Practical Camera and Colour Calibration for Large Rooms

Jacky Baltes

Centre for Image Technology and Robotics
University of Auckland,Auckland
New Zealand
j.baltes@auckland.ac.nz
http://www.citr.auckland.ac.nz/~jacky

Abstract. This paper describes a practical method for calibrating the geometry and colour information for cameras surveying large rooms. To calibrate the geometry, we use a semi-automatic system to assign real world to pixel coordinates. This information is the input to the Tsai camera calibration method. Our system uses a two stage process in which easily recognizable objects (squares) are used to sort the individual data points and to find missing objects. Fine object features (corners) are used in a second step to determine the object's real world coordinates. An empirical evaluation of the system shows that the average and maximum errors are sufficiently small for our domain. Objects are recognized through coloured spots. The colour calibration uses six thresholds (Three colour ranges (Red, Green, and Blue) and three colour differences (Red - Green, Red - Blue, Green - Blue)). This paper describes a fast threshold comparison routine.

1 Introduction

Our research work focuses on the design of intelligent agents in highly dynamic environments. As a test-bed, we use the RoboCup domain, which is introduced in section 2. In this domain, small toy cars play a game of soccer.

This paper describes an accurate, cheap, portable, and fast camera calibration system (Section 3). After an initial preprocessing step (which is guided by the user), it automatically computes real world coordinates for features in the image (Section 4). Section 5 discusses our algorithm in more detail. The Tsai camera calibration algorithm is briefly described in section 6.

Section 7 shows the accuracy that can be obtained by our method in a sample and a real world problem. Both the average and maximum error are sufficiently small for our application.

Section 8 discusses the blob detection used in our video server. Objects are identified using coloured spots. The colour detection uses the R-G-B colour model. Each colour is identified by twelve parameters. Six parameters identify the minimum and maximum threshold for the red, green, and blue colour channels.

Another six parameters identify minimum and maximum values for the difference channels (red - green, red - blue, and green - blue).

To be able to maintain a frame rate of 50 fields per second without special purpose hardware, the video server uses a number of optimizations described in section 9.

In section 10, we discuss ideas for further research to improve the accuracy of the calibration and to find colour thresholds automatically.

2 The Laboratory Setup

RoboCup [4] is a domain initially proposed by Alan Mackworth ([5]) to provide a challenge problem for AI researches that requires the integration and coordination of a large number of techniques. The problem is to create autonomous softbots and robots that can play a game of soccer.

RoboCup is a difficult problem for a team of multiple fast-moving robots under a dynamic environment that requires the designer to incorporate techniques such as: autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning, robotics, and sensor-fusion. RoboCup also offers a simulation environment for research on the software aspects of RoboCup.

RoboCup is a standard problem which allows the evaluation of proposed methods to solve these problems in a friendly competition. Apart from Machine Learning, which has used databases of problems extensively in research [6], such an agreed upon evaluation method is sadly missing from lots of AI research areas. However, the importance of such test-beds has been realized in other AI fields as well. The planning community agreed on a common domain description language and held the first planning competition in 1998.

The RoboCup environment at the University of Auckland consists of a commercially available cheap video camera mounted on a tripod. The video camera is connected to a video server (a Pentium PC). The video server interprets the video data and sends position, orientation, and velocity information to other clients on the network (three PCs).

Lighting is provided by fluorescent lamps on the ceiling. All the equipment is readily available and most of the room has been unchanged. Although playing soccer is our main objective, there are other tasks that we are working on such as parallel parking and time trials on a race track. Time trials along a race track (called Aucklandianpolis [1]) proved to be very popular with students. Figure 1 shows our environment.

In contrast to all other teams in the RoboCup competition, our camera is mounted on the side of the playing field, which introduces large perspective distortions. Therefore, the geometry calibration is very important.

Since we are often asked to give demos of our system, we needed an accurate, cheap, portable, and fast method for camera calibration.

