Kungl. Tekniska Högskolan
CSC

Ball blaster

(Project report in Robotics and autonomous systems, 2D1426)

Author: Anders Boberg, Staffan Gimåker,
Ulf Backudd, Joakim Goldkuhl
E-mail: aboberg@kth.se, gimaker@kth.se,
uffesbox@hotmail.com, joakimgo@kth.se
Teacher: Patric Jensfeldt

Abstract

This report describes how we built and programmed the soccer playing robot Ball Blaster.

Ball Blaster is a wheel driven robot with a camera for vision and various extra equipment such as IR-sensors for sensing the ball at close range, ultrasonic-sensor to avoid driving into walls and a roller for holding the ball. Ball Blaster is built around an Eye Bot microcontroller and is programmed to score goals by pushing an orange ball into a blue or yellow goal. Ball Blaster was entered into a robotic soccer competition and won first place.

Contents

1 Introduction 3

2 Hardware 3

2.1 The Base . . . 3
2.2 Roller front . . . 4
2.3 Sensor turret . . . 5
2.4 CPU board holder and servo mount . . . 6
2.5 Sensors and electronics . . . 6
2.6 Armour . . . 6
2.7 Possible improvements on the hardware . . . 7

3 Strategy overview 7

3.1 The simple strategy . . . 8
3.2 The advanced strategy . . . 8

4 Vision and image processing 10

4.1 Color compensation . . . 10
4.1.1 Image rejection . . . 11
4.2 Camera calibration . . . 12
4.2.1 Image formation theory . . . 12
4.2.2 Lens distortion theory . . . 13
4.2.3 Calibration procedure and results . . . 14
4.3 Classification . . . 16
4.3.1 Gaussian mixture model . . . 16
4.3.2 Adaptive sampling . . . 17
4.4 Filtering and post-processing . . . 17
4.4.1 Median filtering . . . 18
4.4.2 Rejection heuristics . . . 18
4.5 Feature extraction . . . 19
4.6 Image acquisition and motor control . . . 20

5 Localization 21

5.1 Odometric error model . . . 21
5.2 Kalman localization . . . 23
5.2.1 Observation . . . 23
5.2.2 Measurement prediction . . . 24
5.2.3 Matching . . . 25
5.2.4 Estimation . . . 25
5.3 Simulation . . . 25
5.4 Ball localization . . . 26

6 The competition 27

References **30**

1 Introduction

Our strategy was to build a robust and modular robot. The robot should be robust in the sense that we would not program it for special cases that could occur.

We also wanted the program we wrote to be easy to maintain and extend when needed. The same goes with the mechanical part of the project. This was our main concept during the project.

All software development was done in C++ and MATLAB, aided by version control.

2 Hardware

When we started to design the hardware we decided to build it out of four blocks (Base, roller front, camera and sonar turret, and CPU board holder) that could easily be replaced individually. The idea was that we should have a working robot platform on which we could test the software at all times. The advantage of the blocks was that if one of the blocks was not good enough, we could build a new one and quickly replace the block with minimal disturbance to the software testing and development.

We standardized the distance between drilled holes and the diameter of holes so that parts could be reused and relocated easily.

For propulsion we used differential drive which had been successful in previous years and is easy to implement. To sense the environment we used IR-sensors and a camera.

2.1 The Base

The base was the block to which everything else was attached. It consisted of an aluminium plate with the motors and wheels attached. We knew that this part was important to get right the first time, since we needed it at all times to test the software, and also because it consumed a lot of aluminium.

We designed it as a circular disk with a diameter slightly smaller than 18 cm to get some margins to the maximum allowed diameter. The curvature of the disc was cut off at the wheel mounts and it was cut flat at the front were we later mounted a replaceable front.

The wheels were first mounted in slots in the base metal sheet but we discovered that it was time consuming to mount the motors and wheels that way. Therefore we cut away the slots so that the wheels could be slid onto the shaft of the motors without removing the motors from the base.

A bolt was used as support at the stern and smaller bolt was used as a "tipping guard" in the front of the robot. The tip protection was added to prevent the robot from tipping over too much when breaking abruptly from

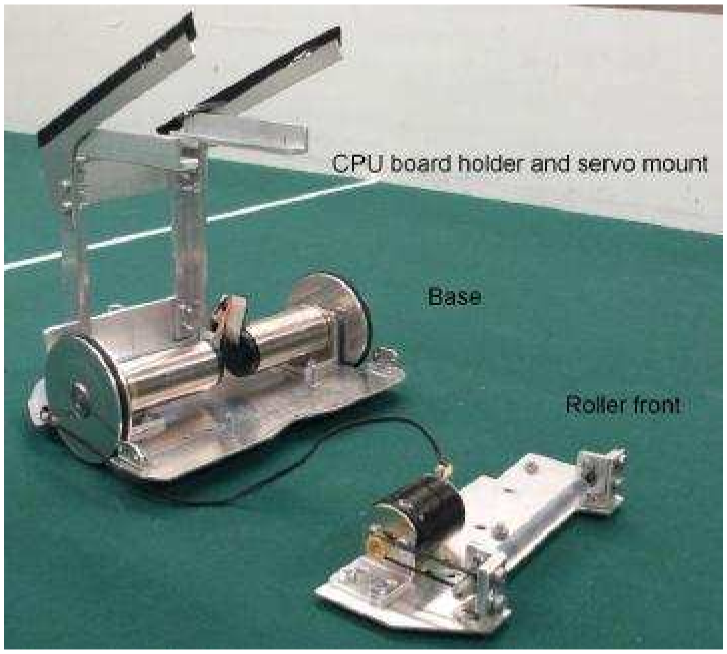

Figure 1: The base roller

full speed. Two M2.5 bolts were fixed to the base front and used as the roller attachment.

2.2 Roller front

To keep the ball close to the robot when driving, we decided to equip the robot with a roller that gave the ball backspin. The roller was made out of a 3mm steel rod covered by a silicone rubber tube. The silicon tube created enough friction against the ball and it worked surprisingly well.

To hold the rod we used ball bearings mounted in aluminium L profiles. The roller was propelled by a small 3.5V motor via a rubber band.

We experimented with a divided rod with a V-shape to get the ball centered on the roller but it did not work as well as we had hoped. Instead we finally used a straight rod with the bearing holders at the sides to stop the ball from spinning away from the robot.

The roller motor was fed with a separate 1000 mAh, 4.8V battery pack. A noise filter, made out of a capacitor between plus and minus poles, was added to the motor cables to reduce the risk of interference from the motor.

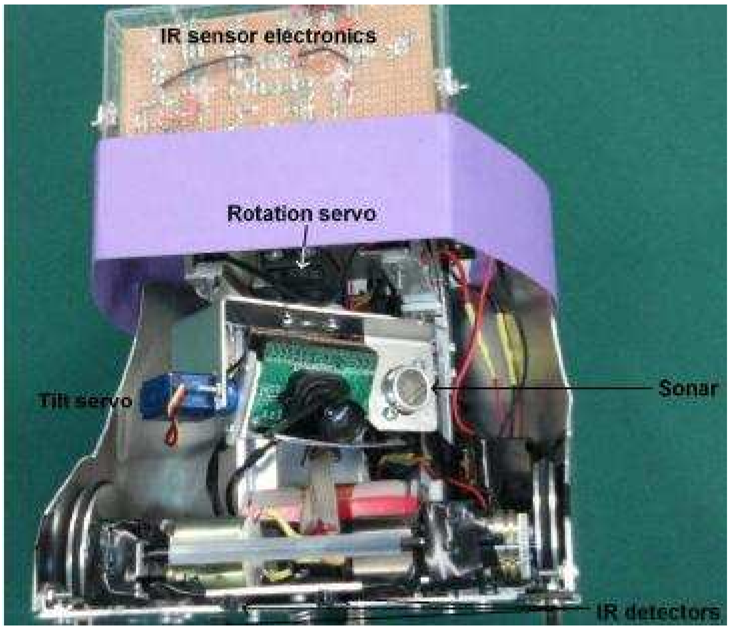

Figure 2:

2.3 Sensor turret

From start we decided to have a camera mount that was movable around two axes, tilt and rotation. The advantage with this arrangement is that it is possible to look for the goal without turning the robot and risk losing the ball.

