CMPack-02: CMU's Legged Robot Soccer Team

Manuela Veloso, Scott Lenser, Douglas Vail,
Maayan Roth, Ashley Stroupe and Sonia Chernova
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
{mmv,slenser,dvail2,mroth,ashley}@cs.cmu.edu

October 2, 2002

1 Vision

The vision module is responsible for converting raw YUV camera images into information about objects that the robot sees. 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.

1.1 Low Level Vision

The low level vision is unchanged from last year. The low level vision converts raw YUV camera images into 4-connected regions of the same symbolic color. It does this in several stages. The image is first segmented according to a threshold table. This converts each pixel from a YUV value to a symbolic color like orange. The color segmented image is then run length encoded to reduce the amount of memory consumed by the image and speed further processing. The run length encoded (RLE) image is then analyzed to find 4-connected regions of the same color. Some noise is then removed by merging nearby regions of the same color.

1.1.1 Color Segmentation

The robot uses a 3D lookup table for color segmenting the image. The lookup table is indexed by the high bits of the raw Y, U, and V values of the pixel. The number of bits used from each component is configurable. We used 4 bits of Y and 6 bits each of U and V at the competition for a total of 16 bits. This yields a lookup table of size 65536 bytes. Each entry of the lookup table stores the index number for the symbolic color to assign to the pixel or 0 if the pixel is background. The thresholds are learned from example images as described in section 1.3. The color segmentation process uses the threshold table on each pixel of the image to produce a color map (cmap) for the image (see Figure 1 for the effect of this process). This cmap image is available to the high level vision to check the symbolic color of individual pixels. We treat the field green and the marker green as the same color when segmenting.

1.1.2 Run Length Encoding

The cmap image is then run length encoded. This consists of replacing horizontal runs of the same color with a run. A run stores the color it is associated with, the x,y location at which it starts, and the number of pixels included in the run. Run length encoding reduces the size of the image in memory. This reduces the amount of time needed by the processor to scan over the entire image while performing some function. Only non-background pixels are encoded into runs, which is why the x,y location is needed. We found that this increases the speed at which the robot can process data. An exception to this is the last background run on each scan line is stored. This allows for more regular code

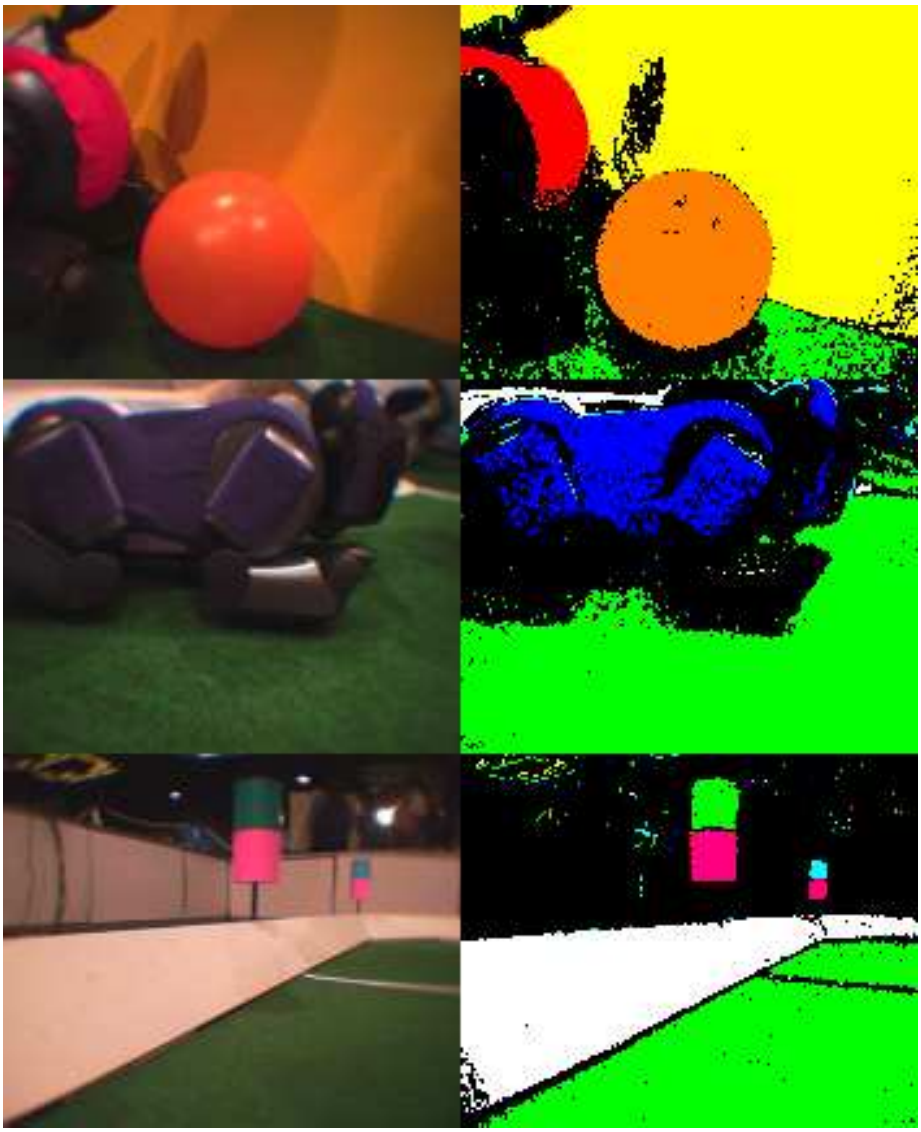

Figure 1: The result of classifying images from the robot. Source images on the left, classified images on the right.

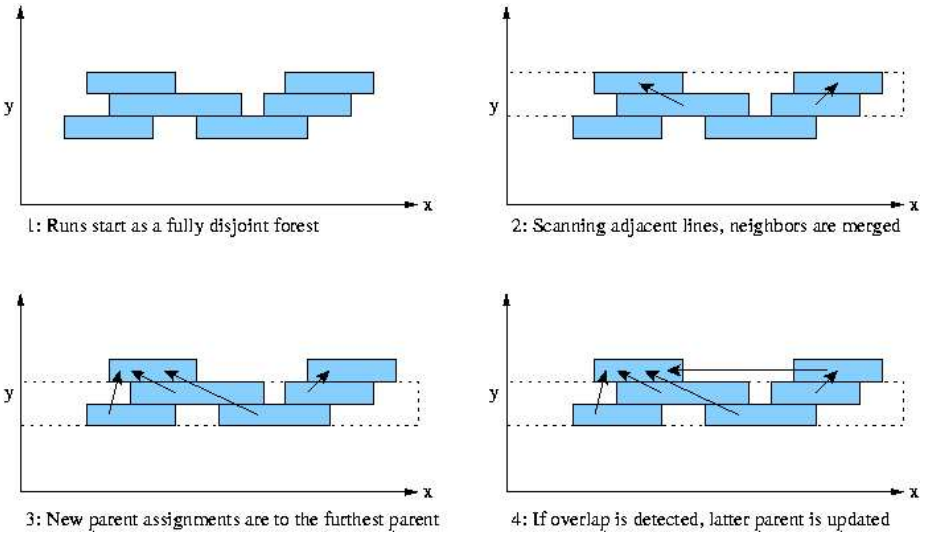

Figure 2: Region forming process on RLE image

since each line has at least one run on it. The output of this stage is a run length encoded (RLE) image.

