(:,:,2)<=70)&I(:,:,3)<=70))
```

Here, I is the RGB image, the first index into I is the row, the second index is the column, and the third index is the respective color component (1 for red, 2 for green, 3 for blue). As can be seen from the above, the color intensity values being searched for are hard coded in. This is due to the fact that a specific color dot was chosen for this purpose. Fluorescent orange was chosen because its RGB values were far from the RGB values of white and black. Any color that is to be used for these dots must have an RGB composition that cannot be mistaken for black or white. If this happens, the segmentation may not work properly. Examples of this occurred in lighting conditions that were not as uniform as our lab is. Therefore, the segment boundary definition process had to be changed. The distortion calibration has been changed so that there are multiple ways of performing the calibration. The first method is by using the aforementioned colored dots. The second way is set up so that the user clicks on the boundaries of the segments instead of having the computer automatically detect them.

At this point, if the dots are used, the distance from every pixel point stored in x,y to every initial point is calculated where the location of the dots are specified to be these initial points. If the distance to a certain initial point is within some predetermined threshold, this point is added to a group of points specified to be closest to this initial point. In such a manner, every pixel in x,y gets classified to some initial point, unless its distance to every initial point is greater than the threshold. If

this is true, the point is deemed noise and simply thrown away. Once every point has been classified, the average x and y values of these points are taken to be the center of the dot and the homography process begins. However, if the dots are not used, the positions clicked on by the user are directly taken as the input to homography.

The checkered matt is 28 x 23 squares with two 2 x 8 pieces used to fill in the goal box. As previously mentioned, Vision 2002 divides the field into four different sections. These sections are our side of the field (14 x 23 squares), their side of the field (13 x 23 squares), our goal (1 x 8 squares), and their goal (1 x 8). Due to the large size of the first two of these four sections, the homography calculations were not able to pinpoint the location of the checker corners. This is what resulted in having the user fine tune any necessary points.

The 2003 distortion calibration tool divides the field into six sections. They are:

1. Our left side: 13 x 7 squares
2. Our middle: 13 x 5 squares
3. Our right side: 13 x 7 squares
4. Their right side: 12 x 7 squares
5. Their middle: 12 x 5 squares
6. Their left side: 12 x 7 squares

Note that these sections do not encompass the entire matt. Specifically, the goal boxes and the outer edge of the matt are not included when calculating the distortion calibration. The reasons for this are twofold. First, creating smaller sections allows the homography to run much faster and more accurately than with larger sections

of the field. And second, because of the small distance from the edge of the matt to where the circles are placed (approximately 10 cm), linear interpolation of these side points is accurate enough for our purposes. The above six sections are depicted on the following figure, in counterclockwise order:

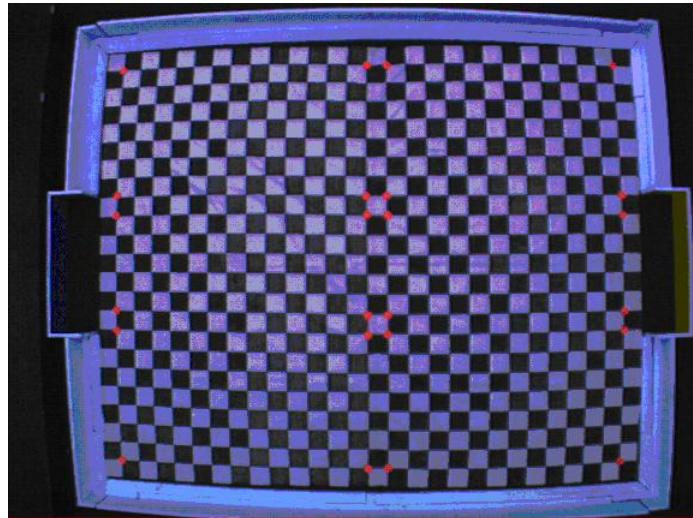


Figure 7.1: Sections used for distortion calibration.

Once the homography is completed for all of the above six sections, the data is corrected for parallax distortion. Parallax distortion results from the fact that we are approximating the 3D world in 2D. All robots and the ball have a definite height component to them, however this information is lost when the camera takes a picture. For the specifics on how parallax distortion is corrected, see the Vision 2002 documentation section 3.5.4.

At this point, the data is put into a text file with the image values and corresponding real world coordinates. It is necessary to have some fixed points in the real world map so that there is some correlation to the pixel positions in the image. As

such, each of the real world coordinates of every corner of the above six sections is stored and used as a reference point to get the real world coordinates of all other points within the grid.

This file that is created as a result of the distortion calibration now serves as a template to convert any pixel coordinate into any real world coordinate. This is currently accomplished by the MIL library via a bilinear interpolation method. Documentation of this method can be found in the MIL library manuals and in Vision 2002 documentation section 3.5.3.

7.2 AI Field Calibration

Field calibration is extremely important for Robocup as it maps critical field locations to their corresponding image locations. This information is crucial for AI as it is used to calculate field boundaries, goal locations, wall positions, etc. in the real coordinate system which is then used by AI for sending commands to the robots.

Up until now, field calibration had been implemented in AI. It was a cumbersome procedure wherein a ball had to be manually placed at various field locations and an image captured for each of those locations. The real coordinates were then calculated by using the distortion calibration parameter file and MIL functions.

Vision 2003 Calibration Toolkit makes this procedure more automated and it is completely implemented in MATLAB. The User's Guide section of the report gives a step by step procedure for carrying out the Field calibration.

This tool takes as its input the image coordinates of critical field points, obtained by mouse clicks over a field picture. The output is a parameter file which has the corresponding real world points which is then passed on to AI.

'OUR_LEFT_CORNER'
'OUR_RIGHT_CORNER'
'THEIR_LEFT_CORNER'
'THEIR_RIGHT_CORNER'
'OUR_LEFT_GOAL_POST'
'OUR_LEFT_GOAL_WALL'
'OUR_RIGHT_GOAL_POST'
'OUR_RIGHT_GOAL_WALL'
'THEIR_LEFT_GOAL_POST'
'THEIR_LEFT_GOAL_WALL'
'THEIR_RIGHT_GOAL_POST'
'THEIR_RIGHT_GOAL_WALL'
'OUR_LEFT_BOX_WALL'
'OUR_LEFT_BOX_FRONT'
'OUR_RIGHT_BOX_WALL'
'OUR_RIGHT_BOX_FRONT'
'THEIR_LEFT_BOX_WALL'
'THEIR_LEFT_BOX_FRONT'
'THEIR_RIGHT_BOX_WALL'
'THEIR_RIGHT_BOX_FRONT'
'CENTER_OF_FIELD'
→ height

Table 7.1: LUT mapping of real word colors to an integer value.