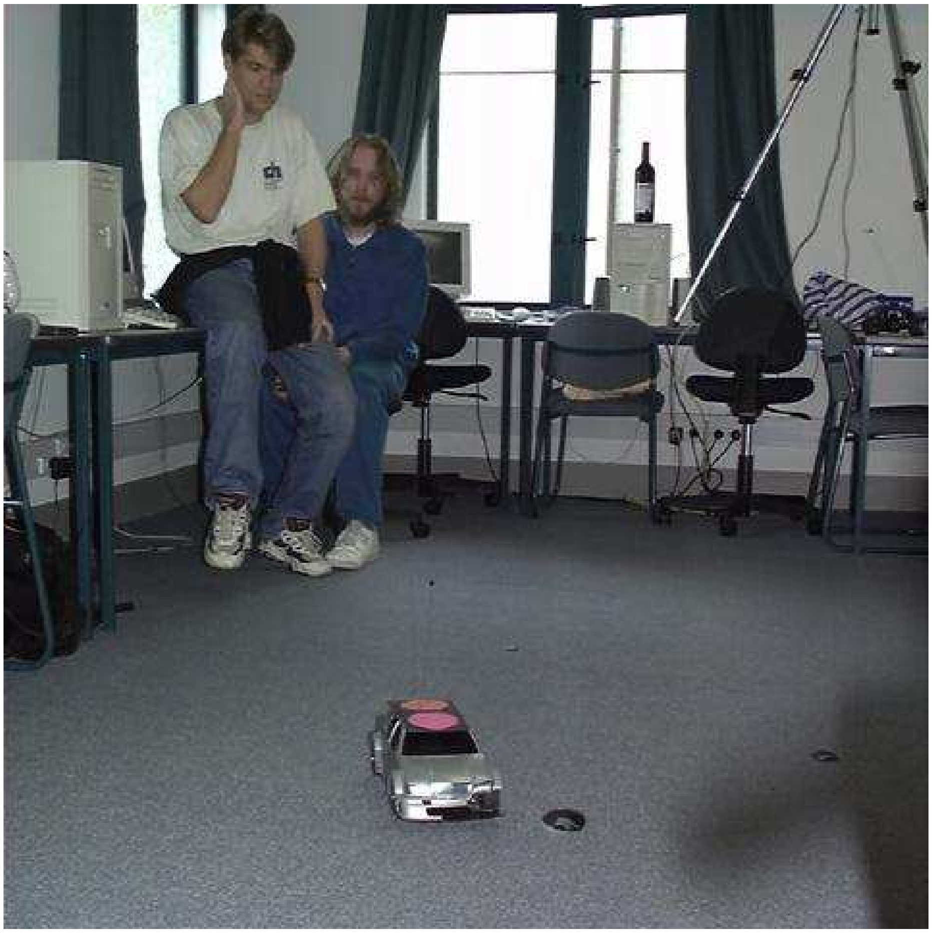

Fig. 1. Aucklandianapolis at the University of Auckland. The tripod of the vision system can be seen on the top right corner of the image. The video camera is just out of the picture. The video server determines position and orientation of the cars by bright dots on the car. As can be seen, the speed trials took their toll on our cars.

3 Camera Calibration

The problem of camera calibration is a very fundamental problem in Computer Vision. The input to a calibration method is a set of known world coordinates and their matching pixel coordinates in the image and the output is a set of external and internal parameters for a camera model. Given this calibrated camera model, it is easy to determine the real world coordinates of image points (if at least one dimension is known) or compute the image coordinates for known real world coordinates.

Traditional camera calibration relies on the availability of known image coordinates for some known world points. For example, a simple pin hole camera model requires that at the real world coordinates of at least 12 image points are known [3]. Once a sufficient number of matching points have been found, well known camera calibration algorithms can be used. For example, the Tsai calibration method uses a complex eleven parameter model with six external and five internal parameters [7]. In our work, we are using a public domain implementation of the Tsai calibration method, which is available from the WWW [9].