1.1.3 Finding Connected Components

The merging method employs a tree-based *union find* with path compression. This offers performance that is not only good in practice but also provides a hard algorithmic bound that is for all practical purposes linear [8]. The merging is performed in place on the classified RLE image. This is because each run contains a field with all the necessary information; an identifier indicating a run's parent element (the upper leftmost member of the region). Initially, each run labels itself as its parent, resulting in a completely disjoint forest. The merging procedure scans adjacent rows and merges runs which are of the same color class and overlap under four-connectedness. This results in a disjoint forest where the each run's parent pointer points upward toward the region's global parent. Thus a second pass is needed to compress all of the paths so that each run is labeled with its the actual parent. Now each set of runs pointing to a single parent uniquely identifies a connected region. The process is illustrated in Figure 2). The regions are then extracted by finding all of the runs that have the same parent. The runs for each region are linked together into a linked list to facilitate further processing. Summary statistics are gathered for all of the regions including: bounding box, centroid, and area. The regions are then separated based on color into linked lists and sorted in order of decreasing area. The linked lists of regions form the output of this last stage and the main input to the high level vision.

1.1.4 Region Merging

The purpose of region merging is to combine several nearby smaller regions into a single larger region in order to remove noise. This is done by considering merging all pairs of regions of the same color. The criterion for deciding to merge the regions is based off of the resulting density of the combined region. Each color has a density threshold associated with it. Two regions are combined if the combined pixel area of the regions divided by the pixel area covered by the new bounding box is above the density threshold.

1.2 High Level Vision

The job of high level vision is to find objects of interest in the camera image and estimate the position of these objects relative to the robot. 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 a 3D location 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 is a number between 0 and 1 which indicates how well the object fits the expectations of the vision system, 0 indicating definitely not the object and 1 indicating no basis for saying it is not the object. The IR sensor confidence is similar. The following sections explain how the various objects in the image are located.

1.2.1 Ball Detection

Ball detection is mostly the same as last year. 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 a bunch of components multiplied together plus an additive bonus for having a large number of orange pixels. The following filters are used:

- A binary filter which checks that the ball is tall enough (3 pixels), wide enough (3 pixels), and large enough (7 pixels).

- A real-valued filter which checks that the bounding box of the region is roughly square with Gaussian falloff as the region gets less square. Regions on the edge of the image are given more lenience.
- A real-valued filter which checks that the area of the region compared to the area of the bounding box matches that expected from a circle. Again, Gaussian falloff is used. Regions on the edge of the image are given more lenience.
- Two filters based on a histogram of the pixel colors of all pixels which 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.
- **New** A real-valued filter based upon the divergence in angle between the two different location calculation methods (see 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° 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.

1.2.2 Marker Detection

Marker detection is unchanged from last year. Markers are detected by considering all pairs of pink and yellow/green/cyan regions out of the largest 10

regions of each color. The most confident marker readings are passed along to the behaviors/localization.

Confidence Calculation The confidence in a marker reading is based upon the following real-valued filters:

- The cosine of the angular spread between the projection of the ray through the center of the two colored regions projected onto the ground plane. This filter is setup to fall to 0 at an angular separation of 15°.
- The relative size of the two colored regions (we expect them to be the same size).
- The average size of the two colored regions relative to the square of the distance between them (we expect these values to be equal).

Location Calculation Two rays are projected through the centers of the colored regions. These rays are made coplanar by projecting both rays onto a vertical plane containing a ray formed by averaging the two rays. The location of the center of the top of the marker is constrained to be 10cm above the center of the bottom of the marker. The location of the top of the marker is also constrained to be directly vertically above the bottom of the marker. The resulting system of equations is solved for the location of the marker.

1.2.3 Goal Detection

The goal detection system was completely revamped this year. 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 3). 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 (big black dot), a line is drawn in image space corresponding to $\pm z$ in robot coordinates (the thick dashed line in Figure 3). 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

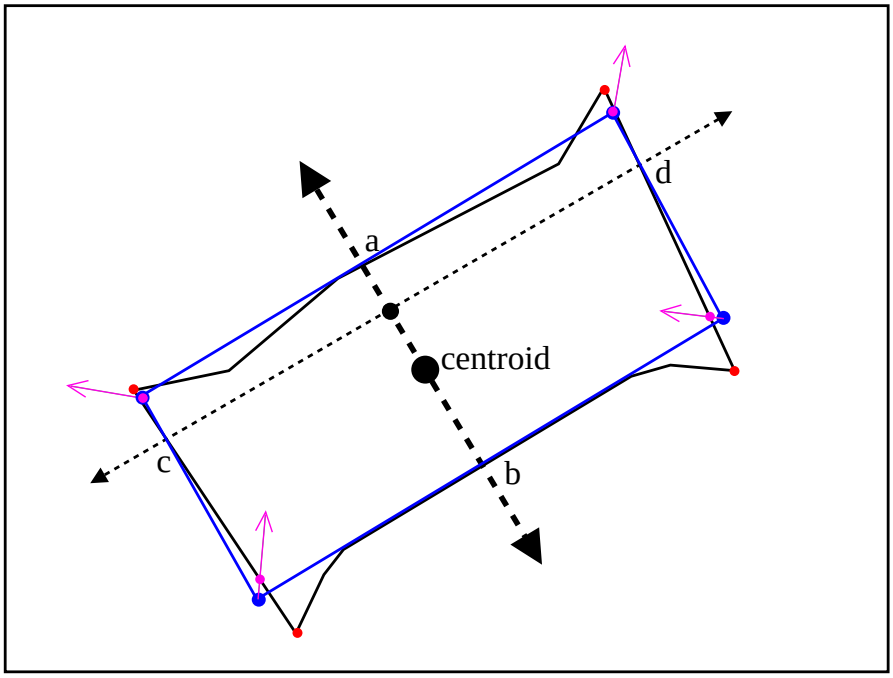

Figure 3: Line searches for goal corners.

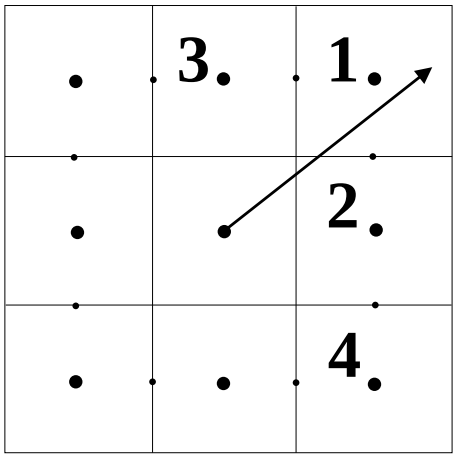

Figure 4: Gradient ascent for goal corners.

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 (blue rectangle in figure) on the image results in 4 approximate corners of the goal (blue dots).

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). This results in 4 new approximate corners which are guaranteed to correspond to goal colored pixels (magenta dots).

Next, gradient ascent is applied to each of these approximate corners to improve the corners. This generates the final corners used (red dots in figure). 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 4 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. The corners found by this process form the basis for the left and right goal edge object 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 the following features:

- 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 pixel 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.

1.2.4 Robot Detection

Robot detection was unchanged for this year. By the time robot detection starts, the low level vision has already created regions corresponding to one or more of the colored patches on the robot. The first thing the robot detection tries to do is complete this process merging together patches from the same robot together while keeping regions of different robots (even if of the same color) separate. This is done by looking at pairs of regions of the same color and considering whether to merge them or not. All region pairs which share at least one common scanline are considered. The basic idea is to keep regions separate when there is a lot of green between them or when they are far apart.