The Field points used are listed in Table 7.1

These points are sequentially selected from the picture using the goal lines and field boundaries. The image to real world translation is carried out with the help of the distortion calibration parameter file.

The field calibration essentially involves finding real world coordinates from given image coordinates by interpolating data obtained from the Distortion Calibration parameter file. This translation is completely based on trigonometric derivations. It uses the basic mathematical principle that given three points, they uniquely define

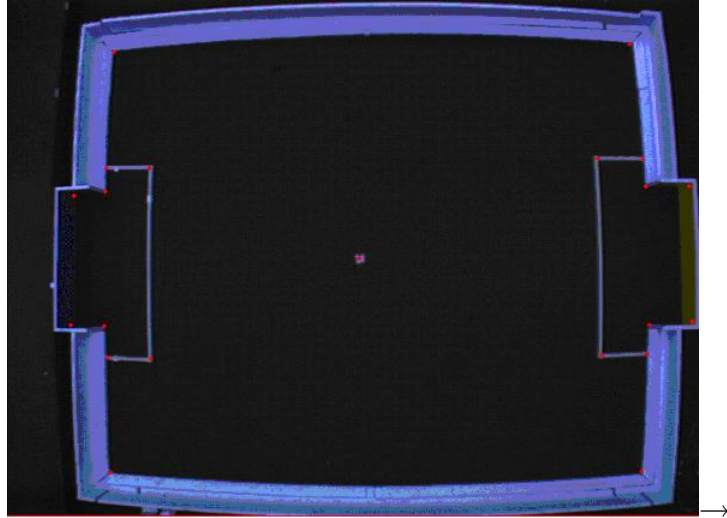


Figure 7.2: AI Calibration Relevant Points.

a triangle. Also used are the properties of similar triangles to map between the real and the image planes.

The interpolation algorithm used for field calibration can be explained as follows. There exists a one-to-one mapping between the image plane and the real plane. Thus each point in the real plane has one and only one corresponding point in the image plane and vice versa. Thus for any triangle defined in one plane, there exists a unique corresponding triangle in the other plane.

Consider the following case shown in the figure. Subscript i denotes image plane and subscript r stands for real plane

Let points A and C be points with known real as well as image coordinates. The image coordinates of point B are known and the real coordinates have to be found out. Due to the 1-1 correspondence between the two planes, the two triangles shown in the figure are similar.

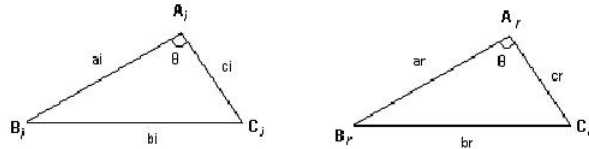


Figure 7.3:

As the coordinates of all three vertices in the image plane are known, the distances a_i , b_i , c_i can be easily calculated. Distance c_r is also known as the real coordinates of A and C are known.

From the properties of similar triangles,

$$\frac{a_i}{a_r} = \frac{b_i}{b_r} = \frac{c_i}{c_r} \rightarrow$$

Thus,

$$a_i = c_i * \frac{a_r}{c_r}$$

Note that this is a relative distance with respect to the triangle. The absolute real location can be determined with the additional orientation information.

Using cosine rule

$$\theta = \cos^{-1}\left(\frac{a_i^2 + c_i^2 - b_i^2}{2 * a_i * c_i}\right)$$

Thus,

$$b_{x-real} = a_{x-real} \pm | a_r * \sin(\theta) |$$

and

→

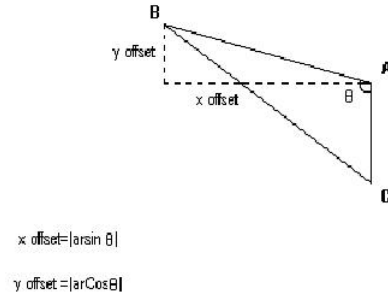


Figure 7.4:

$$b_{y-real} = a_{y-real} \pm |a_r * \cos(\theta)|$$

After acquiring the image coordinates of the field points, the three closest entries to that point are found from the distortion calibration parameter file. Choosing two of these three points at a time, two such triangles can be considered. The two points from the parameter file would correspond to points A and C in the explanation above. The two triangles considered in such a way give two real world values for point B; the mean is then calculated to get the final coordinates.

Due to the way the distortion calibration is designed, the distance between two adjoining points in the calibration grid is 0.1m. Thus $c_r=0.1$ and the equations are simplified accordingly. →

The reason for approximating over two triangles is that because in the real world the grid points are located at right angles, however this may not be the case in the image plane.

From the location of the field points, and thus the corresponding closest points, the triangle obtained can be classified as follows: Points A, C' and C'' are points from

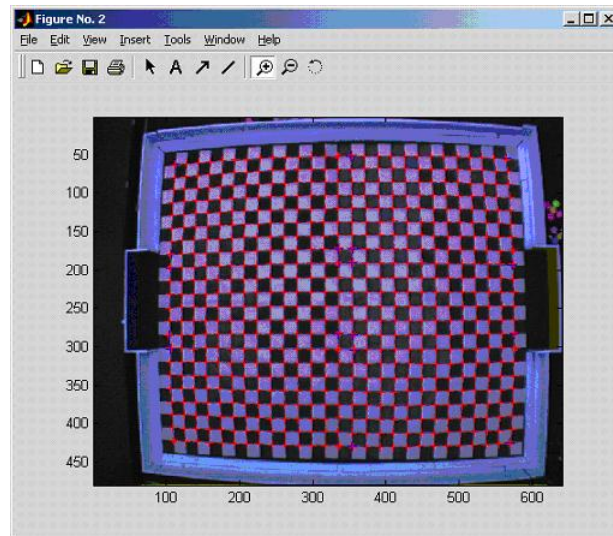


Figure 7.5: Distortion Calibration Grid Points.

→

the distortion calibration parameter file. The real coordinates of point B are to be determined.