Another decision we made at the start of the project, was to place the camera below the violet color marking of the robot, which we later discovered to be a suboptimal solution for the camera placement.

We tried several different servo setups. The first attempt was with two standard servos but the limited space made it difficult to fit the camera holder. Therefore we bought a Hitec micro servo (HS-55).

We discovered by trial and error, that the camera must be mounted so that the camera and tilt servo rotates. If the camera and rotation servo is first tilted and then rotated, an unwanted rotation around the optical axis is created.

The final solution was to mount the standard servo upside down and let that servo control the rotation. The micro servo and sonar were then mounted so that both of them rotated. The camera was fixed to the micro servo so that it controlled the tilt angle.

This setup worked well from a mechanical point of view but we discovered too late that it placed the camera in a shadow, causing problems when processing the images.

2.4 CPU board holder and servo mount

To get enough space for the rotating camera and to get a clean design of the robot we mounted the CPU board and the servo holder onto aluminium L-profiles, attached vertically at the rear part of the base. Also the sensor electronics were later attached to this block

2.5 Sensors and electronics

We used IR-sensors to detect if the ball was in the range of the roller. If the ball was close to the sensor, the roller should be started. The output from the detector electronics was also fed into the processor to detect if the robot had the ball or not. We chose to start the roller by hardware because we wanted a short roller response time and reduce the workload for the processor.

The sensors consist of one IR-LED that illuminates the ball and one photo diode that detects if something is illuminated by the IR-LED. Because the output from the photo diode varies less than one volt between "no ball" and "ball close to the detector", an op-amp was used to detect the voltage difference.

We used the open collector op-amp LM339 that gave us the opportunity to equip the robot with four detectors. Two of them were used for ball detection so that the full length of the roller could be monitored and the other two were used as reverse collision detector and lift of field detector.

The output from the ball detector sensors was used to start the roller motor using an effect transistor. Because the forward current gain of our effect transistor was only 10-25, we amplified the output from the op-amp using a small signal transistor.

The reference voltage into the op-amp was created by a voltage divider that it could be adjusted with a potentiometer. It is important to put the potentiometer in the lower part of the voltage divider so that the reference voltage can be adjusted down to 0 volt.

The first IR diodes we used were too bright and caused illumination of the surroundings so that the pictures from the camera were affected. This problem was solved with smaller, less luminous LEDs. The cameras of other robots detect the emitted infrared light but since it appears as white in the picture, (almost) no one complained.

The packaging of the IR sensor electronics was manufactured from an old tape case.

2.6 Armour

The robot was equipped with armor around the backside. This armor made the robot more resistant to collisions with other robots and it also moved the center of gravity further behind the wheelbase, reducing the nodding tendencies when breaking.

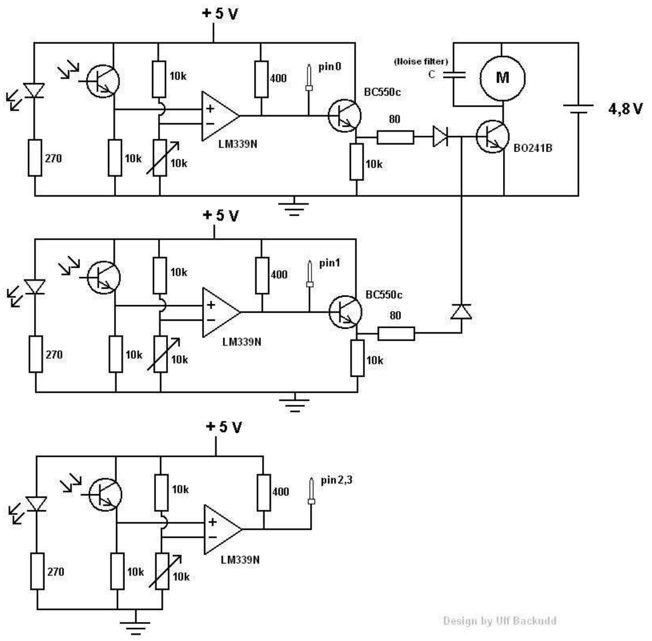

Figure 3: Schematics for the IR-detector controller board.

2.7 Possible improvements on the hardware

We discovered some minor issues on the hardware too late, which could have increased the performance of the robot if we had had the time to fix them. The camera position was the biggest issue. Better pictures could have been received from the camera if we had mounted it higher so that the camera wasn't in the shade.

The second issue was that the roller spun with a speed that created some kind of resonance in the ball. The ball sometimes started to bounce against the roller, sometimes far enough to escape our control. This phenomenon could probably be eliminated if the roller speed was slightly increased.

3 Strategy overview

We developed two strategies in parallel: a simple fall back plan and a more elaborate variant with bells and whistles aplenty.

This first, simple strategy was implemented for and used in lab1.

With the bonus points for the exam were secured, we began to think about a more advanced strategy. In the end we had to abandon the advanced strategy in favor of our fall back due to the inevitable lack of time and a few elusive bugs in the localization code.

3.1 The simple strategy

The basic approach behind the simple strategy was to first find our target, adjust our heading and drive in a straight line to it.

Locating the target was done by turning on the spot and taking pictures. The search was assisted by the panning camera turret which was used as much as possible, being both faster and more reliable.

When the robot saw the ball it drove up to it where the spinner device caught on to it. It proceeded by finding the goal and scoring, using the basic three step procedure described above.

Extra care had to be taken not to drop the ball while turning in search for the goal.

This strategy was quite simple to implement, pretty robust and worked reasonably well but can ultimately not be considered intelligent in any way.

It was implemented as a state machine with the following states:

- START STATE - Start up state, drive forward towards the middle of the field.
- FIND BALL - Search for the ball.
- GO TO BALL - Drive to the ball.
- ADJUST TO BALL - Adjust the heading so it won't miss the ball.
- SCAN FOR GOAL - Look for the goal.
- ADJUST TO GOAL - Adjust the heading so it won't miss the goal.
- SCORE - Drive to goal.
- GOAL GESTURE - Makes a goal gesture, and drive out of the goal.

The state transitions are illustrated in figure 4.

Implementation of the state machine was made with a state manager. It made the code very easy to read, maintain and extend with new states.

When a state has executed it returns a pointer to the next state (which may or may not be the same state) and the manager directs the program to that state.

3.2 The advanced strategy

Given that we could establish our absolute position in the playing field a whole new exciting world of possibilities is opened up – we no longer have to actively search for the goal, path planning and avoiding the opponent is suddenly both feasible and quite simple.

