# CMPack'03: Robust Vision, Kicking, and Behaviors

Manuela Veloso, Douglas Vail, Sonia Chernova, Scott Lenser,
Juan Fasola, James Bruce

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213
http://www.cs.cmu.edu/~coral

## 1 Introduction

The CMPACK'03 team follows our past teams CMPack'02, CMPack'01, CMPack'00, CMTrio'99, and CMTrio'98 [9, 5, 12, 11]. Our research this year included adding new components that are made possible by the increases in the processing power of the robots. We developed new kicking motions to actuate the ball with greater power. CMPack'03 had a rich set of different kicking motions, so we faced the problem of deciding which kick to use in particular states of the world. To address this problem, we created a new approach to model the effects of different kicks and we incorporated the models into the behaviors so the kick with the highest expected utility will be selected [3]. We refined our world model by adding more robust modeling of the ball and opponents. This information feeds into decisions in the behavior system to allow for more intelligent action selection. We also made improvements to our shared world model, which is built up through communication between the robots [8]. These improvements took the form of better uncertainty estimates for our incoming sensor data. In addition to more accurate models, the team's coordination architecture was revisited [10]. Improvements were made by the addition of a token passing mechanism to enforce mutual exclusion in role assignment. Clearer boundaries between roles allowed us to create a more coherent team. The algorithm strongly relies on low latency in the communication. Unfortunately, at RoboCup'03, we realized that the communication was not consistently reliable. We will continue this research in multi-robot coordination to devise an algorithm more robust to communication errors and latency. To aid in the creation of effective behaviors, we formalized the nested state machine architecture that was used the previous year. Through this formalization we added the necessary infrastructure to trace execution and detect loops in our state machines. The increase in the CPU speed of the robots allowed us to move back to Sensor Resetting Localization (SRL), a particle filter based localization approach [6]. We chose to use this approach because it may be extended to use a richer set of landmarks, such as goal lines and walls, than our previous, Gaussian based approach, which only used field markers. Finally, we used Visual Sonar, a model that returns range and object identity information in a fashion analogous to sonar, for improved obstacle avoidance and localization [7].

In this paper, we focus on briefly presenting our high-level vision processing, the motion kinematics, the modeling of kicking effects, and some behaviors. References are given above for the other contributions or will be forthcoming.

## 2 High Level Vision

The vision module is responsible for converting raw YUV camera images into information about objects that the robot sees and are relevant to our task. This is done in two major stages. The low level vision processes the whole frame and identifies regions of the same symbolic color. We use CMVision [2, 1] for our low level vision processing. The high level vision uses these regions to find objects of interest in the image. The robot uses the 176 by 144 image size exclusively.

The high level vision has access to the original image, the colorized (cmap) image, the run length encoded (RLE) image, and the region data structures from the low level vision in order to make its decisions. The region data structures form the primary input to the high level vision. The high level vision also gets input from the motion module in order to calculate the pose of the camera. The parameters sent from the motion object (via shared memory) are the body height, body angle, head tilt, head pan, and head roll. The high level vision looks for the ball, the goals, the markers, and the robots. For each object, the vision calculates the location of the object, the leftmost and rightmost angles subtended by the object, a confidence in the object, and a confidence that the IR sensor is pointing at the object. The location of the object is 3D relative to a point on the ground directly beneath the base of the robot's neck. These position calculations are based upon a kinematic model of the head which calculates the position and pose of the camera relative to this designated point. The confidence in the object (a real number in $[0, 1]$) indicates how well the object fits the expectations of vision, with 0 and 1 respectively indicating definitely not the object and no basis for saying it is not the object. The IR sensor confidence is similar.

**Ball Detection** The ball is found by scanning through the 10 largest orange regions and returning the one with the highest confidence of being the ball.