This paper focuses on the problem of finding a suitable set of matching points for camera calibration. The need for portability and speed of the calibration method ruled out traditional methods of using feature points inherent in the scene (since these feature points will not be available when moving to different rooms) or of painting feature points into the scene (a labour intensive and error prone task for a large set of points). The creation of a special calibration pattern of sufficient size and with a sufficient number of points was also too expensive. For example, a large wooden board with calibration points (a) would be difficult to move, (b) may not fit into rooms that do not have similar geometry (e.g., a part of the rectangle is cut out by a wall), and (c) expensive and labour intensive to manufacture.

However, we clearly needed a portable calibration pattern[1], so we decided to use readily available and light material. We looked at a number of possibilities including carpets (have a dense texture and are expensive) and linoleum carpets (accurate pattern, but expensive and has an undesirable warping property).

In the end, we decided to use a duvet cover (250x200cm) with a square pattern on it. The back half of the duvet cover was removed to reduce artifacts due to the transparency of the cloth material. The duvet cover is well suited for our environment, since it is easily portable and can be adapted to room outlays[2]. Drawbacks are that the cloth material stretches and warps. Both drawbacks can be minimized through the handy use of an iron. However, they can not be eliminated and thus introduce errors, which limit the accuracy of the camera calibration that can be obtained.

[1] Otherwise our overweight charges when flying to the RoboCup competitions would be even higher

[2] It is also a handy blanket for my graduate students when they get caught up in their work and end up sleeping in the lab

Figure 2 shows a picture of the calibration duvet cover as seen by the video camera.

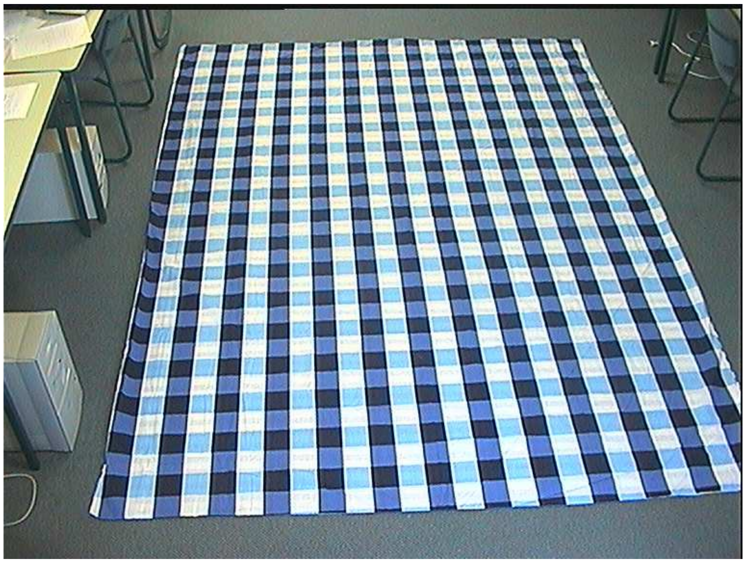

Fig. 2. Calibration Pattern as seen by the Camera

4 Find Matching Points Algorithm

Given the picture shown in Fig. 2, our system uses a semi-automatic method for calculating the matching points. In the preprocessing step, the user removes unwanted parts of the picture, such as the table top on the left side of the calibration picture. Secondly, the colour image is converted into a gray scale image and thresholded, so that only the white squares are left in the image. Currently, we use a global threshold value on the red channel, which was sufficient for our environment.

After this initial preprocessing step, the system automatically computes the matching points. The idea is to find features in the image that can be assigned world coordinates by the known geometry of the calibration pattern (i.e., by knowing that the dimensions of the squares are $8.0 \times 8.1cm$). A false colour image of the result of the preprocessing step can be seen in Fig. 3. The figure shows some of the practical problems in assigning real world coordinates to image features: (a) some of the squares are missing from the right side of the image, and (b) some parts of the squares are missing (e.g., in the bottom right corner).