As you probably guessed already the basic ingredient of this more intelligent approach is robust localization. With localization in place we envisioned

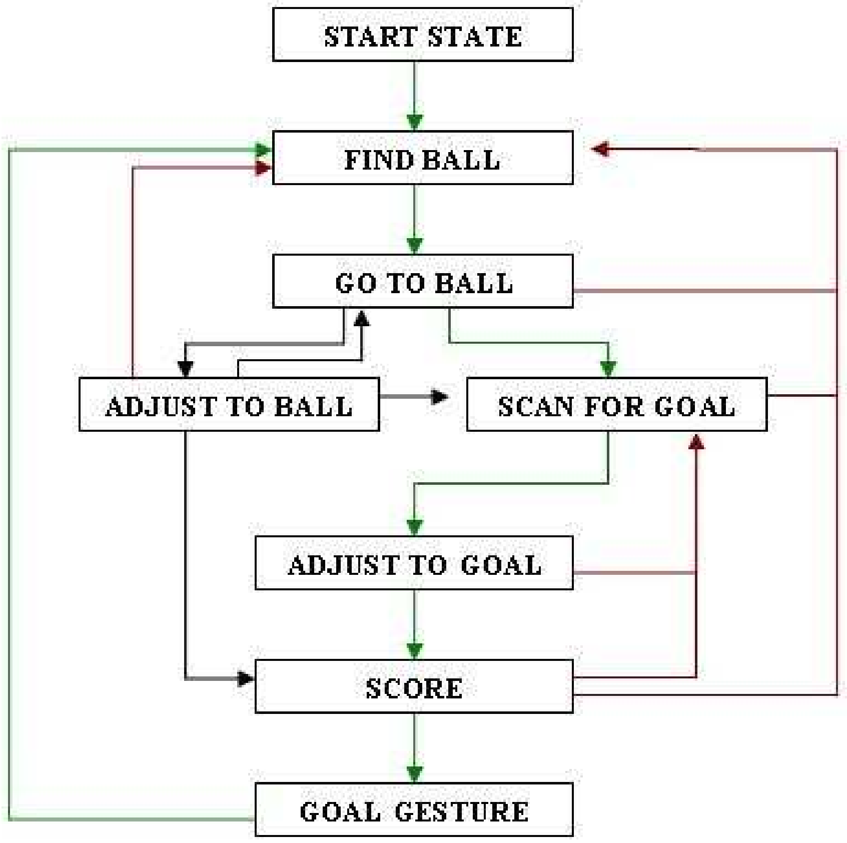

Figure 4: Illustrates all the possible state transitions.

path planning and obstacle avoidance to be implemented using potential field planning.

The state machine approach used in the simple strategy would still serve as the basis for implementing the advanced strategy.

The walls and the opponent would generate positive potential. Negative potential would be generated by the current target, i.e. either the ball or the goal. The potential generating functions would depend on the entire pose of the robot, including orientation. This is desirable since not including orientation in mix implies obvious troubles with driving closely along walls.

If the implementation of this strategy had been completed in time it would probably have been very general and elegant way to navigate the field and score.

4 Vision and image processing

4.1 Color compensation

Perhaps the challenge we spent most time on was getting decent pictures out of the camera. Apart from receiving the occasional strange image the issue we had was extremely poor color constancy, much thanks to the auto-brightness feature of the camera.

Soon after labeling and training on our first batch of training data we started to notice the problem at hand. It would function well for perhaps a day or a few hours only to totally collapse later.

Battery voltage and ambient lighting seemed to cause a significant change in color in the image, as can be seen in figure 5.

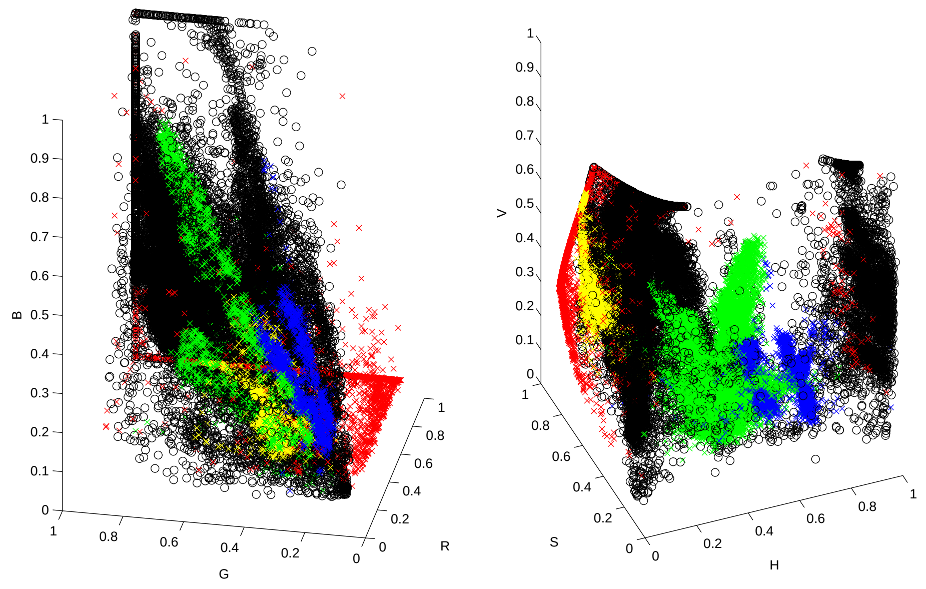

Figure 5: Data obtained in two different runs, taken under different ambient lighting conditions. Different classes are marked using different colors. Note that for most of the classes the data points are dispersed in two distinct blobs corresponding to the two different runs.

At the time, it seemed like a good idea to a one-shot calibration to compensate for the illumination. Alas, so we proceeded to implement it.

The basic idea was to take a picture of a black and white object, analyze it and then compensate all colors so that the black and white appeared as black and white respectively. I.e., given black and white reference colors B

and W transform a color (r, g, b) to (r', g', b') according to:

$$r' = 255 \frac{r - B_r}{W_r - B_r} \tag{1}$$

$$g' = 255 \frac{g - B_g}{W_g - B_g} \tag{2}$$

$$b' = 255 \frac{b - B_b}{W_b - B_b} \tag{3}$$

Furthermore, each component of (r', g', b') is always saturated to lie in the range $[0, 255]$.

Needless to say we ran into a lot of headaches both implementing and using this approach. In retrospect, it's quite obvious that the calibration is practically useless once the auto-brightness adjusts the gain settings.

Once we realized this we tried a second approach – online color calibration, i.e. recalibrate each frame using information in the frame. This is an indirect way of reading the gain values applied to the respective color components.

A small black and white reference was attached below the camera. For each frame, a small number of pixels were sampled from the reference's black and white regions. Using this sample we could determine how black and white were perceived *this* frame. As in our first approach, equations 1 through 3 were used to compensate all colors.

4.1.1 Image rejection

With a successful color compensation scheme in place there was but one hurdle remaining – low dynamic range. This phenomenon was especially bad in the blue channel, where less than 4% of the available range was often used.

The low dynamic range caused noise to have a large impact since its magnitude seemed independent of dynamic range. As a result, as dynamic range tended towards zero we saw more and more blue patches in the image, often in the banded noise usually present in the image.

This problem was greatly exaggerated when the robot was run with it's color marker on, since it made the camera's mounting location an even dimmer place than before. This could probably have been prevented by mounting the camera in a more open space.

Although this problem remained unsolved due to time constraints, a method for detecting bad images was developed.

The idea is very simple: if the dynamic range, measured during calibration using the color reference, is below a certain threshold for one or more channels the picture is discarded and another is taken. As new pictures are taken, the auto-brightness will have time to adapt to the new conditions and result in new pictures with better dynamic range.

4.2 Camera calibration

Being able to reliably measure angles in the image is of great value - it provides the tool needed to track, turn towards or approximate the distance to an object in sight.

A brief review of the required theory is presented here along with our results. The theory serves as a basis for understanding how angles are calculated and which camera parameters we have to recover before doing so.

4.2.1 Image formation theory

Suppose we have a point $P = (X, Y, Z)$ in a three-dimensional space that is projected down on the image plane of a camera with focal length f, as depicted in figure 6. For simplicity, the point P is described in the reference frame of the camera[1].