Every scanline that the regions overlap on is processed to generate some summary statistics for the way the two regions are related. The simpler of the statistics simply keeps track of the average amount of horizontal distance between the edges of the two regions. This is eventually compared to the width of the bounding boxes of the regions to decide whether to merge the regions. The other statistic keeps track of the amount of separation caused by colors not likely to be seen on robots (primarily green). We want to be insensitive to a few pixels of noise but still responsive to a thin band of green from the field between two robots. For each scan line we keep track of the size of the largest green region separating the two candidate regions. If the average of this size is too large, we reject merging the regions.

Distance to the robot is guessed based on the number of pixels in the image. Location is calculated based off of this distance.

1.3 Threshold Learning

Threshold learning was unchanged for this year. The goal of threshold learning is to take a set of hand labelled images and produce a threshold map. The threshold map should generalize from the data as much as possible while correctly classifying as many pixels as possible. The threshold map is a 3D table of color indices indexed by the high bits of YUV pixels. The basic idea is to spread each pixel of data out across the entire table giving it more weight closer to its actual value and much less weight further away. In order to do this efficiently, we use a mathematically simple form of falloff for each pixel. The weight of each pixel falls off exponentially with Manhattan distance from the actual data point. Weights are calculated for each color separately. The final step is to determine the color to make each cell of the threshold map. This is simply done by looking for the color with the most weight in each cell. This color is assigned to the cell if its proportion of the weight is higher than a user set confidence threshold for the color. Any cell not meeting this criterion or with a preponderance of background color is labelled background. This representation and learning method allows us to handle colors with arbitrary distributions including concave and disjoint distributions.

1.4 Visual Sonar

A new module called visual sonar was introduced this year. Visual sonar builds a radial map of the distances to objects in the vicinity of the robot. The name visual sonar comes from the similarity of the resulting model to the information provided by sonar. There are two parts to building the radial map: 1) detecting nearby objects from camera images and 2) propagating belief about nearby objects as the robot moves and senses.

1.4.1 Visual Processing of Proximate Objects

Visual sonar is based upon tracing scan lines through the camera image to look for colors associated with particular objects. Scan lines from the base of the robot's neck are created at regular intervals (we used 5°). The objects found along these scan lines will form a radial map of the objects around the robot with more resolution devoted to nearby objects by the polar nature of the map. A drawing of the scan lines used for a spacing of 15^circ is shown in Figure 5. These scan lines are projected into image coordinates (scan lines not on the image are thrown out) resulting in scan lines on the image (see Figure 6, purple lines are the scan lines used). Each scan line is traced over and converted into a run length encoded form. This run length encoded form simplifies further processing and allows convenient access to the entire ray of the scan line.

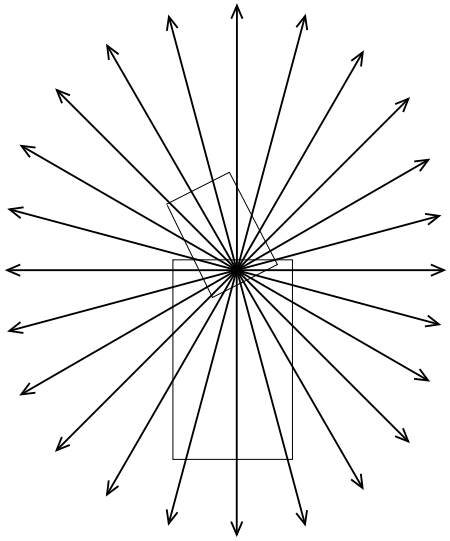

Figure 5: Radial scan lines on ground used in visual sonar. These scan lines correspond to a spacing of 15°.

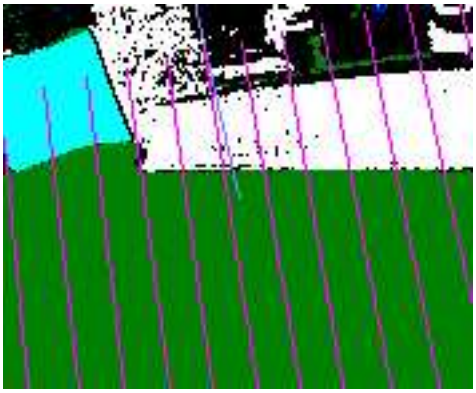

Figure 6: Radial scan lines on image used in visual sonar. The purple lines are the radial scan lines. This image was generated using scan lines every 5°.

Each run in the run length encoded form has the following structure:

Field
Start Point
End Point
Color
Length in Pixels Traversed

Once the runs have been found, each scan line is processed separately to find the objects that lie along it. The following objects are recognized by the scan line processing: visible start (the first location that is visible along this scan line), visible end (the last location that is visible along this scan line), clear start (the first location that is clear field), clear end (the end of the first section of clear field), wall, stripe, robot (any obstacle that can't be identified), red robot, blue robot, ball, cyan goal, and yellow goal. Most of these objects are simply recognized by their color. Unidentifiable obstacles are recognized as areas which are not green. Stripes are distinguished from walls using the width of the white section and the amount of green behind the white section. The distance to each object is calculated using triangulation and the known position of the camera assuming that the object is on the ground plane. This generates correct distance estimates for everything except robot bodies. Robot feet still generate the correct distance. Because the position of the camera is not known with much certainty, the distance estimates obtained are somewhat noisy, especially at long distances. The accuracy at short distances is easily sufficient for obstacle avoidance, however.

1.4.2 Belief Propagation of Proximate Objects

The radial model contains the best estimate the robot has of its immediate vicinity. The radial model using the visual sonar information from vision and the motion commands executes to create a radial map of the robots surroundings. The map is stored as a two-dimensional array indexed by object type and angle relative to the robot. The radial model uses the same number of angles as the vision does. Each entry in the map array stores the distance to the object, the time last seen, and the location of the object in egocentric Cartesian coordinates. It is important that the Cartesian location is stored because otherwise the exact angle that the object was observed at will be lost due to discretization effects. The model is updated for new vision input by erasing all objects that should have been seen (based upon the visible start/end information from vision) and replacing that part of the map with the objects that were actually seen. Objects are update for motion by applying the inverse of the motion command to every object in the map (the stored points are updated). Each object is then inserted into a new map based upon its new angle relative to the robot. Objects of the same type that were adjacent in the old map are assumed to be connected parts of the same object. In the case that these neighboring object readings are no longer adjacent in the new map, additional points are created in the new map for each angle missing by drawing a straight line between the original two readings.