*Confidence Calculation* The confidence is based on several components multiplied together plus an additive bonus for having a large number of orange pixels. The following filters are used with Gaussian falloff, and regions on the edge of the image are given more leniency:

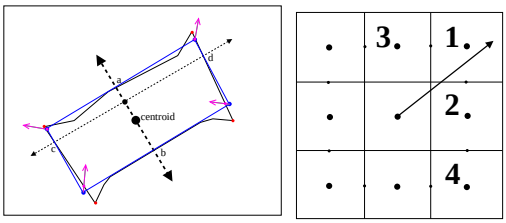
- A binary filter that checks that the ball is tall enough (3 pixels), wide enough (3 pixels), and large enough (7 pixels).
- A real-valued filter that checks that the bounding box of the region is roughly square with as the region gets less square.
- A real-valued filter that checks that the area of the region compared to the area of the bounding box matches that expected from a circle.
- Two filters based on a histogram of the pixel colors of all pixels that are exactly 3 pixels away from the bounding box of the region. The first filter is designed to filter out orange fringe on the edge of the red uniforms for robots in the yellow goal. It is not clear if it is effective. The second filter is designed to make sure that the ball is mostly surrounded by the field, the wall, or other robots and not the yellow goal.
- A new real-valued filter based upon the divergence in angle between the two different location calculation methods, as presented below.

These filters are multiplied together to get the confidence of the ball. The confidence is then increased by the number of pixels divided by 1000 to ensure that we never reject really close balls. The ball is then checked to make sure it is close enough to the ground. The robot rejects all balls that are located more than $5^\circ$ above the robot's head. The highest confidence ball in the image is passed on to the behaviors.

*Location Calculation* The location of the ball relative to the robot is calculated by two different means. Both methods rely on the kinematic model of the head to determine the position of the camera. The first method uses the size of the ball in the image to calculate distance. Geometry is used to solve for the distance to the ball using the position of the camera relative to the plane containing the ball and the estimated distance of the ball. The disadvantage of this method is that it assumes that the ball is always fully visible which obviously isn't always true. The advantage is that is less sensitive to errors in the position of the camera. The second method uses a ray projected through the center of the ball. This ray is intersected with a plane parallel to the ground at the height of the ball to calculate the position of the ball. The advantage of this method is that its accuracy falls off more gracefully with occlusions or balls off the edge of the frame. The disadvantage is that it can be numerically unstable if the assumption that the ball and robot are on the ground is violated. The distance estimate is also slightly noisier than the image size based method but only mostly at large distances. The robot uses the second method whenever possible and the divergence between them is factored into the confidence.

**Goal Detection** The goal detection is based off of finding the corners of the goal and using these corners to estimate the position of the goal. The goal is represented as three separate objects: a left edge, a right edge, and a central object. The central object is intended as a rough estimate of the location of the goal for behaviors that require only a rough idea of the goal location. The left and right goal edges are intended as high quality estimates suitable for fine aiming and localization use.

*Corner Finding* The corners of the goal are found through a multi-step process of successive approximations (see Figure 1 (a)). The largest region of the goal color is found and evaluated to determine if it is the goal or not. The centroid of this region is used as a starting point for finding the corners of the goal. Starting from the centroid, a line is drawn in image space corresponding to $\pm z$ in robot coordinates (the thick dashed line in Figure 1 (a)). Robot coordinates are those where $z$ is up in the world, $x$ is perpendicular to $z$ and aimed in the direction the robot's body is facing, and $y$ is perpendicular to $z$ and $x$. This line is found in image space using projective geometry. The line is traced starting from the centroid looking for the last pixel of goal color in each direction (point a and b). The search is stopped when goal colored pixels haven't been found in a while. Starting from a point midway between the centroid of the goal and point a (small black dot), a line is drawn in the $y$ direction in robot coordinates (thin dashed line). The line is slightly elevated from the center of the goal to help avoid some occlusions. Search is conducted along this line to find the last goal colored pixel in both directions (points c and d). These two searches result in a distance from the centroid to the top, bottom, left, and right of the goal. Completing the rectangle (perfect rectangle in figure) on the image results in 4 approximate corners of the goal.