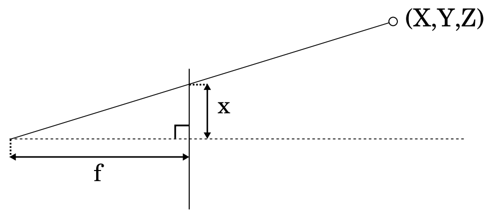

Figure 6: Illustrates how points in 3D-space are projected down to the image plane. The optical axis is illustrated by the dashed line and the image plane by the line perpendicular to it.

We assume that the elements of our image sensor are square and the origin of the image coordinate system is at the intersection with the optical axis. The image coordinate $p = (x, y)$ corresponding to P is then easily calculated using similarity:

$$\begin{array}{c} \frac{X}{x} = \frac{Z}{f} \\ \frac{Y}{y} = \frac{Z}{f} \end{array} \Leftrightarrow \begin{array}{c} x = f\frac{X}{Z} \\ y = f\frac{Y}{Z} \end{array} \tag{4}$$

It's trivial to extend this model to allow rectangular sensor elements and an arbitrary image coordinate system origin. Let (c_x, c_y) denote the origin of the image coordinate system, usually referred to as the camera's *principal point*, and σ the width to height ratio of the sensor elements. The equations then become:

$$x = f\frac{X}{Z} + c_x \tag{5}$$

[1]If it's not, we have to transform P into the coordinate system of the camera before proceeding. This is trivial to do with one or two matrix transformations.

$$y = f\sigma\frac{Y}{Z} + c_y \tag{6}$$

It's often convenient to express this as a linear transformation using homogeneous coordinates and the matrix K formed by the internal parameters:

$$K = \begin{pmatrix} f & \gamma & c_x \\ 0 & f\sigma & c_y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{7}$$

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \frac{1}{Z}K\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{8}$$

Where γ is a skew factor that's non-zero if the sensor elements are rhombic rather than rectangular. For our particular camera, however, this parameter turned out to be close to zero and was thus left out entirely.

4.2.2 Lens distortion theory

The equations derived in the last section only hold for an ideal pin hole camera. In reality, all cameras are plagued by distortion introduced in the lens. In a heavily distorted image angles cannot be calculated using the simple relations presented above, nor is the straightness of lines preserved.

The severity of the lens distortion is usually inversely proportional to the price of the aperture and proportional to its field of view. Thus, it was not surprising that the lens used on our camera introduced clearly visible distortion (see figure 7).

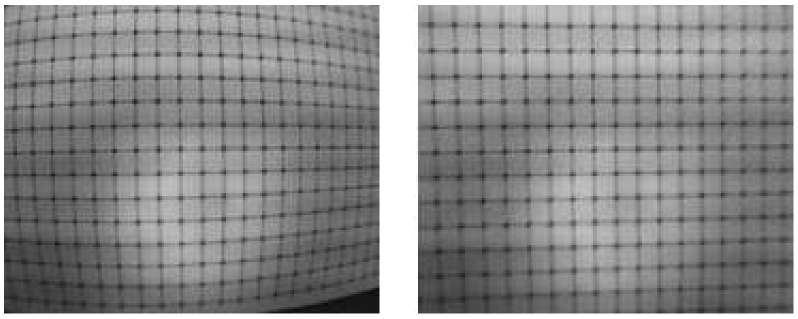

Figure 7: The figure shows the image of a *flat* surface with a grid pattern before and after undistortion.

In the distortion model we employ, each projected image coordinate is distorted by an amount proportional to its distance to the principal point [1]. Given the normalized image coordinate x_n, the distorted coordinate x_d is:

$$x_n = (\tilde{x}, \tilde{y}, 1)^T = \left(\frac{X}{Z}, \frac{Y}{Z}, 1\right)^T \tag{9}$$

$$x_d = L\left(\tilde{r}\right) \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \quad \tilde{r} = \sqrt{(\tilde{x}, \tilde{y})^T (\tilde{x}, \tilde{y})} \tag{10}$$

$$L(\tilde{r}) = k_1 \tilde{r}^2 + k_2 \tilde{r}^4 + k_3 \tilde{r}^6 \tag{11}$$

Incorporating this into equation 8 yields the complete image formation equation:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = K x_d \tag{12}$$

What we can measure in an image is limited to distorted coordinates on the form x_d, whereas what we're really interested in is their corresponding undistorted coordinates x_n. Thus, we want to find the inverse mapping from x_d to x_n.

In our case $k_3 = 0$, which reduces the inverse mapping problem from a pair of seventh order polynomials to a fifth order pair. From equation 9 through 12 we see that the distorted image coordinate (x, y) is related to its real coordinate $(\tilde{x}, \tilde{y})$ by:

$$\begin{cases} \alpha = k_1(\tilde{x}^3 + \tilde{x}\tilde{y}^2) + k_2(\tilde{x}^5 + 2\tilde{x}^3\tilde{y}^2 + \tilde{x}\tilde{y}^4) \\ \beta = k_1(\tilde{x}^2\tilde{y} + \tilde{y}^3) + k_2(\tilde{y}\tilde{x}^4 + 2\tilde{x}^2\tilde{y}^3 + \tilde{y}^5) \end{cases} \tag{13}$$

$$\begin{pmatrix} \alpha \\ \beta \\ 1 \end{pmatrix} = K^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{14}$$

Since no analytical solution can be found for the above equations we used the iterative approximation described in [1] to calculate the inverse mapping.

Calculating the approximation on the robot is not feasible in real-time, instead it's stored in a pre-computed lookup table generated in and exported from MATLAB.

4.2.3 Calibration procedure and results

Our initial plan was to first correct lens distortion in image space, as suggested in [2]. Once rectification is possible the internal parameters K can be recovered using a known 3D object, singular value decomposition and handful of linear algebra tricks [3], [2].

The undistortion process turned out to be less straightforward than it appeared, partly because the required non-linear least squares minimization quite easily got stuck in local minima. Another hurdle is that an imprecisely measured calibration object is likely to incur significant uncertainties to the parameters. Ultimately, this home brew approach was abandoned in favour of a ready-made camera calibration software.

After roaming the web in search for an apt tool we settled on the MATLAB camera calibration toolbox [4].

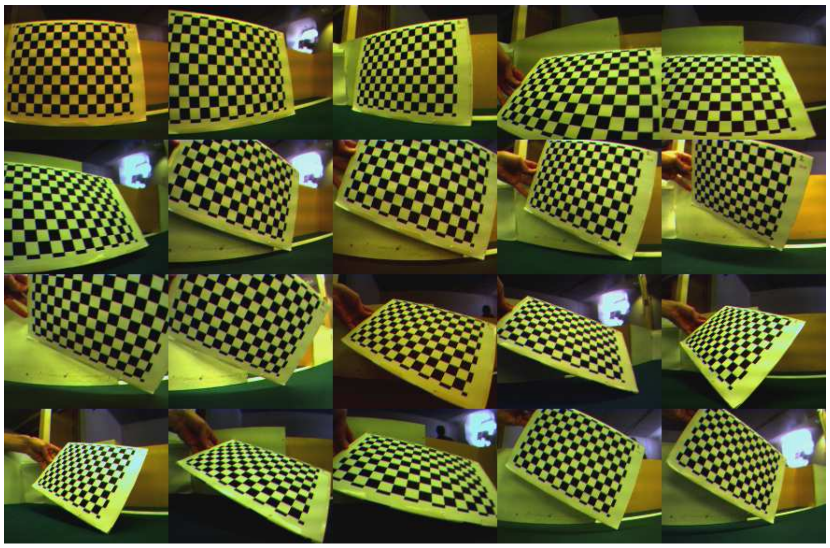

Figure 8: The 20 images used to calibrate the camera.

It was exceptionally easy to use and produced results far better our a layman's attempt ever could have. Only a small amount of manual labour is required and we highly recommend it to anyone looking to solve the same problem.