First, the system uses a simple pattern (5 by 5 pixel squares) to find the white squares in the picture. This step ignores small artifacts and handles missing

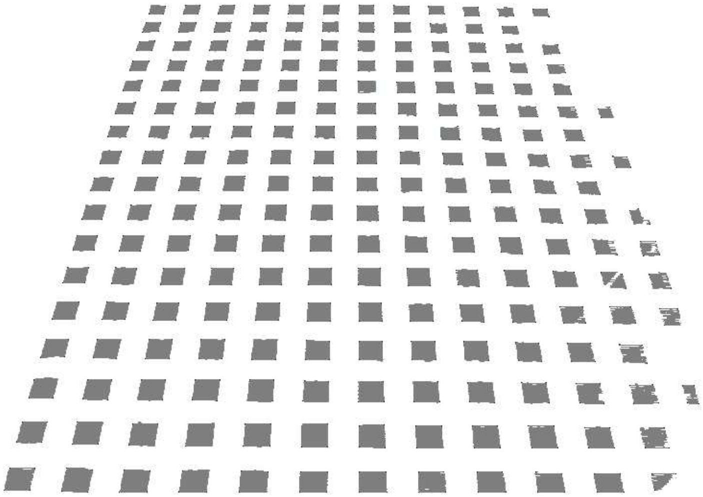

Fig. 3. Calibration Pattern after Preprocessing

squares. The centre point of each square is computed by calculating the moments along the x and y direction. Then, the squares are sorted. This sorting step is of critical importance, since if it is done in the wrong order, the assigned real world coordinates will be wrong, which will result in unusable calibration parameters.

The following algorithm `find_real_world` is used to sort the squares and to assign real world coordinates to their centres. The algorithm takes an unsorted sequence of squares as input and assigns a real world coordinate to the centre of each square. First, the squares are sorted in increasing order of their y coordinate (line 3). This is used to repeatedly extract the next row from the sequence. A row is defined by an initial sequence of squares from `y_sort_squares`, whose y coordinates are within the tolerance limit `eps`. The system also initializes the variable `guess_y`, which is used as a guess of the distance in pixels between the previous and the current row. Lines `8-12` calculate the ratio of the actual distance between the previous and the current row to the current estimate. This ratio is used to check for missing rows in the input image. The current y coordinate `Wy`, and `guess_y` are updated in lines `13-14`. Similarly to the rows, the squares within a row are then sorted based on their x coordinate (line 16) and an x coordinate is assigned (line 24) based on a guess of the distance in pixels to the next square `guess_x` (lines 20-23 and line 27).

Note that the estimates to the next row and column are adaptive, so this method will work in pictures with obvious perspective distortion (as can be seen in Fig. 3) as long as the change from one row to the next is not more than 50%.

```
Procedure find_real_world_coords(unsorted_squares) {

  y_sort_squares=sort(unsorted_squares,y-direction);
  guess_y=0; prev_avg_y=0;
  Wy=0;

  while (row=extract_row(y_sort_square,eps)!=empty) {
     avg_y = average_y_coor(row);
     if (guess_y != 0)
        factor = (avg_y-prev_avg_y)/guess_y;
     else
        factor = 0;
     Wy=Wy+factor*SQUARE_Y_DIMENSION;
     guess_y=avg_y-prev_avg_y;

     x_sort_squares=sort(row,x-direction);
     guess_x=0; prev_square=null;
     Wx = 0;
     foreach square in x_sort_square {
        if (guess_x != 0)
           factor=(square.x-prev_square.x)/guess_x;
        else
           factor=0;
        Wx=Wx+factor*SQUARE_X_DIMENSION;
        square.realworld_x = Wx;
        square.realworld_y = Wy;
        guess_x = square.x - prev_square.x;
        prev_square = square;
     }
     prev_avg_y = avg_y;
}
```

Table 1. Algorithm for finding real world coordinates

After approximate real world coordinates have been assigned to the centres of all squares, the system uses four edge detection steps to find the coordinates of all four corners. If a corner has been identified, it is assigned a real world coordinate by the geometry of the calibration pattern (for example in the first column, the first top left corner has coordinates $0.0, 8.1$, the bottom left corner of the next square is $0.0, 16.2$ and the top left corner of the second square is $0.0, 24.3$.