1. For points at the four corners of the field: Closest points form a right angle triangle with known orientation.→

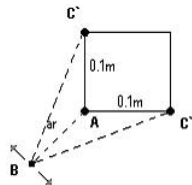


Figure 7.6:

Figure 7.6 shows orientation for one of the field corners, the other three corners

will have a similar orientation though rotated about an angle of 90, 180 and 270. Point A will always be the closest of the three points obtained from the parameter file.

2. For points at the two ends of the field: Closest points are on a straight line

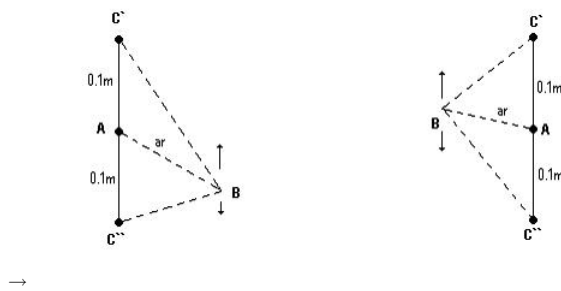


Figure 7.7:

Figure 7.7 shows the triangle orientation for field points on the two goal sides. Point A will always be the closest of the three points obtained from the parameter file.

3. For points in the middle of the field: Closest points form a right angle triangle with unknown orientation

Figure 7.8 shows the case where the field points are in the middle of the field. Any three points from the four grid square corners could be chosen as the three closest points. Point A has to be first extracted as the central point of the closest points before calculating the real coordinates of the field point, B.

In all three cases, the real world coordinates are found with respect to the real coordinates of the center point from the calibration grid, i.e. the point which is

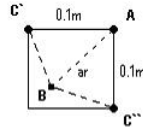


Figure 7.8:

at a distance of 0.1m from the other two points. In the first two cases, this point corresponds to the closest of the three points whereas for the third case it has to be determined by comparing the three closest points. The offsets are added or subtracted depending on the relative positions of the field point and the central closest point in the image plane.

7.2.1 Other Considerations - Edge Detection

In an attempt to fully automate the distortion calibration routine, a first approach considered was a simple edge detecting algorithm followed by feature extraction using correlation and masking. A checker box calibration mat was employed to create an array of distinctive sampling points (intersection points) with known real world coordinates, relative to the field.

The following two schemes were tested:

1. Sobel edge detectors The Sobel edge detection mask was passed over the entire image to extract the intersection points at the boundary of four white/black squares on the calibration mat. This was then followed by a thinning algorithm to obtain one pixel width edges. A '+' shaped mask was then passed over the

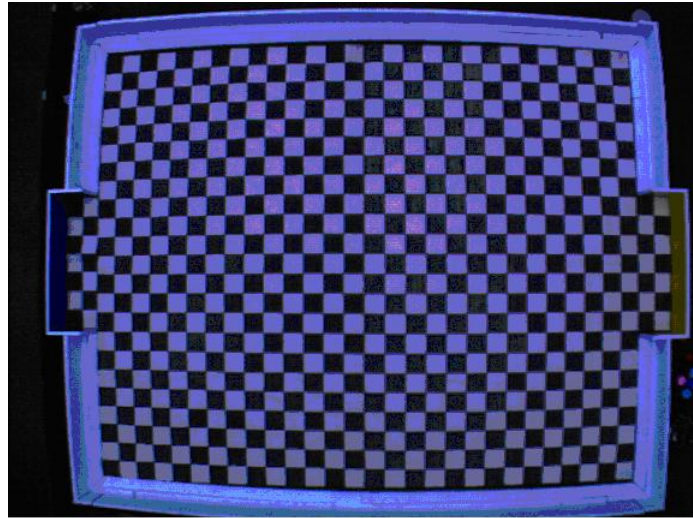


Figure 7.9: Input Image.

entire image as such a mask is highly correlated with the intersection points. Thus the locations with higher amplitudes now correspond to the intersection points. Finally another round of parsing is carried out to obtain the intersection points in a sequential order from left to right and top to bottom of the field so as to map them to the known real world locations.

2. Canny edge detectors In this approach, the Canny edge detection process was implemented. The thinning step was thus avoided as this scheme outputs a single pixel width edge. The rest of the implementation is the same as that described for the Sobel detection scheme.

The final step of parsing is extremely difficult to implement as there is a random variation of 4 or 5 pixels in both x and y direction and hence it is difficult to extract the intersection points in the correct order. This also involves multiple parses over the entire image.

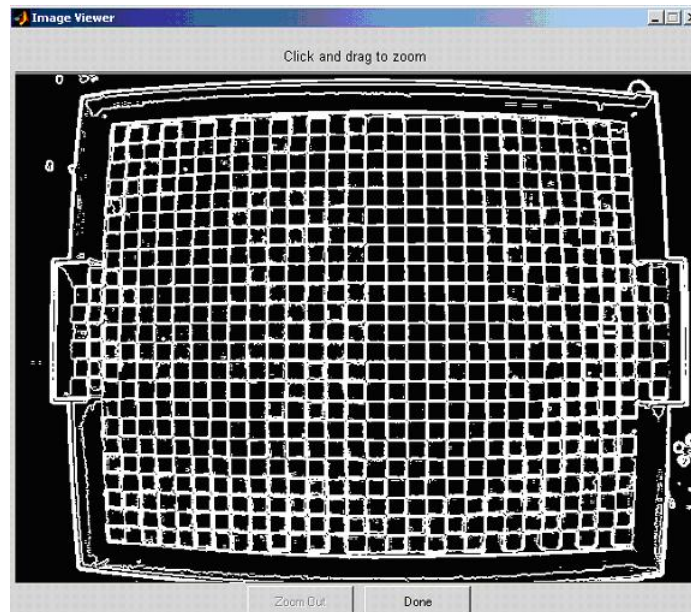


Figure 7.10: Output of edge detection.

This approach was discarded as the distortion calibration technique due to the following reasons:

1. It is computationally exhaustive as it is completely based on pixel operations. The whole image has to be parsed 3 or 4 times.
2. It is very difficult to differentiate between the edges due to the pattern on the distortion calibration mat and those due to the walls.
3. The outputs of the detector are not uniform enough and are heavily dependent on the lighting conditions at the time the image is captured, i.e. sometimes the detected edges are unconnected.