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 140.47 & 0 & 78.64 \\ 0 & 154.75 & 70.70 \\ 0 & 0 & 1 \end{pmatrix} \pm \begin{pmatrix} 0.39 & 0 & 0.78 \\ 0 & 0.44 & 0.58 \\ 0 & 0 & 1 \end{pmatrix}$$

$$k_1 = -0.32385 \pm 0.00562$$

$$k_2 = 0.11651 \pm 0.00893$$

Table 1: The results obtained from calibration. The numerical errors are approximately three times the standard deviations [4].

The toolbox gives an estimate of tangential distortion due to imperfectly centered lenses in addition to the sixth order radial distortion of equation 11.

However, the obtained values were much to uncertain and thus discarded.

Once all parameters are known, the horizontal and vertical angles to a point are trivial to calculate. Let $(\tilde{x}, \tilde{y})$ be the coordinate after undistortion, then:

$$\tan\theta_x = \frac{\tilde{x}}{f_x} \Leftrightarrow \theta_x = \arctan\frac{\tilde{x}}{f_x} \tag{15}$$

$$\tan\theta_y = \frac{\tilde{y}}{f_y} \Leftrightarrow \theta_y = \arctan\frac{\tilde{y}}{f_y} \tag{16}$$

4.3 Classification

The first multi-class classifier we implemented was the simple and well-known KNN algorithm. A functional albeit inefficient implementation of the KNN algorithm is trivial to do and as such it served as a good first candidate.

An investigation detailed enough to draw valid conclusions considering performance was never done since it was dropped in favor of the other methods described below.

Our second attempt used a support vector machine. However, training was so slow that only a far to small subset of the training data could be used.

All our classifiers were implemented in the same manner on the robot – using a $32 \times 32 \times 32$ pre-computed lookup indexed with the color of the pixel being classified.

4.3.1 Gaussian mixture model

The classification method used in the end was a Gaussian mixture model. The basic idea of mixture model is that instead of trying to fit a single mode distribution to a potentially diverse class, a superposition of several distributions is used. The component distributions are adapted to separate clusters of data within the class using some training algorithm - in our case *expectation maximization*.

The motivation for taking the mixture approach is that upon examination, the pixel data points show high intra-class variations and clustering. Figure 5 illustrates this, although in that particular case the variations are exaggerated due to lack of calibration. One reason for these intra-class variations might be varying lighting conditions for different locations on the playing field. Using a mixture model, we could for example adapt a component distribution to a subset of the images for which the lighting conditions are similar.

As mentioned, the training algorithm used for our mixture model was *expectation maximization*, or *EM*. The one parameter we needed to set before using this algorithm was the number of components distributions to be used for each class. This parameter was set individually for each class after inspecting the distribution of their pixel data points in RGB space.

After setting the number of components, the parameters of the component distributions are randomized, and the algorithm proceeds iteratively in two steps: First, the probability density function for each distribution is evaluated for each data points, and the data points are labeled as belonging to the distribution with the highest density. Next, the parameters of each distribution are recalculated using maximum-likelihood estimation on the data points belonging to that distribution. This is repeated until convergence.

Using the gaussian mixture trained in this manner, we constructed a *maximum a posteriori* classifier. This requires calculating the prior probabilities for each class, which is typically done by taking their relative frequency. Instead of doing this, however, we opted to tune these priors manually to achieve the results best suited to our needs. For example, we lowered the prior probability of detecting a ball pixel, since the classifier had a high ratio of false ball positives. One possible reason for this may have been that our training set had a bias towards pictures containing balls.

4.3.2 Adaptive sampling

It's often the case that large portions of an image are of no interest to the control logic – big patches of the opponent's color tag would be the best example, since we completely ignore that information. Given that, reducing computational load by not classifying the uninteresting portions is an obvious and sound idea.

One simplistic approach is to always only sample a subset of the image. However, that would break parts of our post-processing (see section 4.4.1). As a consequence, we chose to implement an algorithm that initially samples a subset of the image and then expands the search to neighbors of interesting pixels (ball and goal pixels) – i.e. it adapts the time spent depending of the value of the information.

For simplicity, we opted for a uniform initial sampling distribution, as illustrated by figure 9. The main reason for rejecting the idea of sampling more frequently at the horizon was our tiltable camera turret.

4.4 Filtering and post-processing

To battle spurious noise and other misclassified pixels an array of post-processing techniques were invented and implemented.

Noise is a big concern due to the Bayer mask used in the camera, which causes odd colors to appear at sharp color transitions. Another incentive to do some sanity checking on the classification is the atrociously poor color constancy of the used camera.

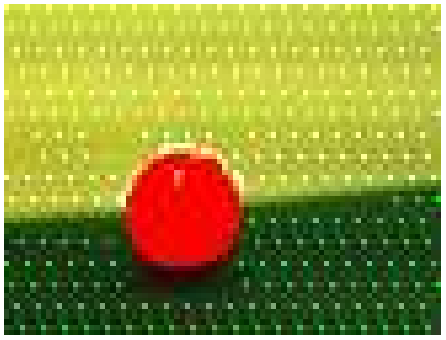

Figure 9: The adaptive sampling in action.

4.4.1 Median filtering

In the image processing field, a standard approach to counteract salt and pepper noise is the median filter. Inspired by it we developed a very similar filter to suppress spurious misclassified pixels.

The filter takes a set Γ of pixel coordinates and outputs a set $\Gamma' \subseteq \Gamma$ according to:

$$\Gamma' = \{(x,y) \in \Gamma \mid |\{(\alpha,\beta) \in \Lambda(x,y) \mid h(\alpha,\beta) = h(x,y)\}| \geq t\} \tag{17}$$

$$\Lambda(x,y) = \{(\alpha,\beta) \in Z \mid |x-\alpha| \leq 1, |y-\beta| \leq 1\} \tag{18}$$

$h(x,y)$ and $\Lambda(x,y)$ denote the classification and neighborhood of (x,y) respectively.

Put in text, a pixel passes the filter if at least t pixels in its 3×3-neighborhood are of the same class.

To save valuable instructions we only filter pixels belonging to classes that we are interested in (ball and goal pixels). For example, the cost of filtering a huge patch of mostly uninteresting grass is seldom justified.

4.4.2 Rejection heuristics

In addition to the median-like filtering the following checks are applied in order:

- If the pixel is above the camera center and classified as a ball pixel it's rejected. Since the camera's height above the floor is greater than the ball's diameter this is impossible.

  This test could also be applied to grass pixels. For performance reasons, this option was not explored – although, in retrospect, this would probably have been a better approach than the one developed below.

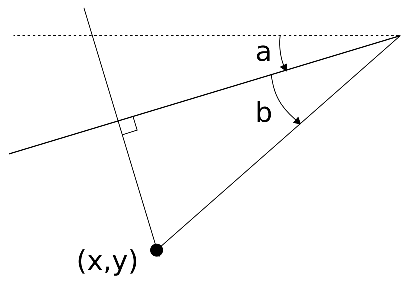

Figure 10: If (x, y) is a pixel classified as ball and $b - a < 0$ it can safely be rejected. a is the camera's tilt angle and b the angle to the pixel in the image.

- Any grass pixels found above a pixel classified as goal can be rejected, knowing that no grass pixels should be found above the floor. This check was developed after it was discovered that the Bayer mask often introduced a thin line of artificially bright green pixels at the upper edge of the blue goal.

  Its imperative that this test is applied *after* the median filtering to reduce the risk of rejecting grass pixels due to misclassified goal pixels.

- All goal pixels below the grass line are discarded. This proved to be an effective tool for removing erroneously classified pixels due to the specular highlight on the ball. Obviously, this isn't perfectly robust and fails when the ball is in close proximity of a wall.