(a) (b)

**Fig. 1.** (a) Line searches for goal corners; (b) Gradient ascent for goal corners.

The approximate corners are improved by ensuring that each corner is goal colored. A line is drawn from each approximate corner at a $45°$ angle to the two coordinate axis used earlier (magenta arrows on figure). A search is started from each approximate corner to find the last goal colored pixel along this line. If the approximate corner is goal colored, the search proceeds outwards to find the last goal colored pixel (top two points in the case in the figure). If the approximate corner is not goal colored, the search proceeds inwards to find the first goal colored pixel (bottom two points in the case in the figure). We get 4 new approximate corners that are guaranteed to be of goal color.

Next, gradient ascent is applied to each of these approximate corners to improve the corners. This generates the final corners used. Each corner undergoes gradient ascent to maximize the dot product of the corner with the $45°$ vector pointed away from the centroid that corresponds to that corner. The corner is restricted to goal colored pixels at all times during this process. The corner is stepped by one pixel increments that increase the dot product until no further progress is possible. Each step is either a horizontal, vertical, or diagonal move. Pixels are tried in order of increasing angle from the $45°$ vector. Figure 1 (b) shows the fine detail of the search process. The arrow in the figure corresponds to the $45°$ line for the corner. Each box represents a pixel with the central box being the current corner estimate. The center of the pixels are labelled with large black dots. The small black dots represent decision boundaries between the second pixel to try being counter-clockwise or clockwise from the first (this is an approximation of the dot product criterion). The order that pixel are tested for being goal colored is shown by the large numbers in the figure. As soon as a goal colored pixel is found, the corner is moved and the process is restarted. The pixels without numbers in the figure are not tested since they move the corner in the wrong direction. These corners form the basis for the left and right goal edge and improve the estimates for the central goal object.

*Central Goal Object* The central goal object relies on the ratio of the width and height of the goal and the amount of green beneath the goal to estimate the confidence that the object is actually the goal. The angle to the goal is calculated from the centroid of the goal region. The distance to the goal is estimated using the height of the goal. The angular span of the goal on the ground is also computed using either the horizontal scan done or the edges of the goal found as appropriate.

*Goal Edge Objects* The confidence of the goal edges is computed based upon:

- a real-valued filter based on the cosine of the angle between the vector from bottom to top of the goal and the up vector in the image
- a binary filter based upon the width and height of the goal
- a binary filter based upon the ratio of the width of the goal to the height

– a binary filter based upon the size of the green region beneath the goal edge
– a binary filter based upon the pixel size of the white region to the outside of the goal edge (the point used to find the region is chosen such that it is approximately $2/3$ of the way up the white wall

The distance to the goal edge is computed by projecting rays through the corners for the edge and finding the distance at which the rays are separated vertically by the height of the actual goal. This produces a high quality estimate of distance. When the goal edge is viewed almost edge on, the distance may be reported to the back of the goal sometimes, however. If the top corner could not be found reliably because it might have been off the screen, triangulation is performed to the bottom corner. This produces a fairly noisy estimate of the position of the goal edge.

## 3 Motion - Walk Engine and Kicking

Our motion system for CMPACK'03 aims at providing stable and fast locomotion, which requires smooth body and leg trajectories, and to allow smooth, unrestricted transitions between different types of locomotion. We developed our own motion engine so that we would have full parameterization and control, which is not available using default provided robot motions. Our current system is based on our previous designs [5]. Our walking system has two main components: a generic walk engine and a frame-based motion for definition of preset kicks. The overall system uses a state machine that includes states for walking, standing, and one for each type of kick and get-up motion. Requests from the behaviors execute code based on the current state trying to achieve a desired state, maintaining smoothness while attempting to transition quickly.