Chapter 8

Intensity Normalization and Masking

8.1 Motivation

Recently, the international Robocup committee determined some rules that may be placed into effect in the near future. One of these proposed rule changes is the use of natural lighting during games. Currently, the Robocup vision system requires controlled lighting conditions in order to function effectively. If natural lighting replaces these controlled lighting conditions, the current vision system would not be able to reliably detect robots. As such, this work makes a first attempt at solving this problem. The purpose of this experiment is to synthesize a method for locating these robots in an image in natural lighting conditions. This work was started as a CS664 final project and then extended to RoboCup.

8.2 High Level Description

On a very basic level, the current vision system locates and identifies robots via a two step process. First, color segmentation is used to find the location of all colored blobs.

Subsequently, the relative positions of these colored blobs are used to identify which team and which identification number a given robot is. Reducing this procedure even further, you get that a robot is determined by first doing color segmentation and then determining its position. As such, the success of the vision 2002 system is heavily dependant on its ability to accurately perform color segmentation.

Due to the fact that the switch to natural lighting will have the most impact on determining the color of a given pixel, the capability of the vision 2002 system to continue to precisely segment out the required colors may be diminished. As such, the proposed method attempts to identify robots by inverting the aforementioned two steps currently used by vision 2002. The exact location of the robot is found before any calculations are performed regarding pixel color extraction.

This is accomplished using a technique common in computer vision known as masking. In masking, an abstract representation of a robot is created, referred to as the mask, which is passed over an image looking for matches between pixel patterns in the image and in the mask. The best matches found via this process are determined to be robot positions. Thus, we have a maximization problem. Additionally, this process is a forced decision problem because we know that at any given time, there must be exactly ten robots on the field. Once this part of the algorithm has completed, the positions of all ten robots are known and the identification procedure begins.

On the physical robot itself, the distances between the center of the robot and the center of the colored circles are fixed. These distances are converted from physical measurements into pixel units and used as a radial distance to sweep out from the center of the robot. While performing this sweep, the intensity of the pixels within the above radius is checked to determine what color they are. Based on this analysis,

the identification and orientation of the robots can be obtained.

8.3 Algorithm

The algorithm was run on images taken in five different degrees of intensity illuminating the field. The example image that will be used throughout this section had eight robots on it instead of ten. The first image below represents the brightest light configuration used in this experiment. This intensity corresponds to the normal, controlled lighting that is used when the system is run in the lab. The second image represents the darkest conditions that this algorithm was run on. The other three images were taken at equally spaced intensities between the following two images.

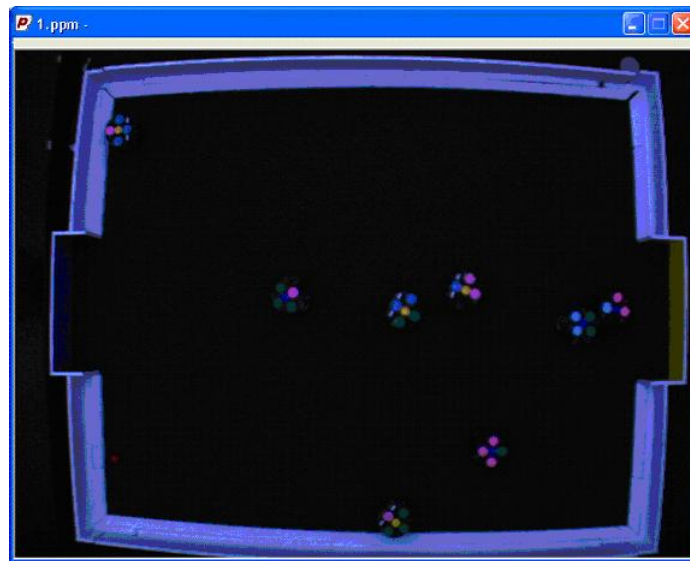


Figure 8.1: Brightest conditions tested, corresponding to normal lighting conditions.



Figure 8.2: Darkest conditions tested.

8.4 Edge Detection

The current vision system grabs images from the frame buffer in .tiff format. However, these .tiff images were converted to .ppm files, for the purposes of this analysis, which are another RGB image file representation. These .ppm images were then converted into grayscale .pgm images to help make the edge detection process smoother. There are several methods for converting RGB images into grayscale images and the following three were considered:

1. $\pi = \text{Max}(r_i, g_i, b_i)$
2. $\pi = 0.3*r_i + 0.59*g_i + 0.11*b_i$
3. $\pi = (\text{Max}(r_i, g_i, b_i) + \text{Min}(r_i, g_i, b_i)) / 2$

where p_i is the grayscale pixel value, and r_i , g_i , b_i are the red, green, and blue components of the RGB image, respectively.

A grayscale image was created for each .ppm image for each of the above conversion types. Subsequently, Canny edge detection was used to convert these grayscale images to unsigned char edge images. Canny edge detection has parameters that must be chosen carefully in order to ensure that the resulting edge image is as clear as possible. Below are the required parameters and the values tested for each of them: `sigma`: min: 1.0 max: 4.0 step: 0.2 `threshold_hi`: min: 0.1 max: 1.0 step: 0.1 `threshold_low`: min: 0.1 max: 0.5 step: 0.1 Canny edge detection was run crossing the above three grayscale conversion types with all of the possible combinations of the above Canny parameters to determine which set of parameters to use in the algorithm. The result of this test was that the first grayscale conversion method was to be used along with a sigma of 2.0, a `threshold_hi` of 0.9, and a `threshold_low` of 0.3. These values were chosen because the resulting edge images had the least amount of noise and the crispiest edges.

8.5 Mask Creation

Before the algorithm is run, there are two processes that must be completed. The first is the choosing of Canny parameters, which has been completed, and the second is the formulation of the mask. Recalling that the mask is an abstract representation of a robot, the mask must be created in such a fashion that it resembles an actual robot. The mask that was chosen is in the shape of a target with a bull's eye. It is a square grid, with an odd number of pixels on each side, which contains five circular sections positioned like a target. Each of these sections starts at a specific radius from

the center pixel of the mask and goes until another specified radius and additionally contains a weight. The weight for each section is based on how likely it is that you will find an edge in this section. An example of this mask appears below. Note how the darker the section, the lower its value.



Figure 8.3: Mask Shape and Size.

The original mask was based on ratios obtained of the robot hat size to the image size, both measured in pixels. This yielded an initial mask size of 25 x 25 pixels. Additionally, the original mask had the following weights for each of the five sections:

1. Large = 100
2. -Large = -100
3. Medium = 50
4. Small = 10
5. Medium = 50

The reason for the -Large section is the fact that in the second section from the center of the mask, you should never encounter any edges. Therefore, a penalty is incurred if an edge is found within this boundary.