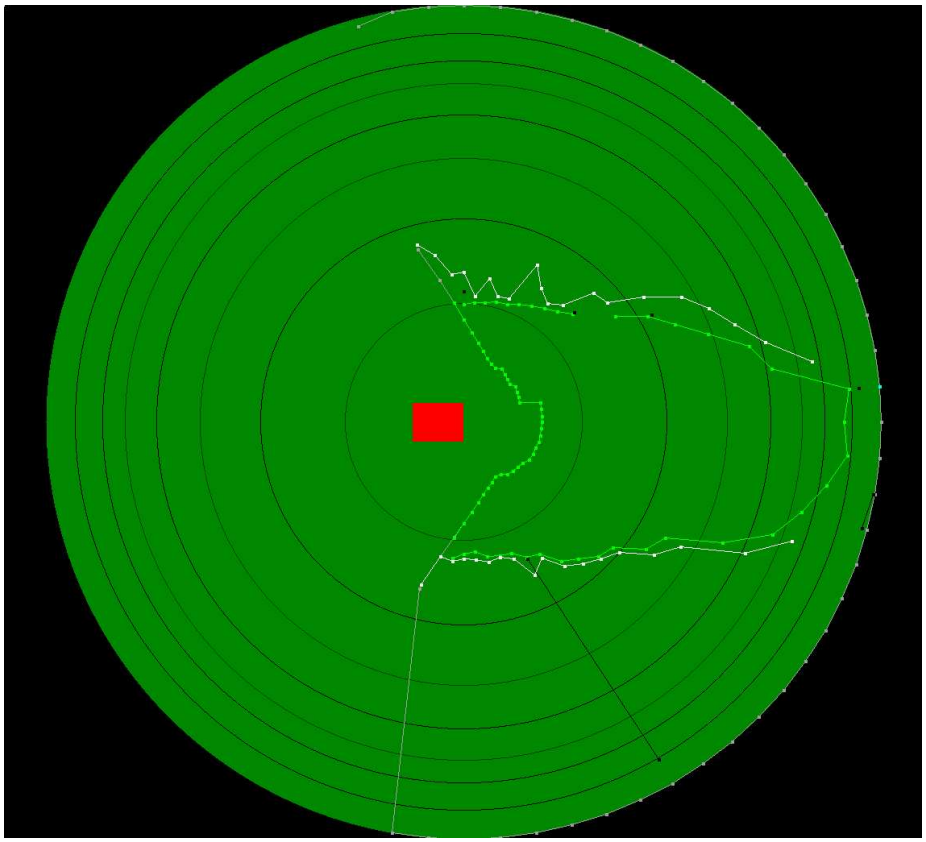

Figure 7: A radial model generated by a robot walking between two white walls.

The intersection of this line with the missing angle is used to fill in the missing data. If this is not done, the model quickly develops holes in objects such as walls that should be continuous. An example of the radial model obtained is shown in Figure 7. This model was generated by a robot walking between two white walls. The red box represents the robot which is moving to the right. The white lines are the walls seen by the robot. The black circles are spaced 1m apart and the dark grey circles are spaced .5m from the black circles. The green lines show the field that the robot can see.

2 Localization

For CM-Pack '02, we developed a new, simple probability-based method for localization: Probabilistic Constraint-Based Landmark Localization (PCBL). The theory is similar to previous Bayesian localization techniques used by our CMU legged robot team from 2001 and that used by the CMU middle-size league in 2001 [9, 7]. We keep a Gaussian probability distribution over possible current locations, P_{loc}, and another Gaussian probability distribution over possible current headings, P_θ. The distribution over position is a two-dimensional non-circular Gaussian.

Updates are performed as in other Bayesian techniques. When the robot sees a marker, it updates P_{loc} using Bayes' rule: $P_{loc|sensor} \propto P_{loc} \times P_{sensor|loc}$. Heading is updated similarly. When the robot moves, it translates P_{loc}, rotates P_θ and then it smooths it to account for the noise in the odometry.

Approximation of the non-Gaussian distribution with Gaussian distributions is typically acceptable. Results converge to the correct location, even when robots are transported to different field locations. The approximation to pose uncertainty is accurate enough to determine when the robot is well localized and when it is becoming lost.

2.1 Pose updates

Two types of landmarks were used for the sensor updates: field markers (6) and goal edges (4). When the robot sees a landmark the vision returns a range estimate for that marker and a bearing to the landmark relative to the robot's current heading. To perform the Bayesian update, the probability distribution of this sensor reading being generated from each location on the field must be known. The true distribution relating X, Y, and θ to range and bearing is highly non-Gaussian, which would lead to highly complex updates.

To simplify the update, several assumptions and approximations are made. First, the distribution over range and bearing are approximated as one-dimensional Gaussians that are independent. The parameters of these distributions were determined experimentally. The updates for range and bearing are also treated independently. When multiple markers are simultaneously seen, independent updates are performed for each marker sequentially. This simplification is made possible by using Gaussian distributions, which can be combined through symmetric operations.

The range measurement places the robot on a circle around the marker, whose radius is probabilistic. This circle is a constraint on position. The area of high likelihood on the circle is the area closest to the robot's previous location. A Gaussian distribution can be aligned tangent to the circle in the part of the arc closest to the previous position estimate. This two-dimensional Gaussian can then be easily incorporated using a Bayesian update.

Similarly, the bearing measurement places the robot along a line that passes through the marker. This line is an additional constraint on position. The thickness of the line grows with distance from the marker due to the uncertainty

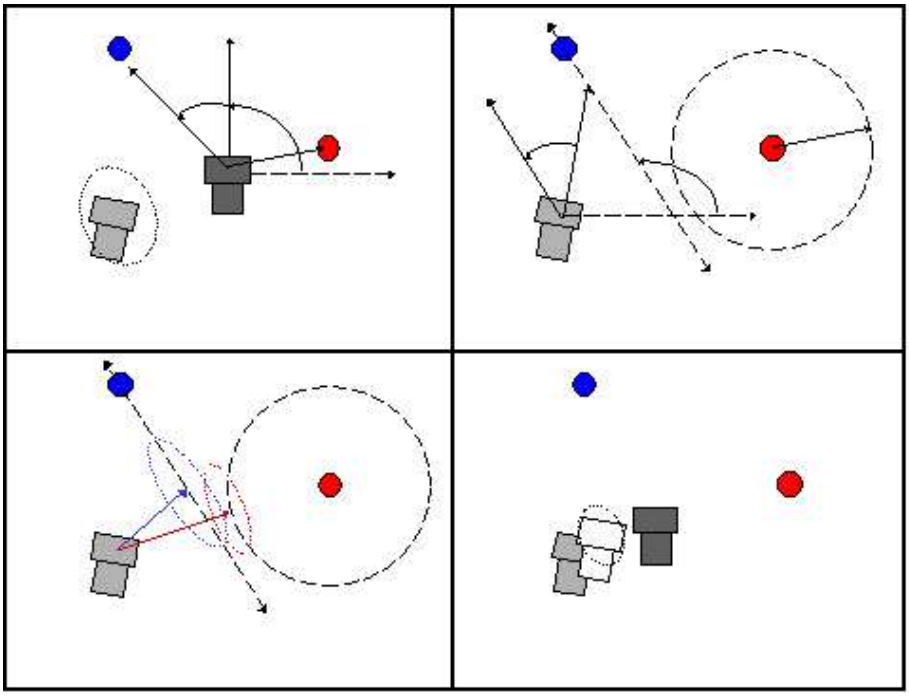

Figure 8: In the upper left corner, the robot acquires one range and one bearing estimate from two separate landmarks. Next, in the upper right panel, it calculates a circular constraint from the range measurement and a linear constrain from the bearing constraint. In the lower left corner, it fits Gaussian distributions to the closest constraint points. In the lower right panel, it combines its previous pose estimate with the new estimates from the sensor update.

in the bearing measurement. A two-dimensional Gaussian can be oriented along the line with uncertainty perpendicular to the line related to the uncertainty derived from angular uncertainty at that distance.