### 3.1 The Walk Engine

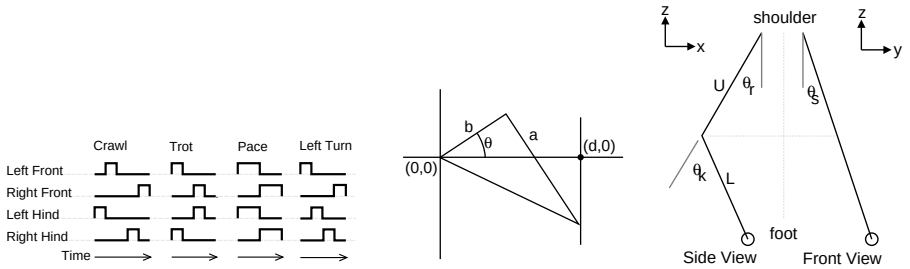
The walk engine can encode crawl, trot, or pace gaits, with optional bouncing or swaying during the walk cycle to increase stability of a dynamic walk. The walk parameters are encoded as a 51-parameter structure. Each leg had 11 parameters; the neutral kinematic position (3D point), lifting and set down velocities (3D vectors), and a lift time and set down time during the walk cycle. The global parameters are the z-height of the body during the walk, the angle of the body (pitch), hop and sway amplitudes, the z-lift bound for front and back legs, and the walk period in milliseconds. In order to avoid compilation overhead during gait development, the walk parameters can be loaded from permanent storage on boot up. We represent each leg by a state machine; it is either down or in the air (see Figure 2 (a)). When it is down, it moves relative to the body to satisfy the kinematic constraint of body motion without slippage on the ground plane. When the leg is in the air, it moves to a calculated positional target. The remaining variables left to be defined are the path of the body, the timing of the leg state transitions, and the air path and target of the legs when in the air. Using this approach smooth parametric transitions can be made in path arcs and walk parameters without very restrictive limitations on those transitions.

**Inverse kinematics** The lowest level in the system is the inverse kinematics engine, which converts foot location targets in world space coordinates to joint angles for the legs. The typical and general solution involves inverting a Jacobian matrix to iteratively

find a solution. This can prove numerically expensive however, and is not necessary for a robot with 3 d.o.f. legs. Here we provide a closed form solution that allows arbitrary translations at each limb, and our solution only requires that the rotator and shoulder joint axes cross. Those familiar with closed-form inverse kinematics may wish to skip to the next section.

The first part of our solution is to solve a simple problem and then express the joint angle calculations in terms of that problem. This primitive problem is finding the roots with respect to $\theta$ of: $F(a, b, d) = \text{Root}_\theta(a \cdot \sin(\theta) + b \cdot \cos(\theta) - d)$.

Note that since the equation can be normalized, there exists exactly two solutions for $\theta \in [0, 2\pi]$. Figure 2 (b) represents this problem graphically. We can then solve for $\theta$ with trigonometry: $\theta = \tan^{-1}\left(\frac{a}{b}\right) \pm \cos^{-1}\left(\frac{d}{\sqrt{a^2+b^2}}\right)$.




(a) (b) (c)

**Fig. 2.** (a) (a) A timing diagram of various walking gaits; The function is zero when the foot is on the ground and nonzero when it is in the air. (b) A graphical representation of $a \cdot \sin(\theta) + b \cdot \cos(\theta) = d$; (c) Model and coordinate frame for leg kinematics.

Let $p$ be the target from the shoulder origin, $u$ be the translation of the upper limb (from the shoulder to the knee), and $l$ be the translation of the lower limb (from the knee to the foot). Define $R_k(\theta)$ to be the right handed rotation matrix for dimension $k \in \{x, y, z\}$ [4]. We can now solve for the knee, shoulder, and rotator angles $(\theta_k, \theta_s, \theta_r)$ for the coordinate system and leg model shown in Figure 2 (c). We can first notice that the distance from the shoulder to the foot determines the knee angle, using the relationship of the distance and the rotation of the lower limb relative to the upper limb ($\|p\|^2 = \|u + R_y(\theta_k) \cdot l\|^2$). The rotation equation can be expanded and simplified into the form of $F$, with the resulting transformation as:

$$a = 2 \cdot (u_x \cdot l_z - u_z \cdot l_x) \qquad b = 2 \cdot (u_x \cdot l_x - u_z \cdot l_z)$$
$$d = \|p\|^2 - \|u\|^2 - \|l\|^2 - 2 \cdot u_y \cdot l_y \qquad \theta_k = -F(a, b, d)$$