After some initial tests with different mask sizes, weights, and radii, the best results were obtained with the following parameters: Mask Size: 31 x 31 Section 1:

Begin radius: 0 End radius: 4 Weight: 100 Section 2: Begin radius: 4 End radius: 6 Weight: -100 Section 3: Begin radius: 6 End radius: 9 Weight: 100 Section 4: Begin radius: 9 End radius: 12 Weight: 10 Section 5: Begin radius: 12 End radius: 15 Weight: 100

Now that the mask has been fine-tuned, the core of the algorithm can be run.

8.6 Algorithm Procedure

The algorithm begins by placing the mask described above in the upper left hand corner of the field. The reason that the mask is placed in this position is due to the fact that it is impossible to find a robot outside of the field, so there is no need to waste resources and time searching for a robot outside of these boundaries. If we suppose that the upper left hand corner of the field is at pixel [x_f , y_f], then the mask gets placed with its center initially at pixel [x_f , y_f]. The mask is then looped over every pixel in the image (for pixels with an x greater than or equal to x_f and a y value greater than or equal to y_f) and the sum of the image pixel values, weighted by the appropriate value in the mask, is calculated and recorded, if necessary. Expressed mathematically, this is equivalent to calculating: $T_j = \sum_i (p_i * m_i)$ for every pixel j such that $x_f = x_j = (640 - x_f)$ and $y_f = y_j = (480 - y_f)$ where i is every pixel covered by the mask, p_i is the intensity at pixel i (0,1), and m_i is the value of the weight in the mask covering pixel i .

As the mask is looped over every pixel j in the image, the parameter T is maximized and the highest ten values of T are always recorded, along with its respective pixel x and y values. Initially, conflicts arose with the locations of the ten best T values that were being stored. Multiple T would occur within the same robot only a

couple of pixels away from one another. This happens if a particular robot happens to fit the modeled mask better than one of the other robots. These conflicts, however, were eliminated by imposing the restriction that 2 or more stored T values could not be simultaneously kept if their respective masks overlapped. If an instance of this did transpire, then the pixel with the highest T value would be kept in the ten best solutions, while all other conflicting values would be thrown out.

At this point, the locations of all ten robots have been found. Each robot center corresponds to a stored (x,y) pixel for each of the maximum ten values of T found in the algorithm above. It is now necessary to determine which robots belong to which team and, if a robot is on our team, which robot number it is and which way it is facing. This is done by looping around a predetermined radius from the center of each robot. As this radius is swept out, each pixel is forcibly classified as one of the relevant colors Blue, Yellow, Cyan, Magenta, or Green based on which theoretical RGB color the pixel is closest to. Once this taxonomy is completed, robot identification and orientation can be found in the same manner as described earlier in this paper.

So to summarize, the algorithm required to locate and identify the ten robots on the field can be represented in this pseudocode:

1. Convert input image into black and white edge image using Canny edge detection
2. Initialize mask and list of ten best solutions T_b
3. Center mask over pixel j and calculate $T_j = \sum_i (p_i * m_i)$
4. If $T_j >$ all elements in T_b , replace lowest element in T_b with T_j and record x and y positions of pixel j

5. Repeat for all such pixels j
6. Classify colored circles within radius r of all $(x,y) \in T_b$
7. Determine orientation and identification based on above colors

8.7 Results

Overall, the mask does quite a good job detecting the robots. When you look at the edge images, you can see that some of the robots came out perfectly. Conversely, some of them have some noise and distortion around them. The centers of the robots whose edges closely resemble the real world image were found accurately. This is what is expected because the mask is built for theoretically ideal conditions. The distorted robots were found with their centers slightly off. However, these values were close enough to the center that the error is small enough to still consider them robot positions.

When the program is run on the original image, and on one that is in near darkness, the results are nearly the same. All robots are found. However, in the darker images, some of the circle markers on the robots are not detected so the center is found slightly shifted from the center found in the image in controlled lighting conditions. This can be seen in the following two figures which correspond to the solutions to the two figures found above.

In the above pictures, a cross represents the center of a robot as found by the algorithm. You can see from both of these figures that every robot was found in both the brightest and darkest lighting conditions. Clearly, robots that had all of their circles show up accurately were found with better precision. Some robots, as can be

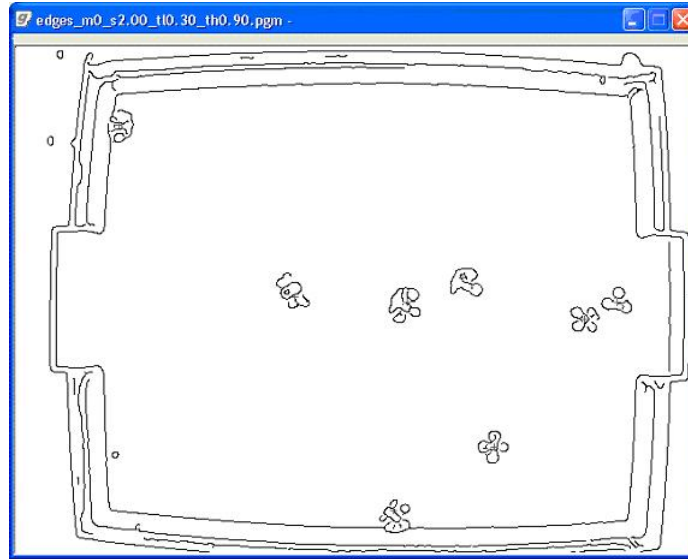


Figure 8.4: Robot positions for highest intensity.

seen in the color images above, have white tags on them. These tags caused some distortion while trying to find the center of the robot, however the algorithm was still able to find the center fairly accurately. If these tags are removed, then the centers will most likely be more precise.

The classification of the colors, however, did not turn out so well. There is much work to be done in this area.

8.8 Conclusions

The algorithm proved itself worthy of future consideration by accurately locating the position of all robots in any lighting conditions. However, this technique must be perfected before being implemented in the actual system as there are many improvements that can and must be made.