The separate angle Gaussian is updated similarly. Given our most probable location and bearing toward a marker, the implied angle the robot is facing can be derived using trigonometry. The uncertainty on this new heading is directly related to the uncertainty on the bearing measurement. Heading is maintained in the range $(-\pi, \pi]$.

Once this approximation is made, Bayes' rule can be applied: $P_{loc|sensor} \propto P_{loc} \times P_{sensor|loc}$. The actual implementation for the position Gaussian follows closely the math in [6] which uses simple matrix operations to update mean and covariance. For the angle Gaussian, we use a simple unidimensional version of this same update.

For quick recovery from teleportation, sanity checks were performed between

previous pose estimates and sensor-based pose estimates prior to updating. In the case of disagreement, the update is made and uncertainties are set high to allow quick convergence to the new pose.

For process updates, a probability distribution is needed to represent the resulting pose given prior pose and motion command. The parameters of this model were approximated as two-dimensional Gaussian for position and one-dimensional Gaussian for heading. The means are updated by adding the mean of the model distributions and covariances are updated additively to grow the uncertainty as the robot moves.

2.2 Performance

PCBL performs well in the soccer environment. Localization updates were performed quickly enough to negligibly affect cycle time. When placed at different locations on the field, the robots were able to localize to the correct location in less than 5 seconds. Recovery from teleportation occurred quickly as well. Corrections to overcome collisions and lack of progress against obstacles occurred within a couple updates incorporating evidence from multiple landmarks once robots could observe landmarks. The ability of this simplified approach to converge to the correct answer is due to the large number of landmarks available on the field.

3 The World Model

The world model was used to track relevant objects on the field, even when they are out of the robot's view. The world model stores the position of the ball and the teammate and opponent robots, all in global coordinates. The world model also uses the known position of the robot to return the position of the goals in local coordinates.

3.1 Ball tracking

The position of the ball was stored as a single Gaussian, in global coordinates. The position of the ball was updated primarily from vision. If the ball was not seen in a given update step, its standard deviation was increased. However, if the ball had not been observed for long enough to consider it "lost" (about 5 seconds), the position of the ball was requested from the Shared World Model [5]. If one of the robot's teammates had observed the ball, this ball position was available from the shared world model. By sharing information with its teammates, each robot was able to know the position of the ball, even if it had not itself observed the ball recently.

3.2 Robot tracking

Opponent robot and teammate robots are tracked independently. Because we use communication to share information between teammates, each robot has a shared world model that contains the position of each of its teammates, as reported by that teammate. These positions are trusted by the world model, and visual observations of the teammates are discarded.

Opponent robots are observed by attempting to match an observed robot position to a previously observed opponent position. This is done by finding the closest opponent position to the new observation. If the closest position is closer than a threshold distance (0.5 meters), the two observations are merged. If no previous opponent position is within the threshold distance to the new observation, the oldest position is replaced by the new observation.

3.3 Goal position

The goals exist in known locations on the soccer field. The world model contains a method that converts this known global location into a local position given the robot's estimate of its position. The goal positions can be returned either as one location, in the center of the goal, or as two locations, one for each goal edge.

4 Behaviors

The addition of a fourth robot to each team and the increase in field size created a need for explicit coordination between agents. Fortunately, wireless communication provided a medium through which to collaborate. Our focus this year when designing behaviors for CM-Pack '02 was on how to take advantage of communication to assign roles and coordinate the movement of the different robots [10].

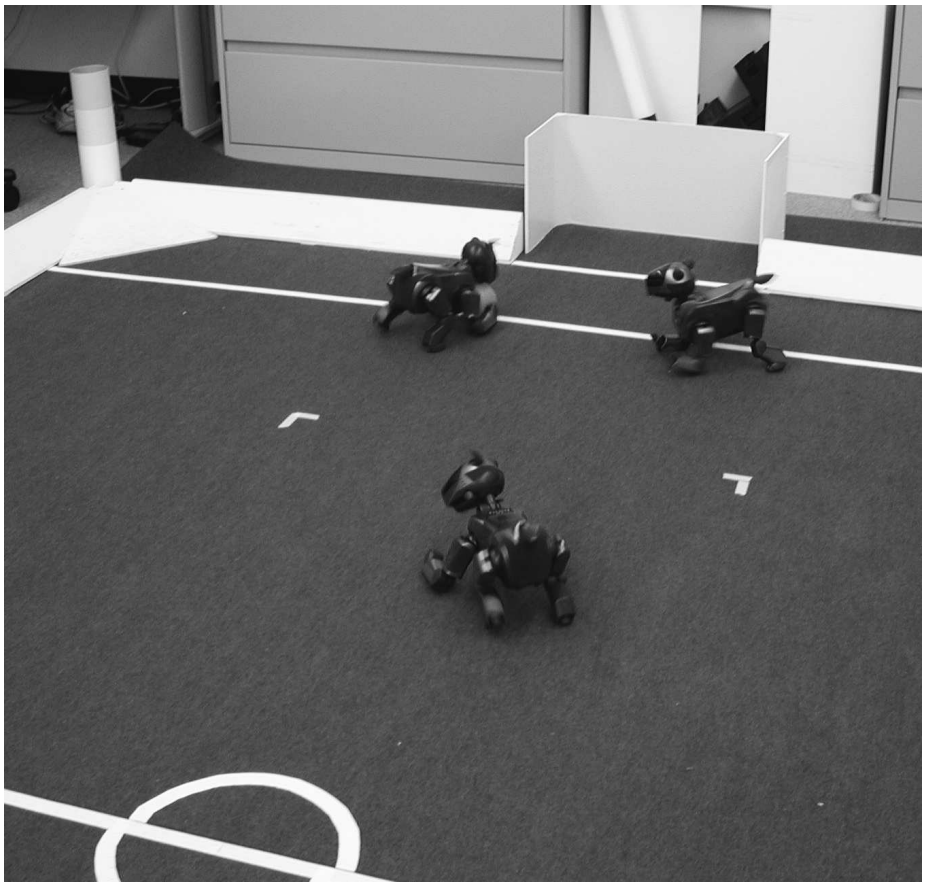

Figure 9: The primary attacker holds the ball slightly to the left of the other team's goal. The offensive supporter waits by the right edge of the goal to recover the ball if the primary attacker misses. The defensive supporter positions itself down field of the ball to recover it if the ball moves behind the primary attacker. The cylinder standing at the left corner of the field is one of the markers used for localization.

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. Figure 9 shows the robots positioning themselves in these roles.

The three agents negotiate among themselves using a predefined protocol so that a single robot fills each role. In addition, they coordinate with the goalie to avoid approaching the ball while the goalie is clearing it from the defense zone and they avoid collisions with their teammates.

Before providing the details of how the different roles are assigned and how the robots fill those roles, we briefly describe information sharing between teammates.

4.1 Shared information

In our framework, the robots must communicate in order to coordinate effectively. Coordination methods that rely on local information alone are not feasible in this domain since there are many cases where a robot cannot observe the ball or its teammates. Since a known, small number of robots are collaborating, we chose to use a system of broadcast messages to share information. This approach does not scale to large numbers of robots, but it is very simple to implement and understand.

Twice a second, each robot broadcasts a message to its teammates. This message contains the current position of the robot, according to its localization system, as well as an estimate of the uncertainty in that position. The message also contains the robot's estimation of the ball's position and the uncertainty associated with that measurement. The final pieces of information included are flags indicating whether or not the robot is the goalie and if the robot currently sees the ball. The goalie flag is needed for role assignment since the goalie can never play a different position and is the only robot allowed to clear the ball from the defense zone. The flag indicating whether or not the robot currently sees the ball is used when building a shared world model to avoid incorporating evidence about ball location from robots that do not see it.