After the knee angle is fixed, the leg is essentially a single rigid body, and position including the knee angle can be calculated as $q$. We can then rotate $q$ into the plane of $p_y$ using the shoulder joint, $q = u + R_y(\theta_k) \cdot l$, and simplify the problem as: $a = -q_z; b = q_y; d = p_y; \theta_s = F(a, b, d)$.

Now we can calculate a new leg position $r$ that differs from $p$ only in rotator angle. Using a similar approach, we can determine the required angle to rotate $r$ into the plane of $p_x$: $r = R_x(\theta_s) \cdot q; a = -r_z; b = r_x; d = p_z; \theta_r = -F(a, b, d)$.

We now have the joint angles $(\theta_k, \theta_s, \theta_r)$. The multiple problem solutions can be pruned by the reachable angles at each joint, and its proximity to the current joint angle.

The result is a simple high precision non-iterative method for calculating joint angles. This simple method allows high performance on relatively modest machines. A straightforward implementation of the above calculations can calculate 32400 sets of leg angles per second on a 200MHz machine.

### 3.2 Modeling the Effects of Kicking Motions

In addition to the walking motion, we continue to study kicking motions. For CMPack'03, we introduced a new "swing kick" that showed to be very effective. Figure 3 illustrates several of the frames of this new swing kick.

**Fig. 3.** Frame-based kicking motion: the "swing" kick

Given that CMPack now has several different kicks, we address the problem of modeling the effects of different kicks. We then incorporate these models in the behaviors to select the most promising kick in a given state of the world. We modeled the trajectory angle and distance reached by the ball in the forward kick and head kicks. The swing kick was effective developed closer to RoboCup'03 and was not used in the experimental modeling.

**Trajectory Angle** The angle of the ball's trajectory after a kick is an important characteristic of all kicking motions. No external cameras or sensors are used in this experiment in order to make it applicable in any environment. The robot's vision module provides information about the ball's location for every vision frame where the ball is in the visual range of the camera. The ball position history records the position of the ball relative to the robot's frame of reference over the past 25 frames, or approximately 1 second. The ball history reports the ball position as unknown for frames where the ball is outside the camera field of view. Ball location estimates from the world model are not used in order to minimize error. Table 1 shows the algorithm we developed for approximating the angle of the ball's trajectory.

**Algorithm 3.1:** TRACKANGLE()

$timeOfKick \leftarrow 0$

**do**
- TRACKBALLWITHHEAD()
- **if** $BallWithinKickingRange =$ **true**
  - **then**
    - KICK()
    - $timeOfKick \leftarrow currentTime$
- **if** $currentTime - timeOfKick > t_{delay}$
  - **then**
    - $angle \leftarrow$ CALCANGLEUSINGBALLHISTORY()
    - **output** $(angle)$

**Table 1.** Computation of the angle of ball's trajectory from an input of the estimated ball distance and angle values from vision.

The robot performs all analysis autonomously, and requires human assistance only in placing the ball in front of the robot for the next kick. The robot does not approach the ball on its own in order to assure the consistency of the trials.

The value of the variable $t_{delay}$ is a key factor in the success of this procedure. Since the vision ball history is continuously keeping track of the ball's location, it is important to analyze the data in the history at the right time - before the ball is so far away that the values become unreliable, but after enough time has passed to record at least a second of the ball's trajectory. The standard value of $t_{delay}$ is $t_{kick} + 1.0sec$, where $t_{kick}$ is the duration of the kick. Slightly larger $t_{delay}$ values are used for kicks with a lot of head movement that may cause a delay in tracking the ball after the completion of the kick.