This means that the assignment of the real world coordinates to the corners is independent of the assigned real world coordinates of the centres of the squares themselves. This is an important feature in our algorithm, since the centres of objects are distorted by the perspective projection and are moved to the lower end of the picture (see Fig. 4), which means that they are unsuitable for applications that require high accuracy. Of course, given an accurate camera model, this perspective distortion can be compensated for, but this leads to a chicken and egg problem, since we are using this information to calibrate the camera in the first place.

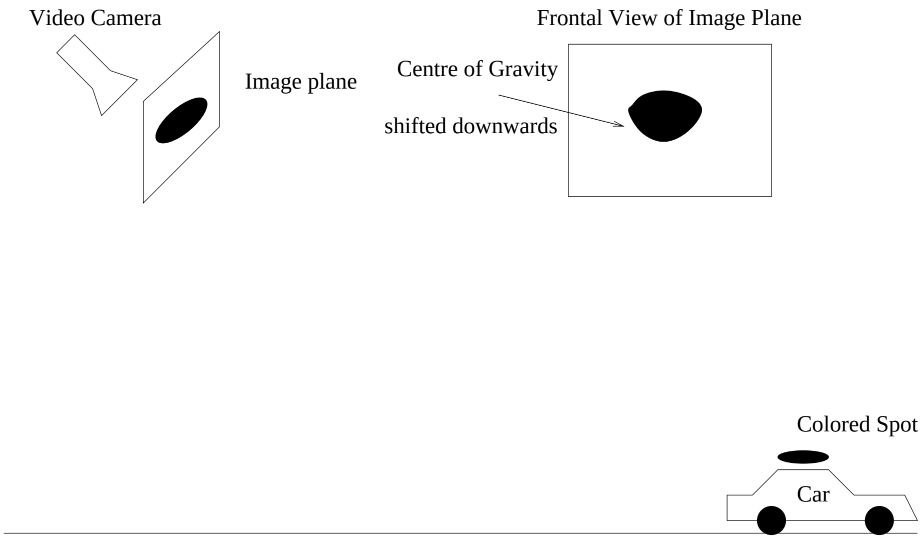

Fig. 4. Movement of the center of a circle under perspective distortion

The real world coordinates of the centres are only used for sorting the squares, which means that only their relative values are important since they are used to determine, which square is the next square in a row or column or whether a square is missing.

Also we found in our tests that this two-stage approach (sort centre of squares, find corners for each square) works better than assigning world coordinates to all corner points. Missing squares or missing data points makes this one step assignment very difficult and error prone.

5 Discussion

The `find_real_world_coors` algorithm assumes that the perspective distortion increases along the y axis. This assumption does not always hold. Sometimes, our camera has to be mounted such that the perspective distortion increases along the x axis. In this case, the user can simply rotate the image by 90 degrees. Note that rotation by 90 degrees simple involves swapping pixel coordinates and does not incur a loss of information.

Given the corners of the calibration carpet, it is also possible to compute the necessary rotation angle. We experimented with arbitrary rotation angles and found that the errors introduced through the rotations were too big and that the calibration data computed was therefore useless.

The algorithm also does not deal with missing squares in the first column. Without knowing the values for the perspective distortion it is impossible to compute where the next row starts and therefore to find out whether the first square in a column is missing.