A detailed explanation of the shared information and how this information is combined may be found in [5]. Next we describe how this shared information is used to assign roles to different agents and how the agents fill those roles.

4.2 Role assignment

The three robots playing offense need to be assigned to the roles of primary attacker, offensive supporter, and defensive supporter. Role assignment is done in a fixed, total order. The primary attacker is chosen first, followed by the defensive supporter, and finally the offensive supporter is picked. This order is designed to make the system more robust; if one or two of the robots fails, the remaining member(s) of the team can carry on playing.

All of the robots use a common set of functions to calculate real valued bids for each task. These functions encode heuristic information about the world to return an estimate of how suitable the robot is for a particular task. For example, the bid function for the primary attacker activation takes ball proximity and the relative orientation of the opponents' goal into account. Robots first calculate their own suitability using local information from their world models and then they use the same function to calculate the bids of their teammates using only the shared information provided by each teammate.

It is important to note that only the reported information is used for calculating teammates' bids; in effect the agent doing the calculation is putting itself into the shoes of the agent whose bid is being calculated. If the agent performing the calculation used its own information, it could erroneously assign another agent a different potential than it calculated for itself.

A concrete example of this would be when the agent performing the calculation sees the ball next to the other robot, but the agent whose bid is being calculated does not see the ball (perhaps the ball is behind it or occluded by an opponent). Since the robot that cannot detect the ball is not confident about the ball location, it will assign itself a low bid for the primary attacker role. The agent that sees the ball should not assign its teammate a higher bid than the teammate would pick for itself. And the teammate should not use the shared information from the robot that sees the ball to assign itself a higher value. In the case where an opponent is occluding the ball, the robot does not have a clear path to the ball; although it would be reasonable for it to use the shared information to turn to the ball.

Once each robot calculates the bids for itself and each of its teammates, it compares them. If it has the highest bid for the role being assigned, it assumes that role. If it was not the winner, it assumes that the winning robot will take up the role and performs calculations for the next role in the list. The winners of previous auctions are not considered in subsequent auctions for different roles; they have already been assigned a task. In principle, all of the robots are performing the same calculation on the same shared data, so they should arrive at the same result. In practice, no synchronization is provided, so it is possible for teammates to calculate different bids for each other due to factors such as network delays and transmission errors.

To address this, hysteresis is added to the system. Once a robot takes a particular role, it does not relinquish that role for a short time - on the order of seconds. Since the bid functions are self-reinforcing, that is to say once a robot takes up a role its actions increase its suitability to fill that role, this hysteresis is enough to overcome the lack of synchronization in practice.

Another question is: why broadcast so much information? Why not have agents send only their own bid values? It seems wasteful to broadcast position estimates instead of a real valued bid. However, robots must broadcast their position and their estimate of the ball position anyway. The utility of sharing ball information is obvious; robots frequently find themselves in situations where they cannot see the ball due to distance or occlusion. In these cases, teammates can help each other find the ball. The robot position information shared by

teammates can be used for obstacle avoidance. The robots have a limited field of view so vision alone is not sufficient to detect neighboring robots.

We present a bidding function to calculate the robots' activation for the primary attacker role as a concrete example. Bid functions for other roles may be designed in a similar manner and there are many other possible functions that could be used for the primary attacker auction. For example, it might be desirable to take localization uncertainty into account in a principled way. This particular function is designed to produce high bids when robots are close to the ball and also to take into account how well lined up the robot is to kick the ball into the opponent's goal.

$$Bid = \underbrace{\frac{\theta_{goal}}{\pi}}_{angular\ component} + \underbrace{(1 - min(1, d_{ball}))}_{distance\ component} \quad (1)$$

In this equation, θ_{goal} is the angle formed between the line running from the robot to the ball and the line from the ball to the goal. When θ_{goal} equals π, the robot is perfectly lined up to kick the ball into the opponents' goal. The d_{ball} parameter is the distance from the robot to the ball in meters. This distance is capped at 1 meter.

4.3 Coordination

The robots use the same mechanism for both coordination and obstacle avoidance. They overlay a potential field over the environment and sample local points in the field to approximate its slope at their current location. They follow the gradient of the potential field until they reach a local minimum. The components of the field are designed such that local minimums arise at positions from which the robots can support the primary attacker. In the case of the offensive supporter, the field guides the robot to a good position to receive passes or recover the ball if the shot on goal goes wide. In the case of the defensive supporter, the gradient guides the robot to a position where it blocks its own goal and can recover the ball if it is intercepted by the opposing team. The primary attacker does not make use of the potential field; it always seeks out the ball and counts on its teammates to move out of its way instead of avoiding them.

The potential field is the sum of several linear components. Each of these components either represents heuristic information about the world, such as the offensive supporter should not block the primary attacker's shot on goal, or obstacle information, such as repulsion terms from the walls and other robots. Typically the components of the potential functions are bounded at zero. This makes the effect of the terms local and helps prevent undesirable interactions between terms.

Currently only teammates are included in the list of robots to avoid due to the difficulty of perceiving other robots. Teammates report their own positions via the wireless network; since opponents do not do this, high fidelity

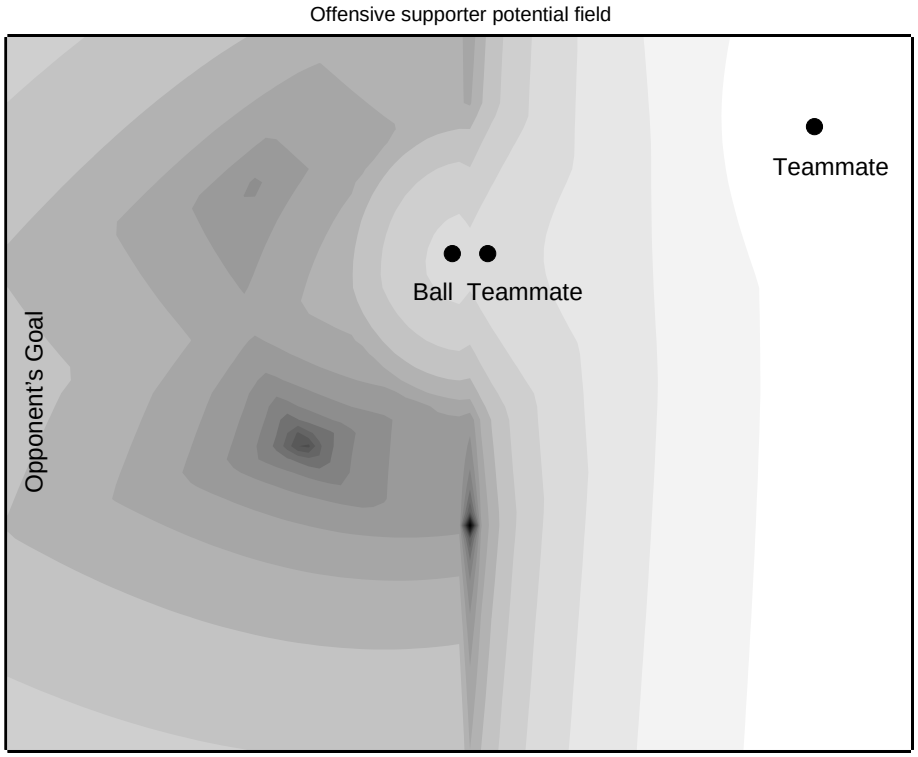

Figure 10: A contour plot of the potential function used by the offensive supporter to position itself on the field. Darker shading corresponds to lower areas of the surface; the robot follows the gradient down to these minimum values. The opponent's goal lies on the left edge of the field and the goal being defended is on the right side.

information about their locations is not available. However, this is a perceptual problem - the composite nature of the functions makes it trivial to add terms for opponents as soon as the perceptual system is able to provide that information.