A linear regression algorithm is applied to the points in the history buffer to calculate the angle of the ball's trajectory. The slope of the line is used to approximate the angle of the trajectory. In order to assure accuracy, we require that a minimum of 20 out of the last 25 vision frames contain information about the ball. Trials with fewer data frames are often a sign of an error. This can occur if the $t_{delay}$ value is too small, if the kick fails or if the robot for some reason fails to locate and track the ball. Eliminating these trials ensures that inaccurate data is not introduced into the analysis and makes this learning process more efficient, requiring little human intervention.

**Distance** The second attribute important in understanding the effects of different kick motions is the distance the ball travels or the strength of the kick. The robot is unable to track the entire trajectory of the ball because the ball travels out of the robot's visual range for many of the powerful kicks. Instead, the robot calculates the final position where the ball stops relative to the original position of the robot before the kick. Table 2 shows the algorithm we developed for accurately calculating the displacement of the ball after a kick.

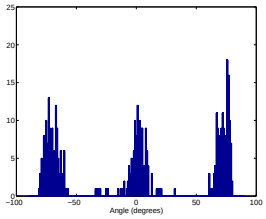
**Algorithm 3.2:** TRACKDISTANCE()

```
while 1
do { APPROACHBALL()
     KICKBALL()
     STANDANDLOCALIZE()
     initialBallPosition ← currentRobotPosition
     FINDBALL()
     APPROACHBALL()
     if ballDistance < 50cm
       then { STANDANDLOCALIZE()
              finalBallPosition ← currentBallPosition
              ballDispVector ← finalBallPosition − initialBallPosition
              output (ballDispVector)
```

**Table 2.** Computation of a vector representing the ball's displacement relative to the location of the kick, given the estimates of the ball and robot locations from vision.

We selected two specific attributes to model the effects of the kicking motions, the angle of the ball's trajectory and the distance reached by the ball after the kick. Analysis of both kick attributes was performed using only the robot platform with no additional sensors and minimal human intervention. We used the acquired data to build a model

that represents each kick in terms of its effects on the ball. Figure 4 (a) shows the angle variation for three kicks and Figure 4 (b) shows the average angle and distance reached by the ball for the differnent kicks. We did not gather experimental results for the hard head kick but we approximate the angle values to the ones of the normal head kick given their similarity with respect to angle.




(a)

| Kick | Angle (degrees) | Distance (m) |
|---|---|---|
| Forward | 2.1 | 2.2 |
| Normal Head Right | 72.6 | 1.64 |
| Normal Head Left | -70.4 | 1.64 |
| Hard Head Right | 72.6 | 3.07 |
| Hard Head Left | -70.4 | 3.07 |

(b)

**Fig. 4.** (a) Ball angle distribution for a left head, forward, and right head kicks; (b) Average values for ball angle and ball distance for five kicks.

## 4 Behaviors

The CMPACK'03 behavior system draws heavily from our 2002 team. Each behavior is represented as a finite state machine. State machines may themselves contain encapsulated state machines. Transitions between states are represented programmatically and may be arbitrarily complex. By formalizing the allowed state transitions and recording them as they occur, we are able to trace the execution of our behaviors, which aids debugging, and detect loops in our state transitions. We continue to develop single-robot and multi-robot behaviors. We briefly discuss our multi-robot coordination approach for CMPACK'03 as an extension of the one of our 2002 team and then introduce our new developed method to select kicking motions based on the models of their effects.

### 4.1 Multi-robot coordination

The robots playing offense are assigned three, separate roles. These roles are a *primary attacker*, which approaches the ball and attempts to move it upfield; an *offensive supporter*, which moves up the field from the primary attacker and positions itself to recover the ball if the primary attacker misses its shot on goal; and a *defensive supporter*, which positions itself down the field from the primary attacker to recover the ball if the other team captures it [10].