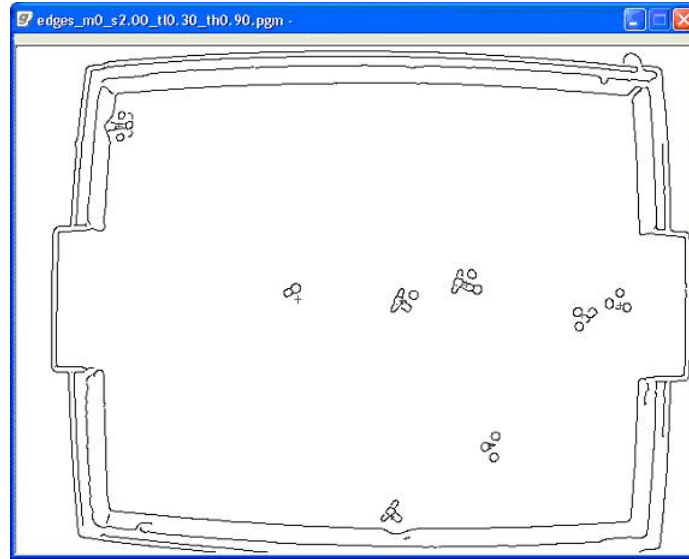


Figure 8.5: Robot Positions for lowest intensity.

First and foremost, the algorithm needs to be modified so that it does not consume so many CPU resources and so that it does not take so long to compute the solution. As it stands now, the algorithm takes way too long to run to be useful in a real time system. Suggested improvements in these areas could be to go directly from the .tiff image to the binary edge image, instead of having all of the intermediaries, and the points given below.

The first of these is to place the mask at every other pixel, or more, instead of every single pixel. The scores for adjacent pixels should be extremely similar and should not have an effect on the algorithms ability to find the center of the robots. In other words, if a pixel is determined to be the center of a robot, selecting an adjacent pixel (or even one 2 pixels away) as the center of the robot is just as sufficient.

Additionally, a mask that represents the playing field could be constructed and placed over the image to find the field boundaries automatically instead of by hand.

This would be beneficial because it would once again cut down the search space required to search for potential robots. Like the robot mask, it would only have to be created once because the field is static from frame to frame.

There is also an optimization that can be performed outside of the code. Theoretically, the circles on the Robocup hats should be a specific diameter and placed at precise locations. However, this is not the case. On some robots, these circles overlap and vary slightly in size. If new hats were meticulously constructed, the mask would be better adept at locating robots.

There is also some work that can be done in the future to add to the functionality of this project. These include determining whether the mask will locate opponent robots as accurately as it finds our robots. Opponent robots can have slightly different configurations. For example, they can use rectangles instead of circles on their hats and can vary the distances between them. They do, however, have to use the same colors and design (one center marker with three or four outer ones.) Additionally, functionality could be added to locate the ball. As it stands now, the ball is not bright enough to be located in dark lighting conditions.

Chapter 9

Conclusion

Developing a real-time vision system that is able to determine all the necessary information for the RoboCup AI is very difficult. The algorithm design and memory management are critical to developing a system that can operate at 60 fps. Documenting this system is just as important since it is the only for way next years team to understand and use the system.

9.1 Achieved Requirements

The main goal for the vision team was to deliver a vision system that had the theoretical latency of less than two frames. The vision team was able to reduce the overall latency of the system to a time of 1.8 frames. Much of the functionality of the vision system for RoboCup has remained the same. A lot of the code was directly used or slightly modified from the Vision 2002 to work with the current vision system. The system is also less complex with both fewer objects and fewer threads. This makes the system a lot easier to understand and to test new ideas. The current vision system provides an excellent foundation for future teams to explore their ideas on how to improve it.

The second goal of the vision team was to provide a more automated system. This includes an automated camera calibration, AI calibration and LUT table generation. All three of these have been significantly improved over those used for the Vision 2002 system.

The last goal was to provide a system to locate the ball in the occluded regions of the field. This code is still being developed but a preliminary system works at 60 fps.

9.2 Unresolved Bugs

9.2.1 Dice System Bugs

There are currently no known bugs with the Dice System.

9.2.2 Aux System Bugs

The Aux system is in development but there are some known bugs. The Aux system is currently very unstable. Sometimes it will run at 60 fps and other times it will not. The solution is either a faster computer or a longer processing time by running at 30 fps. Generation of a distortion calibration file has not been tested with the Aux cameras.

Appendix A

Possible Problems

This is the documentation of the problems that occurred during the development of Dice and Aux that took some time to debug.

1. The color red is showing up as blue on the screen. Check to see if one of the pins on the frame grabber cable is bent wrong. This reinforces the necessity to be extremely careful when plugging and unplugging cables.
2. It is hard to segment out a color for the LUT. Make sure the images from the camera are not too bright by adjusting the iris.
3. The blobs move around a lot once they are segmented. The LUT is missing a color right in the middle of the marker.

Appendix B

Possible Improvements

This is a short list of what could be developed for next year.

1. Time the Vision 2002 erosion operator and verify that it runs faster than the Mil Library function. If this is the case it should be implemented.
2. Look into the Mil Library implementation of a LUT for segmentation.
3. LUT generation should be a stand alone program interacts with the AI computer to generate the LUT.
4. Segmentation should use YUV (HSI) encoding instead of BGR (RGB).
5. Use the Mil Library distortion calibration functions with a new calibration mat.
6. Segmentation could be a threaded operation which may result in faster performance.
7. Do not merge the blobs into one vector after blob analysis. The blobs in the first vector tell the location of the ball and the robots. The blobs in the second vector can be used to determine the orientation of our robot. This will reduce the processing slightly.

Appendix C

General Advice

Dice was developed with the hope that it would be used as the foundation for vision systems in years to come. It has a simple process flow that allows functionality to be removed and tested very easily. The class structure is explained and purposely kept simple so that development can be done with very little understanding of the complete system. Break points are documented above in the code explanation.

1. Spend a week and play around with Image Capture and the Mil Library handbook to get a general understanding of the system. Try to develop your own vision system just to locate the ball or robots.
2. Learn vision as soon as possible because the most exciting part is testing your ideas of how to make the system better. This can only be done once the vision system has been mastered.
3. Time everything. Do not make major changes to the vision system without timing each change.
4. Do not believe the rumors you hear about vision. Trust the documentation for 2003.

Appendix D

Dice and Aux User Manual

The Dice program is located on Antares while Aux is located on Hercules. To run Dice or Aux please follow the following steps.