In practice, both of these aspects are under the user's control. The carpet is manually aligned with the camera coordinate system through visual feed back. The second problem is solved by the user removing any leading columns with missing squares.

6 Tsai Camera Calibration

After the computation of the matching points, we use a PD implementation of Tsai's camera calibration to compute the extrinsic and intrinsic parameters of the camera model.

The Tsai calibration method uses a four step process to compute the parameters of a pin hole camera with radial lens distortion.

Firstly, the position (X_T, Y_T, Z_T) and the orientation (R_X, R_Y, R_Z) of the camera with respect to the world coordinate system is computed. This involves solving a simple system of linear equations. This step translates the 3D World coordinates into 3D camera coordinates and computes the six extrinsic parameters of the camera model.

In Step 2, the perspective distortion of a pin hole camera is compensated for. This step is a non-linear approximation and computes the focal length f of the camera. The output of this step are the ideal undistorted image coordinates.

Thirdly, the radial lens distortion parameters (κ_1, κ_2) are computed. These parameters compensate for the pin cushion effect of video cameras, that is straight lines along the edges of the camera are rounded. An example is seen in the top and bottom row of the calibration image in Fig. 2. The output of step 3 are the distorted image coordinates.

Lastly, the image coordinates are discretized into the real image coordinates by taking the number of pixels in each row and column of an image into consideration.

The last three steps compute five intrinsic parameters of the camera model (focal length, lens distortion, scale factor for the rows, and the origin in the image plane).

The Tsai method is a very efficient, accurate, and versatile camera calibration method and is therefore very popular in computer vision.

7 Evaluation

We evaluated the system in practice (by calibrating different rooms on a number of occasions) and quantitatively through the use of syntheticly generated and real camera pictures.

The synthetic picture was generated by computing a perfect image of all feature points (corners of squares) given our current camera setup (camera mounted on a tripod, $2.58m$ above ground). Since in this case the matching points are 100% accurate, it gives an indication of the maximum accuracy that can be obtained with an eleven parameter camera model.

Given the input image shown in Fig. 2, the corner detection finds 815 corner points. Table 2 summarizes the average error and the standard deviation of the error with increasing number of calibration points n. The data in the table was generated by averaging the results of three cross validation runs for each picture. In each test, n points were selected at random. The camera was calibrated with the data from the calibration points and then the average error, standard deviation, and the maximum error (all in millimeters) were computed.

n	Synthetic Picture			Real picture		
	avg. err	stddev	Max. err	avg err	stddev	Max. err
50	0.9936	0.0653	0.7291	15.2802	7.6748	85.0945
100	0.0964	0.0553	0.3307	17.2455	7.9873	50.0908
150	0.0931	0.0511	0.3068	13.0654	3.8769	37.0576
200	0.0939	0.0557	0.5121	13.8500	5.0923	55.2477
300	0.0904	0.0498	0.3186	13.6753	4.3130	43.3685
400	0.0901	0.0504	0.3207	13.6320	4.2632	56.5799
500	0.0899	0.0497	0.3152	13.5105	3.6942	34.5634

Table 2. Results of the Evaluation. All measurements are in millimeters.

As expected, increasing the number of calibration points improves the calibration of the camera in the synthetic picture. A similar trend can be observed in the real picture.

The differences in errors between the synthetic and the real world image are due to warping of the material and inaccuracies in determining the feature coordinates precisely.

Also, even when using only 150 points, the predictive power of the algorithm is sufficient for our purposes. The error of the calibration is less than $1.3cm$ on average and the maximum error is $3.4cm$. This data is confirmed by testing the accuracy of the coordinates in uncovered areas of the picture (on the very top and bottom of the image). Although, there were no calibration points that covered these areas, the measured error for this region is around $1.5cm$.

The system also proved its worth during competition. The camera calibration was tested at the PRICAI-98 RoboCup and RoboCup-99 competition and proved very stable and fast [2]. For example, it took us less than 15 minutes to calibrate the geometry of our camera system.