4.5 Feature extraction

To enable localization (see section 5) we need the ability to make observations that can be related to the pose of the robot. Later in section 5.2.1 we shall see how an image coordinate of known height can be used for this purpose.

An algorithm for detecting and locating goal corners in the images was developed, providing four distinct features – a left and right corner feature for each goal.

These particular features were chosen on the merit of their relatively good robustness and ease of implementation. The extraction is independent of the robot's current pose, which avoids coding for special cases and boosts robustness.

If a large enough number of goal pixels are found the feature extraction is triggered. Beginning at the midpoint coordinate of the detected goal pixels, search outwards horizontally until a wall pixel is found. The x-coordinate of the feature is calculated as the mean value of the x-coordinates of the first wall pixel and the last goal pixel found during this search.

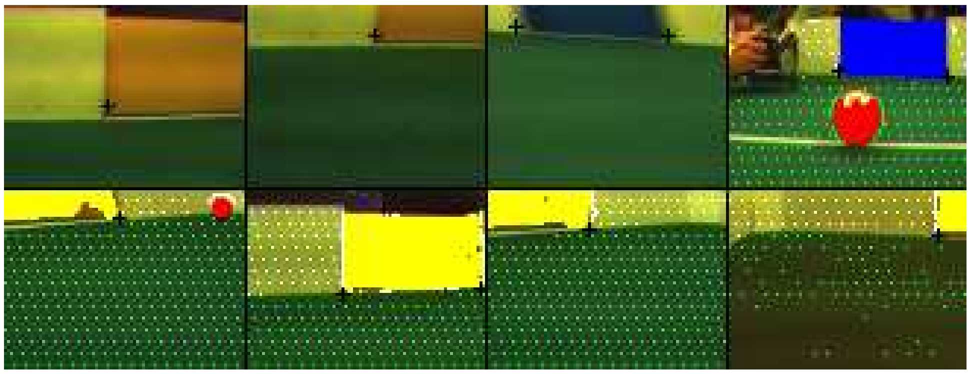
Figure 11: Eight examples of the feature detector in live action – taken by the robot during a live game. The features are marked by the black crosses.

If the wall is found, the search proceeds downwards until the first grass pixel is encountered. This provides the y-coordinate of the feature, calculated analogously but using the last wall and first grass pixel instead.

Although simple and straightforward, this algorithm works surprisingly well, as can be seen in figure 11.

As a post-processing step unfeasible features are discarded using the same technique used to reject ball pixels based on angle, described in section 4.4.2.

4.6 Image acquisition and motor control

The RoBIOS platform used on the EyeBot ships with functionality for controlling velocity and angular velocity of differential drive robots. This type of locomotion control interface is powerful and very simple to use, and thus almost a prerequisite for developing the robot control software.

Due to a questionable hard- and software implementation the built-in controller of RoBIOS severely limited the usable frame rate of the camera. In our case, the quirks and problems of it were worse, to the degree of being fatal – when any wheel was turning the output from the camera got scrambled. Alas, we didn't even have the option of using the built-in controller.

Instead, we developed our on home brew PID-controller for controlling the motors in a similar fashion.

This had the side effect of allowing a higher camera frame rate. We used it with no problems at 7.5 Hz only got a few bad frames at 15 Hz. For comparison, other groups reported that they were limited to frame rates of roughly 2 Hz.

5 Localization

Unless a soccer bot is content with simply following the largest blob of color in its field of vision, like a moth flying blindly towards the nearest source of light, it has to be able to keep track of its position on the playing field. This is called localization, and is typically accomplished by analyzing sensor data, possibly combining several different sources through sensor fusion.

Needless to say, localization is also a prerequisite for most planning methods, since they typically boil down to finding some path from the current position to the most desirable goal position.

Our group aimed to implement localization by using a kalman filter to fuse odometric data with feature measurements extracted from the camera. Unfortunately, due to a few elusive bugs and lack of time, localization had to be dropped before the tournament. However, these bugs have since been eliminated, and the following sections describe the system we had intended to use in detail.

5.1 Odometric error model

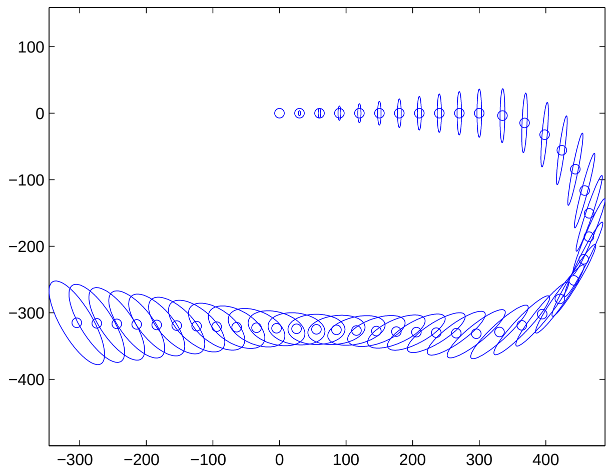

Figure 12: The position uncertainty of the odometric measurements grow in time as a result of error propagation.

An initial approach to localization might be to simply use wheel encoders to calculate the robot's movements relative to some initial position. As we shall soon see, it is rather naive to trust wheel odometry completely, and one quickly realizes the need for sensor fusion.

Our robot uses a differential drive configuration, and thus uses the following rule to update its pose[5]:

$$p' = f(x, y, \theta, \Delta s_r, \Delta s_l) = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \begin{pmatrix} \Delta s \cos(\theta + \frac{\Delta\theta}{2}) \\ \Delta s \sin(\theta + \frac{\Delta\theta}{2}) \\ \Delta\theta \end{pmatrix}$$

Where $\Delta s = \frac{\Delta s_r + \Delta s_l}{2}$ is the distance travelled by the robot, $\Delta\theta = \frac{\Delta s_r - \Delta s_l}{b}$ is the change in rotation, Δs_r and Δs_l are the distances travelled by the left and right wheels respectively, and b is the distance between the two wheels.

Examining this expression, it is easy to see that even a slight error in measurement of the starting angle will cause a drift in position that increases with time.

We assume that the initial pose is a three-dimensional, normally distributed stochastic variables with covariance matrix:

$$\Sigma_p = \begin{pmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{pmatrix}$$

And that the wheel updates Δs_r and Δs_l are normally distributed with the following covariance matrix:

$$\Sigma_\Delta = \begin{pmatrix} k|\Delta s_r| & 0 \\ 0 & k|\Delta s_l| \end{pmatrix}$$

Then the propagation of error in position can be approximated using the error propagation law:

$$\Sigma_{p'} = \nabla_p f \cdot \Sigma_p \cdot \nabla_p f^T + \nabla_\Delta f \cdot \Sigma_\Delta \cdot \nabla_\Delta f^T$$

Where $\nabla_p f$ and $\nabla_\Delta f$ are the Jacobians of f with respect to x, y, θ and $\Delta s_r, \Delta s_l$ respectively.

The parameter k used in the covariance matrix of the position updates, Σ_Δ is a measurement of the uncertainty of the odometry. The wheel encoders typically have very high precision, but there are many other factors weighing in, such as wheel slippage. After some experimentation in a MATLAB-simulation (see 5.3), a value of $k = 0.05$ was chosen. Since we never got to testing our localization system on the soccer field, the validity of this value is uncertain.

In order to have the most recent odometry information available at all times, the odometry update procedure was carried out in the $v\omega$-controller, at the same rate as the PID-controller updates.

5.2 Kalman localization

To combat the ever-increasing uncertainty of the odometric model, we decided to localize using features whose coordinates in the world frame were known. The features of choice for this were the goal corners, which proved to be easily detectable(see 4.5).

The Kalman localization procedure can be divided into the the following steps[5]: *position prediction*, *observation*, *measurement prediction*, *matching* and *estimation*. The first of these, *position prediction*, corresponds to updating the position using the odometry error model and has already been discussed in 5.1. The remaining steps are discussed in the following sections.