1. Open the Dice or Aux executable.
2. Make sure the correct parameter file is imported.
3. Press the start button to run Dice or Aux.
4. Press the stop button to stop processing the images.
5. Press Exit to close the program.

The capture button allows a snapshot of the field to be taken and processed by Dice or Aux. This provides a quick way to determine if Dice or Aux is operating correctly without connecting to an AI computer.

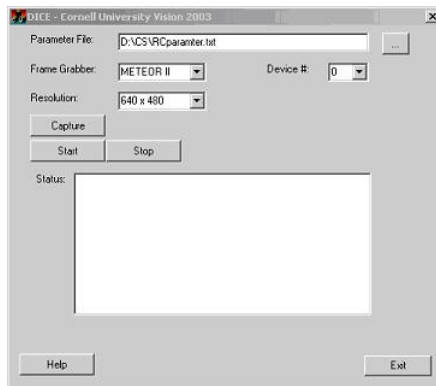


Figure D.1: Dice Main Interface.

Appendix E

CALIB User Manual

E.1 Distortion Calibration

The current version of the Vision 2003 distortion calibration resides on Antares. To run distortion calibration, adhere to the following steps:

1. Place checkered distortion mat onto the field (it is not necessary to use the two goal extension pieces of the mat)
2. If you are not using the colored dots (and are clicking on the section boundaries yourself), goto step 3. If you are using the colored dots (and not clicking on the sections yourself), place each of the 24 fluorescent orange circles at the locations depicted in the image below. In the x direction, the dots are 13, 1, and 12 squares apart. In the y direction, the dots are 7, 1, 5, 1 and 7 squares apart. Ensure that once an image is saved via image capture (see figure E.1), the image has a separation of 13 dots in the x direction on the left side of the image. If this is not the case, the algorithm will not run properly.
3. Open the image capture executable located at:

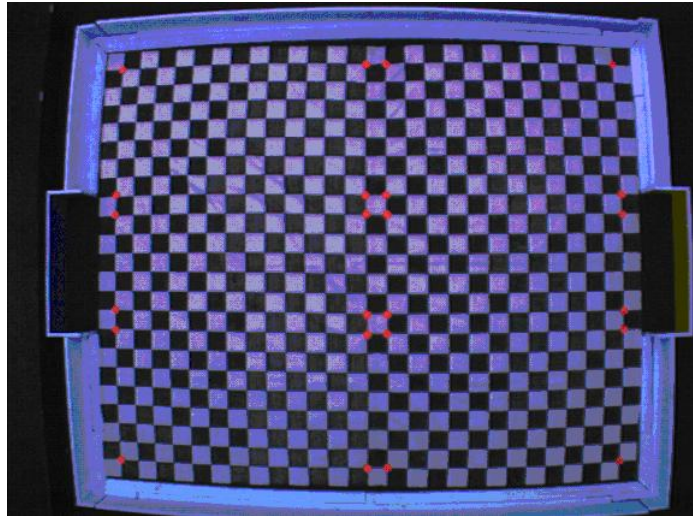


Figure E.1: Distortion Calibration Mat Setup.

'D:/CS/vision2002/development/imagecapture/Release'. You will see the following window:

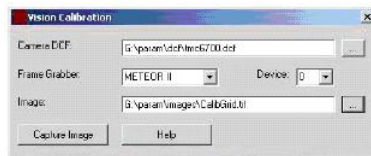



Figure E.2: Image Capture.

4. When running image capture on Antares and on Athena, the required parameters for image capture are different. In the following instructions, parameters in the sentence refer to those required by Antares while those in parenthesis are used on Athena.
5. Click on the  button for the camera dcf and select the following file:

'D:/CS/vision2002/runtime/vision2002/parameters/robocup/main vision/
meteor_II_sony.dcf' (9000vga.dcf)

6. Ensure that the selected frame grabber is METEOR II (GENESIS).
7. Ensure that the selected device number is 0.
8. Click on the button to specify the name and location of the .tiff image to be generated.
9. Click on the capture image button to save the .tiff file.
10. Open matlab on Antares (Athena).
11. Ensure that either the current working directory of matlab is: D:/CS/vision2002/development/distortioncalibration or that the matlab path contains this directory.
12. At the matlab command prompt, type 'distorttest'. This will give you the main distortion calibration window depicted in figure E.3. Note that this figure shows the window after an image already has been loaded.
13. Click on the New/Clear button to specify a name and location of the output text file. This output file should be named robocup1.txt and placed in the main vision folder of the Vision 2002 system that is being run. This is located in the directory
'/vision2002/runtime/parameters/robocup/main vision' of the current vision system that is being used.
14. Click on the load image button and select the image that was saved in step 9.

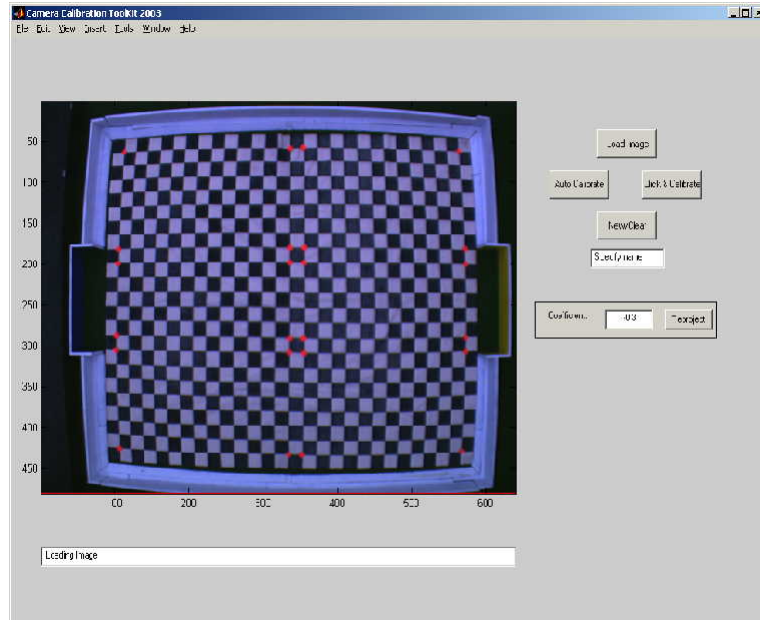


Figure E.3: Distortion Calibration Main Window.

15. This will cause the field image to appear in the main window. If you are using the dots, click on the 'Auto Calibrate' button and goto step 17. If you are specifying corners by clicking, click on the 'Calibrate' button and goto the next step.