Our 2002 algorithm was rather effective but it did not guarantee a unique assignment of roles to the different robots. We developed a *token-based* method to uniquely assign roles. The role of primary attacker is assigned by passing a token between the robots to ensure mutual exclusion, as two robots approaching the ball at the same time tend to become entangled. The primary attacker divides the other two robots between the two supporting roles, namely the offensive supporter and the defensive supporter, when it first receives the token. Robots in supporting roles request the token from the primary attacker when they detect that they are better positioned than it to approach the

ball. The primary attacker may choose to pass the token or refuse to hand it off. Once the robots have been assigned roles, robots in supporting roles use a potential field to position themselves, as in 2002. This field encodes both obstacle information and heuristic information, such as robots should avoid blocking shots on their opponents' goal.

Our implementation of the token-based role assignment for CMPACK'03 showed to be strongly dependent of a reliable communication channel that would ensure that the token was successfully passed among robots. Unfortunately, at RoboCup'03, we realized that network problems greatly affected the performance of our implementation, and the robots, which led into a poor role assignment under communication problems. Improving multi-robot coordination is part of our on-going research agenda.

### 4.2 Selection of kicks

In CMPACK'03 , we have improved the selection of different kicks based on the models that we empirically collected of the effects of kicking motions (see section 3.2).

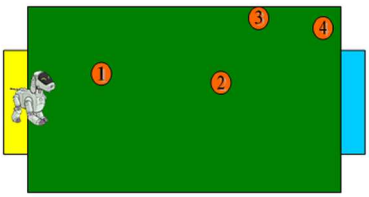
Once we have modeled the kick motions, we developed an algorithm for selecting an appropriate kick to use. Previous implementations of kick selection algorithms have not relied on prediction or modeling of actions. Decisions were usually hard coded based on the programmer's intuition about different effects of the kicks. Although a human can roughly estimate the average distance that the ball is expected to travel after seeing a series of repeated trials, this data is not very accurate. Estimating the angle of the trajectory can be especially difficult. We briefly present a new kick selection algorithm that takes advantage of the acquired data and selects actions in a more robust manner.

**The kick selection algorithm** The data gathered in the modeling stage can be organized into a motion library where each kicking motion is classified by its effects on the ball, mainly in terms of the mean angle and distance values. The library provides an easy reference and allows the user to easily add or remove motions.

The new kick selection algorithm was designed to take advantage of this new organization in the following manner. The localization system provides information about the robot's position on the field and the relative location of the goal, which allows the robot to calculate the desired trajectory of the ball. The motion library is then referenced to select the most appropriate kick whose effects match the desired trajectory. If no kick motion provides a close match, then a series of behaviors can be sequenced together to achieve the desired effect. For example the robot may choose to turn or dribble the ball in order to achieve a better scoring position. We performed several experiments to test the effectiveness of the selection algorithm.

*Experimental results* We report on an experiment to test the robot's ability to score a goal on an empty field without any other robots present. Testing single robot performance is important because it allows us to fine tune individual performance and analyze the robot's strategy without interference from other robots. Additionally, since the robot's vision of other robots is often inaccurate, this is one of the best metrics of performance. We also tested the ability of the robot to score a goal when playing against a single opponent. The results are not to be credited solely to the kick selection algorithm, as playing in the presence of other robots clearly offers another level of challenges.

Figure 5 (a) shows the four ball positions for the single robot scoring experiment. The robot is placed on the goal line of the defending goal. The location of the ball is initially unknown to the robot and the ball may or may not be within visible range. The robot's performance is evaluated by recording the time it takes to score. The specific location of the four points for the position of the ball were selected to test various cases.




(a)

| Point | CMPack'02 | Modeling |
|---|---|---|
| Point1 | 56.7 | 39.8 |
| Point2 | 42.5 | 27.2 |
| Point3 | 76.5 | 60.0 |
| Point4 | 55.0 | 52.0 |
| Total | 57.8 | 44.8 |

(b)

**Fig. 5.** (a) Points 1-4 of the ball positions for robot scoring experiment with model-based kick slection. (b) Performance comparison of a CMPack'02 attacker and a CMPACK'03 attacker using the kick modeling selection algorithms. Values represent mean time to score in seconds, averaged over 13 trials per point.