8 Colour Calibration

In the RoboCup domain, the ball is a bright orange golf ball and to simplify recognizing the cars they are marked with coloured dots (blue and yellow) or table tennis balls. These dots provide position and orientation information. Since we do not have access to special purpose video hardware, all processing must be done by the video server (Pentium 200MHz).

A simple and fast method for the colour detection is to use minimum and maximum thresholds for the three colour channels R, G, B. In this model, a colour is defined as a cube in the R–G–B cube. This method is not robust enough, since in any practical situation, the colour values will vary greatly with lighting across the field. This means that the thresholds for the colours must be made very large and only a small number of different colours can be detected. In our experience, even when spending a lot of time fine tuning the calibration, it is impossible to distinguish more than four colours with this method.

Although a change in lighting will affect the absolute colour values (e.g., the R, G, B channels are lower in a shadow), the relative distribution of colours is more stable. Therefore, our video server also computes the three difference channels R - G, R - B, and B - G and uses minimum and maximum thresholds for the three difference channels.

The addition of the difference thresholds allows us to detect eleven separate colours reliably, which is sufficient in our domain.

9 Object Tracking

To be able to maintain recognition at 50 fields/sec, the video server uses a number of optimization techniques to reduce computation time: integer threshold comparison, object prediction, and a sampling grid.

9.1 Integer Threshold Comparison

The most frequently used subroutine in our video server is the colour matching routine. Therefore, it was a natural choice for optimization.

Firstly, we tried standard improvements such as hand coding the routine in 80X86 assembly language. Secondly, we even used the special purpose MMX instructions. Neither of these approaches led to the hoped for improvement. The Assembly language implementation only led to 5% speedup. The MMX routine run somewhat faster, but stalled the FPU which slowed down subsequent computations of the real world coordinates. Both approaches, of course, have the additional disadvantage that they are specific to the Pentium CPU and are thus not portable to others architectures.

Therefore, we looked to a general solution that would make use of the following facts:

– Most modern processors support word (32 bit) operations on integer operand-s and word memory accesses.
– Pixels are stored as words (32 bit) in either ARGB (big endian) or BGRA (little endian) format.

The motivation for our approach is to test all three channels R,G and B against the minimum in one operation by interpreting the pixel as a 32 bit word. Similarly, our method only uses one operation in the comparison against the maximum.

Our implementation is based on the realization that subtracting two bit fields will result in a borrow if the first operand is smaller than the second operand. If bits at position i are both 0, then there can never be a borrow. Therefore, if the resulting bit is a 1, it must have resulted from a borrow at position $i-1$.

Our colour threshold routine uses the least significant bit of the alpha, red, and green channel as a stop bit to detect borrows from the red, green, and blue channel respectively. This means that the least significant bit of the colours is ignored. This does not cause a problem, since there is very little difference between for example, a red value of 110 or 111.

The algorithm for our colour thresholding routine is shown in table 3. Variable `pixel` is an integer representation of the pixel value. Variable `lower` is the concatenation of R_{min}, G_{min}, and B_{min} anded with `0x7efefeff` (so that the least significant bits are cleared). Variable `upper` is the concatenation and masking of the least significant bits of the upper thresholds respectively.

The least significant bits of the red and green channel in the pixel are cleared and the lower threshold is subtracted from the pixel. Should a colour channel (R, G, B) be less than the corresponding threshold, a borrow will have resulted in bits 8, 16, or 24. If such a borrow occurred, the routine returns 0, otherwise 1.

Comparisons against the upper thresholds are done similarly by subtracting the pixel value from the maximum threshold.

We use a similar method to calculate and test the difference thresholds R-G, R-B, and G-B. This routine resulted in a 20% speedup in our code.