16. The status label at the bottom of the screen will direct you where to click. For example, the first position to click on is (1,1). This means that you must click on the corner in the mat corresponding to one square DOWN and one square to the RIGHT. The shift button must be held down when clicking on these points. This is to ensure that errant clicks do not disrupt the calibration process. Additionally, by going to the Tools menu, you can zoom in to get a more accurate depiction of where to click. Shift click on all points specified in

the status label (the last one will be the point (22,27)).

17. Once the status label says that a file has been saved, the window below will pop up. This window shows the locations of all of the extracted corners with red +'s. At this point, distortion calibration has been completed.

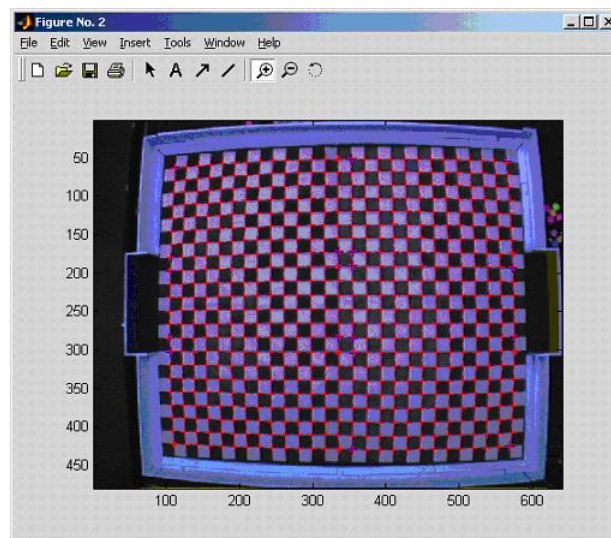


Figure E.4: Corner Extraction Results.

E.2 AI Calibration

1. Note that Distortion Calibration must be completed first before AI Calibration can be done. This is due to the fact that the AI Calibration process depends on output generated by the Distortion Calibration procedure. The distortion calibration generates a final called finalpoint.mat. This file must be in the working directory of matlab in order to perform AI calibration.
2. Ensure that nothing appears on the field, especially the distortion calibration matt, and that the goalie box lines are clearly visible.
3. Follow steps 3 - 11 of distortion calibration user manual (section D.1).
4. At the matlab command prompt, type 'aiCalibGUI'. This will give you the main AI calibration window depicted below. Note that this figure shows the window after an image already has been loaded.
5. Click on the New/Clear button to specify a name and location of the output text file. The name and location of this file is yet to be determined. It is the AI personnel's decision.
6. Click on the Load Image button and select the file created in image capture in step 3 above.
7. Click on the Begin/Restart button.
8. The status label (currently displaying the string "Shift Click on OUR_LEFT_CORNER) will cycle through the areas that need to be clicked on. To ensure no erroneous clicks are made, you must hold down the shift button while clicking the specified location with the mouse.

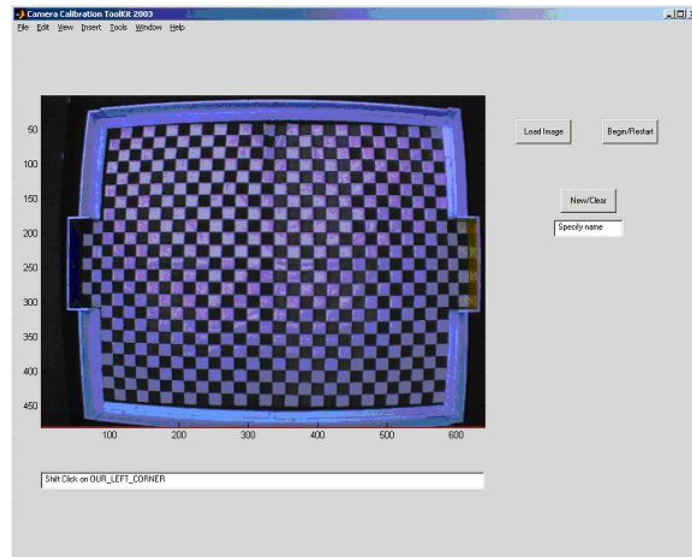


Figure E.5: AI Calibration Main Window.

9. The last position to be clicked on is `CENTER_OF_FIELD`. After this has been clicked, wait a moment until the status label declares that AI Calibration has been completed. At this point, the calibration process is finished and the window can be closed.

Bibliography

- [1] Thibet Rungrokitititoyot, Christian Sigian. *Final Documentation 2001 Vision System*. Autonomous Vehicles & Controls Lab., Cornell University, 2001.
- [2] *The Cornell University RoboCup Vision System*. Autonomous Vehicles & Controls Lab., Cornell University, 2000.
- [3] Chee Yong Lee, Remik Ziemplinski. *A Modular Approach for Real-Time, Multi-Camera Global Vision 2002*, Cornell University, June 2002.
- [4] Chee Yong Lee, Remik Ziemplinski. *User Guide to Vision Pack 2002*, Cornell University, June 2002.
- [5] Chee Yong Lee, Remik Ziemplinski. *Technical Documentation for the Vision Developer*, Cornell University, June 2002.
- [6] Thomas C. Karpati. *Real-Time Visual Perception for Autonomous Robotic Soccer Agents*. M. Eng. Design Report, E.E. Dept., Cornell University, 1999.
- [7] Chee Yong Lee, Remik Ziemplinski. *Color Recognition for Constrained Global Vision*. Autonomous Vehicles & Controls Lab., Cornell University, 2001.
- [8] Raffaello D'Andrea, *Identifying Position and Orientation*(handwritten notes), Cornell University, 2002.

- [9] Roger Y.Tsai. *A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses.*
- [10] Tom Mitchell. *Machine Learning.* 1997.
- [11] R.Jain, R. Kasturi, B.Schunck, *Machine Vision,* McGraw-Hill, Inc, 1995.
- [12] *Mil 7.0 User's Guide.* Matrox Electronic Systems, Quebec, Canada, 2001.
- [13] *MIL 6.1 User's Guide.* Matrox Electronic Systems, Quebec, Canada, 2000.
- [14] Mike Blaszczyk. *Professional MFC with Visual C++ 5.* Wrox Press, 1997.
- [15] Jeff Prosis. *Programming Windows with MFC Second Edition.* Microsoft Press, May 1999.
- [16] *3CCD Color Video Camera: DXC-9000,* Sony Corporation, 1996.