Depending on their supporting role, the robots may use different subsets of the components. For example, the offensive supporter does not use the component that guides the robot to positions between the ball and its own goal - that heuristic information is not applicable when filling an offensive role.

Next we review the individual components of both the offensive and defensive supporters' potential fields. In the following equations c_n indicates a positive constant and k_n indicates a positive slope.

$$P_{wall} = max(0, c_1 - k_1 \cdot d_{wall}) \tag{2}$$

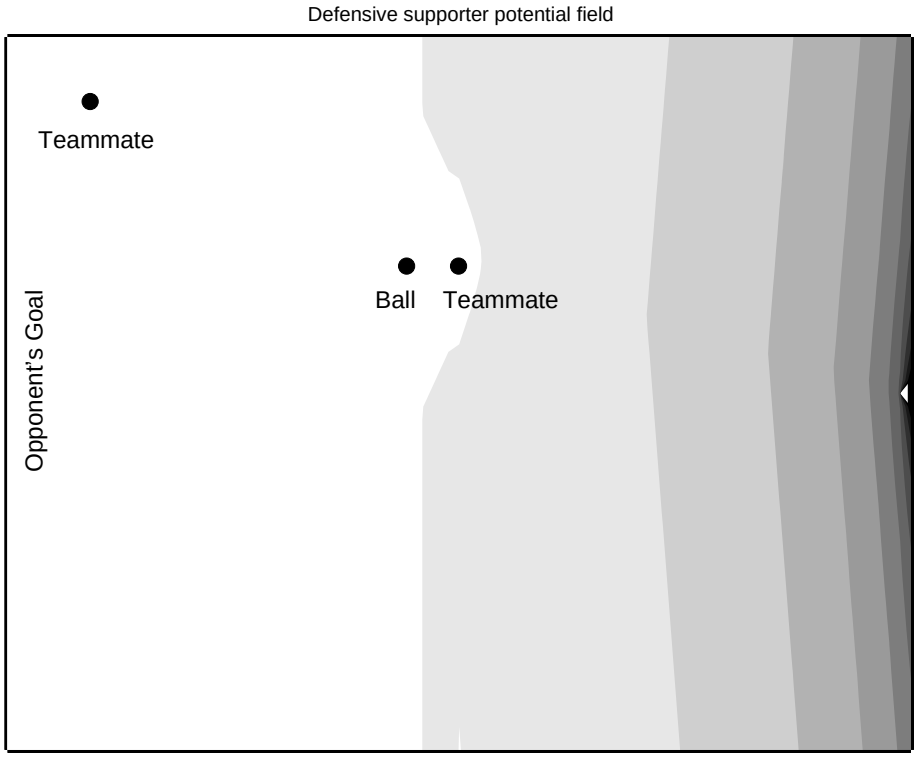

Figure 11: A contour plot of the potential function used by the defensive supporter to position itself. Darker areas correspond to lower values and the agent navigates down the gradient. The opponents goal is on the left edge of the plot and the goal being defended is on the right edge.

The wall potential term encodes a linear repulsion from the walls and the team's own defense zone; only the goalie on each team is allowed to be in the defense zone. c_1 is a positive base potential for when the robot is at the wall. The potential falls off linearly with the distance of the robot from the wall with a slope of k_1. This term is shared by both the offensive and defensive supporters.

$$P_{ball} = \|c_2 - d_{ball}\| \cdot k_2 \tag{3}$$

The ball potential term guides the offensive supporter to a position that is an equilibrium distance, c_2, away from the ball. The potential increases linearly with a slope of k_2 as the robot moves away from the equilibrium distance.

$$P_{teammate} = max(0, c_3 - k_3 \cdot d_{teammate}) \tag{4}$$

The teammate repulsion potential is a positive value that falls off linearly with distance. As with wall repulsion, this term is shared by both types of supporter.

$$P_{forward\,bias} = max(0, k_4 \cdot d_{behind\,ball}) \tag{5}$$

The forward bias potential guides the offensive supporter to a position parallel to or in front of the ball. The $d_{behind\,ball}$ parameter encodes how far the offensive supporter is down field from the ball.

$$P_{defensive\,bias} = k_5 \cdot d_{from\,goalline} \tag{6}$$

The defensive bias potential is analogous to the forward bias only it acts on the defensive supporter. It forces the robot to remain in a position close to its own goal; it increases in value linearly as the robot moves up the field away from the goal line.

$$P_{ball\,corridor} = \|c_6 - d_{shot\,path}\| \cdot k_6 \tag{7}$$

The ball corridor potential encodes the heuristic information that the offensive supporter should not block shots on the goal, but it should also position itself close to the path taken by the ball in order to recover the ball if it stops before reaching the goal. c_6 represents the equilibrium distance of the agent from the ball path. $d_{shot\,path}$ is the actual distance of the agent from the path. The shot path is defined as the line segment from the ball to the center of the opponent's goal line. The offensive supporter is the only robot that uses this potential.

$$P_{block\,goal} = d_{block\,path} \cdot k_7 \tag{8}$$

The block goal potential guides the defensive supporter to a position on the line between the ball and its own goal. $d_{block\,path}$ is the distance between the robot and the line segment running from the ball to the center of the robot's goal line.

$$P_{side\,bias} = max(0, k_8 \cdot offset_{robot} \cdot \frac{offset_{ball}}{\frac{1}{2} \cdot width_{field}}) \tag{9}$$

The side bias term applies only to the offensive supporter. It encodes the fact that the robot should position itself across the field from the primary attacker. The $offset$ terms represent the offset of either the ball or the robot from the line drawn between the centers of the two goals. Notice that this is not a distance - the offset has a negative value for one half of the field.

4.4 The Primary Attacker

While the offensive and defensive supporters simply move into desirable positions on the field, the primary attacker performs more complicated actions as a part of its role. The focus of these actions is to move the ball upfield toward the opponents' goal. If the primary attacker is close to the ball, it traps the ball between its front legs, turns to face the goal, and kicks the ball. The only exceptions are when the robot is facing a wall, in which case it attempts to move the ball off the wall using its head or if turning all the way to face the opponents' goal would cause a holding penalty, the primary attacker dribbles the ball briefly before capturing it again. If the robot is not next to the ball, it alternates between walking to its last known position (or the shared position that is broadcast by a teammate) and the center of the field.

5 Motion

The motion system for CM-Pack '02 has to balance requests made by the action selection mechanism with the constraints of the robot's capabilities and requirement for fast, stable motions. The desirable qualities of such a system are to provide stable and fast locomotion, which requires smooth body and leg trajectories, and to allow smooth, unrestricted transitions between different types of locomotion and other motions (such as object manipulation). The motion system is given requests by the behavior system for high level motions to perform, such as walking in a particular direction, looking for the ball with the head, or kicking using a particular type of kick. We decided to implement our own motions for the robots so that we would have full parameterization and control, which is not available using the default motions provided with the robot. The system we used for CM-Pack '02 was based on the design we developed in the previous year [4].