It's worth noting that it's not necessary to go through all steps for each position update. Typically, odometry information arrives at a much higher frequency than camera observations. This means that there are usually several prediction steps between each observation.

5.2.1 Observation

The observation step consists of detecting a feature and transforming the measurements of this feature to coordinates in the camera reference frame. Thanks to camera calibration (see 4.2), we are able to accurately measure the x- and y-angles to a coordinate on the camera's image plane. Using these angles, and the known height $h = 75\ mm$ of the camera above the goal corner, we are able to triangulate the relative location of the corner (see figure 13) in the camera frame.

The relative location of the feature is calculated using the following formula:

$$p_f = \begin{pmatrix} x_f \\ y_f \end{pmatrix} = f_f(\alpha, \beta, h) = \begin{pmatrix} \frac{h\cos(\alpha)}{\tan(\beta)} \\ \frac{h\sin(\alpha)}{\tan(\beta)} \end{pmatrix}$$

Errors in the measurements of α, β and h are described by

$$\Sigma_{\alpha\beta h} = \begin{pmatrix} \sigma_\alpha^2 & 0 & 0 \\ 0 & \sigma_\beta^2 & 0 \\ 0 & 0 & \sigma_h^2 \end{pmatrix}$$

These variances should be estimated experimentally, but unfortunately we had no time to do so. The error in (α, β, h) propagates to (x_f, y_f) according to the following approximation:

$$\Sigma_f = \nabla f_f \cdot \Sigma_{\alpha\beta h} \cdot \nabla f_f^T$$

where ∇f_f is the jacobian:

$$\nabla f_f = \begin{pmatrix} -\frac{h\sin(\alpha)}{\tan(\beta)} & -\frac{h\cos(\alpha)\left(1+\tan^2(\beta)\right)}{\tan^2(\beta)} & \frac{\cos(\alpha)}{\tan(\beta)} \\ \frac{h\cos(\alpha)}{\tan(\beta)} & -\frac{h\sin(\alpha)\left(1+\tan^2(\beta)\right)}{\tan^2(\beta)} & \frac{\sin(\alpha)}{\tan(\beta)} \end{pmatrix}$$

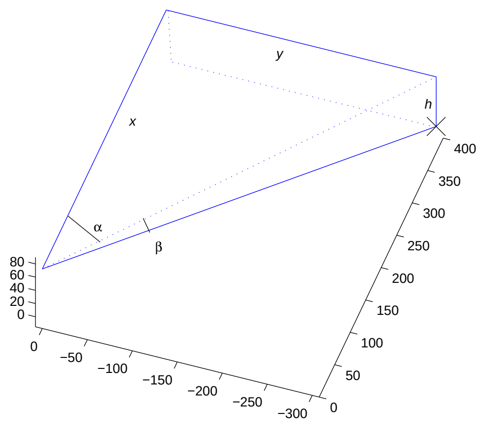

Figure 13: Knowing the angles α and β, and the height h, we can triangulate the relative position of a goal corner feature.

5.2.2 Measurement prediction

This step consists of predicting what the feature coordinates would be in the camera reference frame, given the current estimation of the robot pose and the known feature coordinates in the world frame. We define the following transition function:

$$p_C = f_{pred}(\hat{p}, p_W) = \begin{pmatrix} (x_W - \hat{x})\cos(\hat{\theta}) + (y_W - \hat{y})\sin(\hat{\theta}) \\ -(x_W - \hat{x})\sin(\hat{\theta}) + (y_W - \hat{y})\cos(\hat{\theta}) \end{pmatrix}$$

Where p_C is the feature location in the camera frame, $\hat{p} = (\hat{x}, \hat{y}, \hat{\theta})^T$ is the current estimation of robot pose and $p_W = \binom{x_W}{y_W}$ is the location of the feature in world coordinates. As before, to propagate the measurement error of the robot pose estimation to the relative feature coordinates, we use the error propagation law:

$$\Sigma_{pred} = \nabla f_{pred} \cdot \Sigma_p \cdot \nabla f_{pred}^T$$

$$\nabla f_{pred} = \begin{pmatrix} \frac{\partial x_C}{\partial \hat{x}} & \frac{\partial x_C}{\partial \hat{y}} & \frac{\partial x_C}{\partial \hat{\theta}} \\ \frac{\partial y_C}{\partial \hat{x}} & \frac{\partial y_C}{\partial \hat{y}} & \frac{\partial y_C}{\partial \hat{\theta}} \end{pmatrix} =$$

$$= \begin{pmatrix} -\cos(\hat{\theta}) & -\sin(\hat{\theta}) & -(x_W - \hat{x})\sin(\hat{\theta}) + (y_W - \hat{y})\cos(\hat{\theta}) \\ \sin(\hat{\theta}) & -\cos(\hat{\theta}) & -(x_W - \hat{x})\cos(\hat{\theta}) - (y_W - \hat{y})\sin(\hat{\theta}) \end{pmatrix}$$

5.2.3 Matching

In this step we match the observed features against the predicted ones, calculating the "innovation" of a feature as the difference between the observation and prediction. In the general case, we would not necessarily know which prediction corresponded with which observation. In such cases one usually picks the closes match, provided that it is closer than some chosen threshold. If it isn't, then the observation is discarded.

In our case, we chose to have complete confidence in our ability to ascertain the identity of a feature. We are limited to a mere four features and each of them are clearly distinguishable from the others. For this reason, we were able to skip the matching step. We still calculate the innovation for each observation, however, since it is needed in the final step.

$$v = p_f - p_C$$
$$\Sigma_v = \Sigma_{pred} + \Sigma_f$$

5.2.4 Estimation

In the final step we update our estimation of the robot pose based on the observations we have made. This is done by calculating the *Kalman gain* and applying it to the innovation:

$$K = \Sigma_p \cdot \nabla f_{pred}^T \cdot \Sigma_v^{-1}$$
$$\hat{p}' = \hat{p} + K \cdot v$$

The covariance matrix is updated according to:

$$\Sigma_{p'} = \Sigma_p - K \cdot \Sigma_v \cdot K^T$$

5.3 Simulation

As mentioned earlier, Kalman localization was not finished in time for the tournament. However, some simulations were run in MATLAB using the model presented in the previous sections. In this simulation the robot moved without obstacles and was presented with angular observations of the features every third time step, provided that the features were located in a 90-degree field of vision in front of the robot. Noise was added to both the odometric measurements and the feature observations. Figure 14 shows the movement and believed poses of the robot during a run of this simulation.

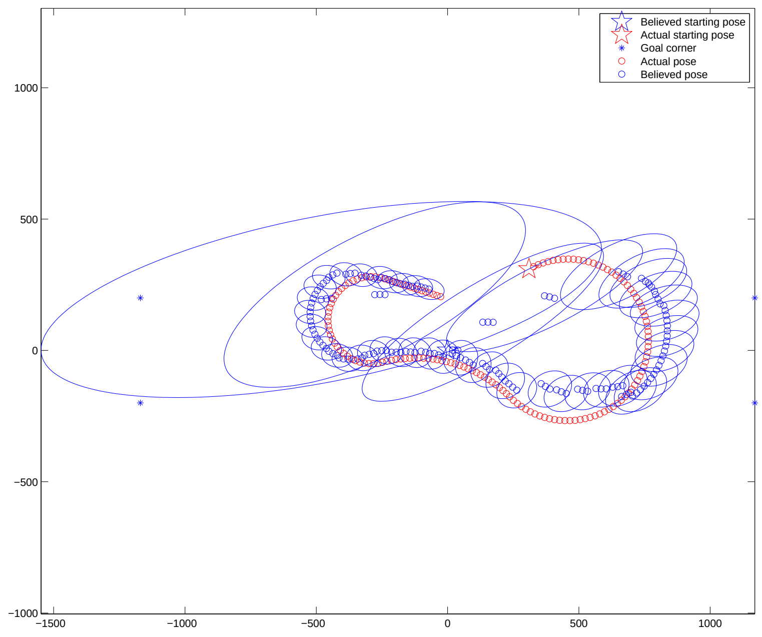

Figure 14: A simulation of kalman localization on the soccer field. A generous amount of noise has been added to the angle measurements.