- Point 1 tests the robot's ability to score from a large distance. The ball is located close to the initial position and in many cases the robot will require more than one kick to score.
- Point 2 tests the accuracy of the kicks. The point is located at a short distance and a direct angle to the goal. The robot is expected to score directly.
- Point 3 tests the robot's ability to score from the side when the ball is near a wall. The robot may or may not see the ball at this distance and often requires more than one kick to score.
- Point 4 tests situations when the ball is located in the corner at a wide angle to the goal. The robot does not see the ball at this distance. Scoring with one kick is very unlikely from this position.

The table in Figure 5 (b) summarizes the results of the experiment. A total of 104 trials were tested, with 13 trials for each ball position. Results show that the modeling and prediction algorithm performed better for every point, with an overall improvement of 13 seconds. The statistical significance of the results was confirmed using the Wilcoxon Signed Rank Test.

The new kicking selection algorithm showed to be effective in game situations in which the robots use the model of the kicks for selecting the most promising kick. Our modeling research is ongoing and we envision experimentally gathering and using new models of the effects of the robot's actions. We are especially interested in devising algorithms to improve the action selection of the robots in the presence of complex multi-robot situations.

## 5 Conclusion

With CMPACK'03 , we pursue our research on teams of intelligent robots. We continue to note that a team of robots needs to be built of skilled individual robots that can coordinate as a team in the presence of opponents. We research on single robot perception and on multi-robot team coordination. In this paper, we focused on presenting our high-level vision for robust object recognition, our motion and empirical modeling of kicking motions, and how the behaviors use the kicking models.

Our work for CMPack'04 will be strongly focused on teamwork and on learning additional models of the robot performance so the robot can automatically adapt its motion and choices to different environments.

## Acknowledgements

This research has been partly supported by Grant No. DABT63-99-1-0013, by generous support from Sony, Inc., and by a National Physical Science Consortium Fellowship with stipend support from HRL Laboratories. The content of this publication does not necessarily reflect the position of the funding agencies and no official endorsement should be inferred.

## References

1. J. Bruce, T. Balch, and M. Veloso. CMVision, www.cs.cmu.edu/˜jbruce/cmvision/.
2. J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of IROS-2000*, 2000.
3. S. Chernova. Adaptive motion and behavior for four-legged robots. Technical report, Computer Science Department, Carnegie Mellon University, 2003. Undergraduate Senior Thesis.
4. J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics, Principles and Practice*. Addison-Wesley, Reading, Massachusetts, second edition, 1990.
5. S. Lenser, J. Bruce, and M. Veloso. CMPack: A complete software system for autonomous legged soccer robots. In *Autonomous Agents*, 2001.
6. S. Lenser and M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Proceedings of ICRA-2000*, 2000.
7. Scott Lenser and Manuela Veloso. Visual sonar: Fast obstacle avoidance using monocular vision. In *Proceedings of IROS'03*, 2003.
8. Maayan Roth, Douglas Vail, and Manuela Veloso. A world model for multi-robot teams with communication. In *Proceedings of IROS'03*, 2003.
9. William Uther, Scott Lenser, James Bruce, Martin Hock, and Manuela Veloso. CM-Pack'01: Fast legged robot walking, robust localization, and team behaviors. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002.
10. Douglas Vail and Manuela Veloso. Dynamic multi-robot coordination. In Alan C. Schultz, Lynne E. Parker, and Frank E. Schneider, editors, *Multi-Robot Systems: From Swarms to Intelligent Automata*, volume 2, pages 87–98, 2003.
11. M. Veloso and W. Uther. The CMTrio-98 Sony legged robot team. In M. Asada and H. Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*, pages 491–497. Springer, 1999.
12. M. Veloso, E. Winner, S. Lenser, J. Bruce, and T. Balch. Vision-servoed localization and behavior-based planning for an autonomous quadruped legged robot. In *Proceeding of the 2000 International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, 2000.