The walking system implemented a generic walk engine that 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 were 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 were 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 could be loaded from permanent storage on boot up. This system also allowed us to use different sets of parameters for different motions. Specifically, one set of parameters was used for motions with a large rotation component, and a different set was used for straight-line motions. Using two sets of parameters allowed us to optimize each for its particular function resulting in faster, more stable movement. In order to switch between motion parameters on the fly while maintaining stability, parameters were selected to be as similar as possible without compromising their individual effectiveness. Using this interface, we developed a semi-dynamic trotting gait with a maximum walking speed of 200 mm/sec forward or backward, 170 mm/sec sideways, or 2.1 rad/sec turning.

Additional motions could be requested from a library of motion scripts stored in files. This is how we implemented get-up routines and kicks, and the file interface allowed easy development since recompilation was not needed for modifications to an existing motion. Adding a motion still requires recompilation, but this is not seen as much of a limitation since code needs to be added to the behaviors to request that that motion be used.

The overall system was put together using a state machine that included states for walking, standing, and one for each type of kick and get-up motion. Requests from the behaviors would execute code based on the current state that would try to achieve the desired state, maintaining smoothness while attempting to transition quickly.

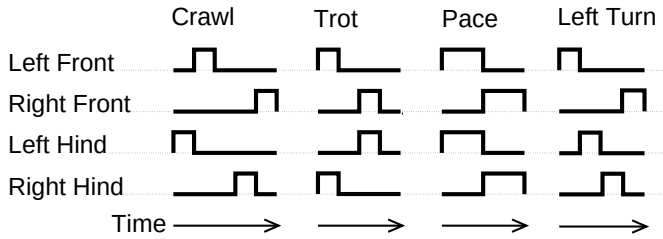

Figure 12: A timing diagram of various walking gaits. the function is zero when the foot is on the ground and nonzero when it is in the air.

5.1 Approach

The technique we used in implementing the walk was to approach the problem from the point of view of the body rather than the legs. Each leg is represented by a state machine; it is either down or in the air. 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.

5.2 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 a roots of the following function with respect to θ:

$$F(a, b, d) = \text{Root}_\theta(a \cdot \sin(\theta) + b \cdot \cos(\theta) - d)$$

Note that since the equation can be normalized, there exists exactly two solutions to the problem for $\theta \in [0, 2\pi]$. The problem above can be represented graphically:

We can then solve for θ with trigonometry:

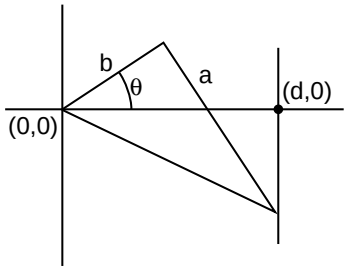

Figure 13: Graphical representation of $a \cdot \sin(\theta) + b \cdot \cos(\theta) = d$

$$\theta \quad = \quad \tan^{-1}\left(\frac{a}{b}\right) \pm \cos^{-1}\left(\frac{d}{\sqrt{a^2+b^2}}\right)$$

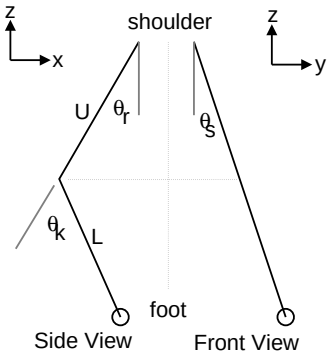

Figure 14: Model and coordinate frame for leg kinematics.

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 $k \in \{x, y, z\}$ [3]. 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 14.

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:

$$
\begin{aligned}
a &= 2 \cdot (u_x \cdot l_z - u_z \cdot l_x) \\
b &= 2 \cdot (u_x \cdot l_x - u_z \cdot l_z) \\
d &= \|p\|^2 - \|u\|^2 - \|l\|^2 - 2 \cdot u_y \cdot l_y \\
\theta_k &= -F(a, b, d)
\end{aligned}
$$

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, or again using the form of F, simplify the problem as:

$$
\begin{aligned}
q &= u + R_y(\theta_k) \cdot l \\
a &= -q_z \\
b &= q_y \\
d &= p_y \\
\theta_s &= F(a, b, d)
\end{aligned}
$$

Now we can calculate a new leg position r that differs from p only in rotator angle. Using a similar approach as before we can determine the required angle to rotate r into the plane of p_x:

$$
\begin{aligned}
r &= R_x(\theta_s) \cdot q \\
a &= -r_z \\
b &= r_x \\
d &= p_z \\
\theta_r &= -F(a, b, d)
\end{aligned}
$$

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.

5.3 Air path and target selection

The air path is one of the most unspecified parts of the walk, in that the path needs only to allow the foot to clear the ground while moving forward for it to work in our system. The air path and foot placement target are very important however for keeping the robot stable and maintaining the foot position within its reachability space during the entire walk cycle. The air target was chosen

so that after the foot is set down it will pass near the neutral position at a specific time in the walk cycle. This can be achieved by evaluating the expected future position of the robot using the current body trajectory, plotted using the current robot's speed and the constant acceleration bound.

Once the target location for the foot is known, all that is needed is a path for the foot in the air. We chose to represent this using a piecewise polynomial, specifically a Hermite cubic spline [3]. A Hermite spline is specified by two points (p_0, p_1), and two derivative vectors, $(\delta p_0, \delta p_1)$, and can be calculated as shown below. Thus we can specify the initial and target point, along with the velocity with which to pick up or place down that foot, and the spline will generate a smooth path between the two.

$$
\begin{aligned}
H(t) &= [x(t)\ y(t)\ z(t)] \\
&= \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}^T \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ \delta p_0 \\ \delta p_1 \end{bmatrix}
\end{aligned}
$$

References

[1] J. Bruce, T. Balch, and M. Veloso. CMVision (http://www.coral.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] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics, Principles and Practice*. Addison-Wesley, Reading, Massachusetts, second edition, 1990.

[4] S. Lenser, J. Bruce, and M. Veloso. CMPack: A complete software system for autonomous legged soccer robots. In *Autonomous Agents*, 2001.

[5] Maayan Roth, Douglas Vail, and Manuela Veloso. A world model for multi-robot teams with communication. In *Proceedings of ICRA'03, the 2003 IEEE International Conference on Robotics and Automation*, Taiwan, May 2003. under submission.

[6] Ashley Stroupe, Martin C. Martin, and Tucker Balch. Merging gaussian distributions for object localization in multi-robot systems. In *Proc. of the Seventh International Symposium on Experimental Robotics (ISER '00)*. Springer-Verlag, December 2000.

[7] Ashley Stroupe, Kevin Sikorski, and Tucker Balch. Constraint-based landmark localization. In *Proceedings of the 2002 International RoboCup Symposium*, 2002.

[8] R. E. Tarjan. *Data structures and network algorithms*. 1983.

[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*, Berlin, 2002. Springer Verlag.

[10] Douglas Vail and Manuela Veloso. Multi-robot dynamic role assignment and coordination through shared potential fields. In *Proceedings of ICRA'03, the 2003 IEEE International Conference on Robotics and Automation*, Taiwan, May 2003. under submission.