Table 3. The Colour Threshold Routine

```
int matchColourThreshold(int pixel, int lower, int upper) {
    int ret;

    pixel = pixel & 0x7efefeff;
    if (((pixel - lower) & 0x81010100)||
        ((upper - pixel) & 0x81010100)) {
        ret = 0;
    } else {
        ret = 1;
    }
    return ret;
}
```

9.2 Object Prediction

Another method that we use to speed up the object detection routine is to use the previous position of an object as a starting point for a new search. The video server maintains the X and Y velocities of all object. When looking for an object in the next frame, a new position for the object is predicted using these velocities and a small 32*16 pixel subarea is searched for the object.

If the object is not found within this area, the object is put on a scan queue.

Object prediction works very well in our domain. In over 90% of the times, an object can be found in the predicted region. The reason for prediction failure is most often a fast moving ball, which is deflected or occluded by a robot.

9.3 Sampling Grid

Given the current hardware, we do not have sufficient processing power to scan the whole image even once. Therefore, we use a sampling grid whose size is determined by the smallest object that we are trying to find.

In our domain, these are the yellow and blue ping pong balls, which on the far end of the field are about 6*3 pixels. Therefore, we are using a 6*3 scanning grid.

9.4 Field Mask

As can be seen in the sample picture 2, only about 2/3 of the image contains the actual playing field. The tables on the left side and the top of the picture are not used. The video server uses a mask to distinguish the playing field from the surrounding area. This has two advantages: (a) finding objects is faster since only a sub area of the image must be scanned, and (b) the video server is more robust, since if someone with blue shoes walks through the image it will not be incorrectly classified as an opponent.

10 Conclusion

This paper describes a practical implementation of camera calibration in large rooms. It combines the use of a well known calibration algorithm with a semi-automatic method for computing the matching points.

The method uses a two stage approach. Initial approximations of the centres of objects (in our example squares) are used to sort the objects, but specific object features are used to assign real world coordinates. We intend to use feature detection mechanisms with sub-pixel accuracy, such as the ones described in [8] in the future to improve the accuracy of the calibration.

Object detection is based on blob detection of coloured spots on the car and the ball. The videoserver uses three colour ranges and three difference ranges to identify different colours. Under general lighting conditions, such as the ones that exist during RoboCup, this method allows us to distinguish between up to eleven different colours. A fast integer threshold comparison is used which lead to a 20% speed-up of the video server.

Currently, only geometry and brightness information in the calibration image is used to calibrate the camera. We are currently working on extending the system to compute the colour changes for blue and white squares. This would allow us to estimate the spectrum of the light source. The goal is to compute the colour thresholds for orange, blue, and yellow balls automatically given a single calibration picture as input.

References

1. Jacky Baltes. Aucklandianapolis homepage. WWW, February 1998. http://-www.tcs.auckland.ac.nz/~jacky/teaching/courses/415.703/aucklandianapolis/-index.html.
2. Jacky Baltes, Nich Hildreth, Robin Otte, and Yuming Lin. The all botz team description. In *Proceedings of the PRICAI Workshop on RoboCup*, 1998.
3. K.S. Fu, R.C. Gonzales, and C. S. G. Lee. *Robotics: Control Sensing, Vision, and Intelligence*, chapter 7.4, pages 306–324. McGraw Hill, 1987.
4. Hiroaki Kitano, editor. *RoboCup-97: Robot Soccer World Cup I.* Springer Verlag, 1998.
5. Alan Mackworth. *Computer Vision: System, Theory, and Applications*, chapter 1, pages 1–13. World Scientific Press, Singapore, 1993.
6. C.J. Merz and P.M. Murphy. UCI repository of machine learning databases, 1998.
7. Roger Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4):323–344, August 1987.
8. Robert J. Valkenburg, Alan M. McIvor, and P. Wayne Power. An evaluation of subpixel feature localisation methods for precision measurement. In *Videometrics III*, volume SPIE 2350, pages 229–238, 1994.
9. Reg Willson. Tsai camera calibration software. WWW, 1995.