5.4 Ball localization

Some early experimentation was done on ball localization using the same triangulation method described in 5.2.1. The accuracy of this model was evaluated using a simple test: a set of images, each showing the ball located on the playing field was taken with the robot's camera. In each image, the classification and feature extraction algorithms would estimate the location of the ball in the image, and calculate the angles to it's center. These estimations were then compared against a human user's verdict on the ball location, which we consider to be correct within reasonable precision. The difference was then used to calculate the mean and covariance of the error in angle measurement.

It was found that the computer typically shot low - a fact that can most likely be attributed to the specular highlight from the lighting on top of the ball, which was usually misclassified. After compensation for this systematic error, the error propagation from angles into relative x,y-coordinates was calculated. The results of ball localization on the image shown in figure 16 is shown in figure 17.

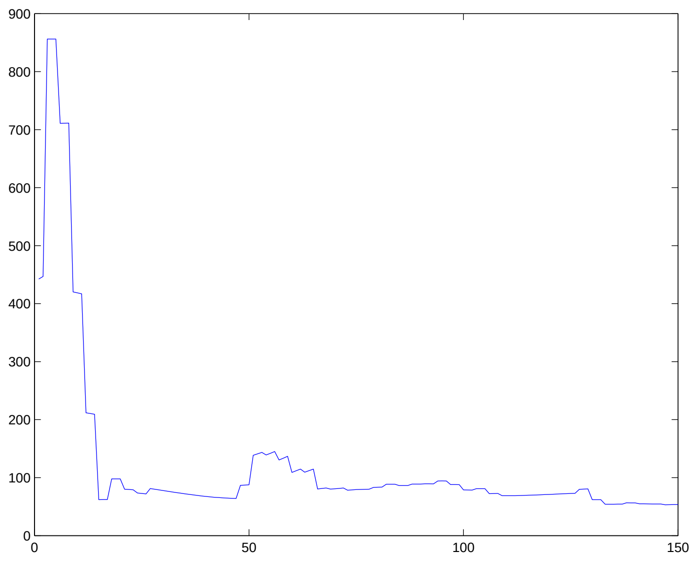

Figure 15: The distance error for the simulation illustrated in figure 14. The error decreases in time as more measurements are incorporated.

The positional uncertainty of ball localization was judged too great for the method to be used effectively. Kalman filtering the ball location may be able to reduce the uncertainty, but would depend on continuously tracking the ball with the camera. We decided that this would not be feasible given the quick movement of the ball, and opted instead to go for the "blind moth" approach.

6 The competition

The competition started out badly for us. One of the other groups had complaints regarding front mounted IR-detectors. Because of the emitted light they saw blue goal pixels in the grass directly below our IR-detectors.

Thus we had to put some tape underneath the detectors so it would not light up the grass. Because of this our roller did not work properly in the seeding round.

We still managed to score one goal and accompanied by five other teams we made it through to the actual tournament.

In our group were Clas Ohlson, Zidane and Ball Blaster (us).

Our first game against Zidane ended 1-0 to our favor.

In our second game we played against Clas Ohlson. Clas Ohlson scored

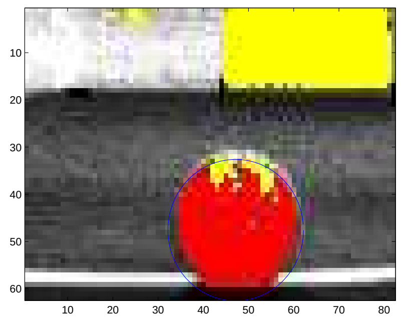

Figure 16: The blue circle shows the feature extraction algorithm's estimation of ball location in the image.

first – this was the only time in the whole tournament that we were not in the lead. But we managed to even the score and then in the very last second we scored the final goal, which made us the wining team with 2-1.

In the semifinal we met Rasdalf. This game was won with the score 2-1 and our chance to partake in the grand finale was secured.

In the final we met RAAS, who had an impressive track record from the group play and semi-finals. We felt that we were underdogs from the start so we had to do a real good match in order to beat RAAS.

We decided to make one last change in our program before the final. The program was tuned to include an initial berserk mode that should initiate the game with a quick thrust and a well aimed shot directly at the goal.

This should be done blindly and with speed none had ever dared try before. Sadly, due to an unknown but quite vicious bug the berserk mode fizzled, transforming the breath taking speeds we programmed it for into a modest crawl.

Even with this bug present, this tactic turned out to work well. We hit the ball (which was not at all clear that we should be able to do) and even though we did not score we got the ball down to the left corner next to our opponents goal.

With this position of the ball RAAS was not able to get a grip of the ball and we got the ball closer and closer to their goal.

When one minute remained of the game we scored. Then something very unfortunate happened – during a wild maneuver to get away from a wall we slammed into a wall so hard that we lost power. This caused our program to be lost and since it was to big to store in the EyeBot ROM memory we couldn't reload quickly but rather had to take the robot out of the field to reload the program from our computer.

This process takes well over a minute and during this time RAAS was

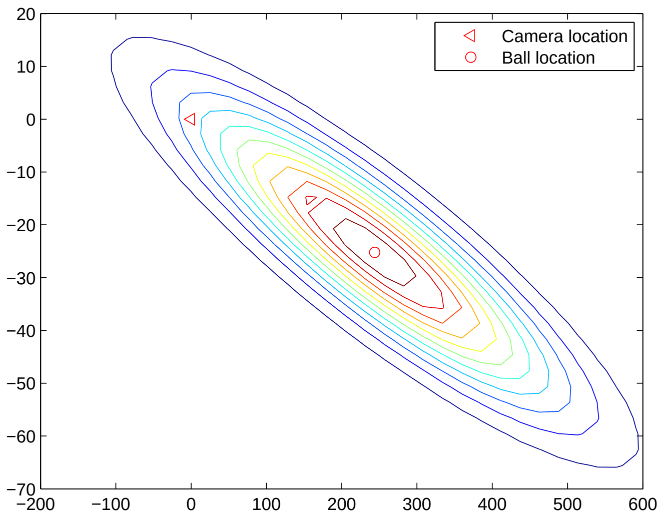

Figure 17: The estimated map location of the ball detected in figure 16.

all alone in the playing field, leaving our goal fully unguarded.

Luckily RAAS did not manage to score during this time and we were the winners of the competition.

References

[1] Heikkilä J., Silvén O. (1997) *A Four-step Camera Calibration Procedure with Implicit Image Correction.* IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97), San Juan, Puerto Rico, p. 1106-1112.

[2] Hartley R., Zisserman A. (2003) *Multiple View Geometry in computer vision, 2nd ed.* Cambridge University Press, Cambridge, United Kingdoms (2006). ISBN: 978-0-521-54081-3.

[3] Carlsson S. (2007) *Geometric Computer in Image Analysis and Visualization.* Numerical Analysis and Computer Science, KTH, Stockholm, Sweden.

[4] Bouguet J., *Camera Calibration Toolbox for Matlab.* http://www.vision.caltech.edu/bouguetj/calib_doc/ (2007-06-07).

[5] Siegwart R., Nourbakhsh I.R. (2004) *Introduction to Mobile Autonomous Robots*, The MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts