CMDash'07: Technical Report

Manuela Veloso, Juan Fasola, Somchaya Liemhetcharat, Mike Phillips,
Gregory Delmar

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

1 Introduction

The CMDASH'07 team follows our past teams CMDash'06, CMDash'05, CMPack'04, CMPack'03, CMPack'02, CMPack'01, CMPack'00, CMTrio'99, and CMTrio'98 [5, 3, 7, 6]. We have continued our research into new ways of modeling the world and maintained our focus on robust behaviors and cooperation. In this paper, we provide an overview of the specific technical components in our team and focus in detail on several recent additions.

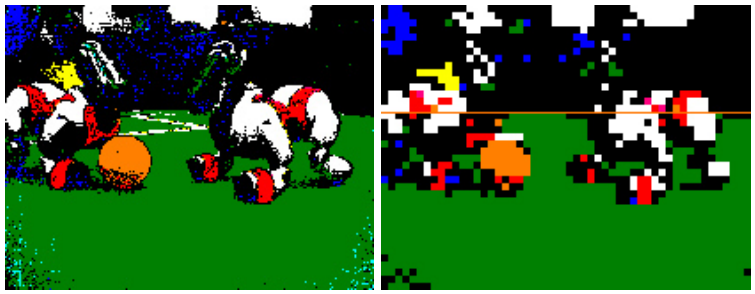
2 Vision

The vision module is responsible for converting raw YUV camera images into information about objects that the robot sees and are relevant to the task. This is done in two major stages. The low level vision processes the whole frame and identifies regions of the same symbolic color. As in previous years, we use CMVision [2, 1] for our low level vision processing. The high level vision module uses these regions to find objects of interest in the image. We have continued to improve both aspects of the robot's vision which allows our team to operate well in a variety of lighting conditions.

2.1 Robot Detection

One way to address the problem of robot detection is to search for red or blue regions in the color segmented image, and if the regions conform to some defined constraints, a robot is then detected. This seems like a reasonable solution, however, in our experience the shade of blue of the robot uniforms is so dark that it shows up as black in the color segmented images. Since the background color is black and the shadows on the robot are also black, finding blue robots effectively with this method is difficult. In addition, trying to correct the segmentation to emphasize the recognition of the uniform color blue results in classifying a lot of the background and shadows as blue as well, so the problem still persists. Instead, we chose a solution that would first identify where the field was located in the image and subsequently detect obstacles lying on the field. Finally, the detected obstacles were assumed to be robots, as the only obstacles on the field during game play, besides the ball, are robots.

Pre-processing the segmented image The first pre-processing step is to reduce the image size of the segmented image. The original 208x160 segmented image is reduced by a factor of four, by essentially scanning across the image along grid lines that are parallel to the image horizon line and recording one out of every four pixels. By scanning along these grid lines instead of the image rows/columns, the algorithm is able to account for any rotation of the AIBOs camera. The image is reduced in order to simplify and speed up the task of searching for robots in the image. All further processing is performed on this reduced image. After size reduction, the image is scanned to find the field horizon. The field horizon is defined as the line that best separates the background from the soccer field in the original image, and that is parallel to the image horizon line. The field horizon is detected by scanning the columns of the reduced image looking for the start of a sequence of green pixels. The height of the highest point in the image that starts a sequence of green pixels denotes the location of the field horizon. After the field horizon has been detected, field lines are removed by scanning the image columns for small white regions surrounded by green. Field lines and the center circle are removed because they tend to interfere with the detection of obstacles on the field. Figure 1 shows an example of a reduced image and detected field horizon.

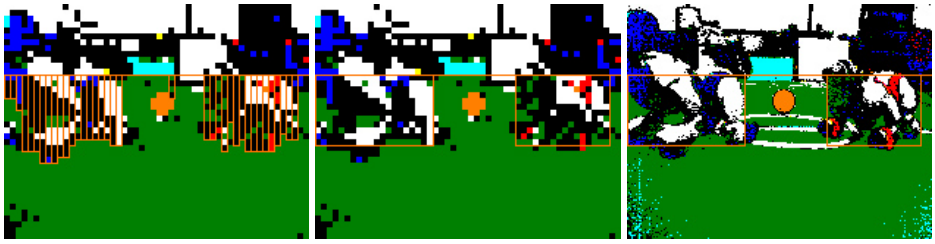


(a) The original segmented image (b) The reduced image with a detected field horizon

Fig. 1.

Detecting Obstacle Regions Using the assumption that robots are always on the field, we can search below the field horizon for pixels that are not green (not field) and not orange (not ball) and group them to form obstacles, which will eventually become our detected robots. Each column in the image is scanned below the field horizon searching for the start of a sequence of four green/orange pixels, at which point the scanned length along the column is recorded and scanning is initiated on the following column. This procedure essentially finds the length along each column of an obstacle that intersects the field horizon line. Columns with small lengths are thrown out. Neighboring columns with lengths a reasonable amount apart are grouped together. The lengths of all the columns within a group are averaged to find the length for that group. Using this length,

along with the starting and ending column positions, a bounding box is created for each group; these bounding boxes represent the obstacles located on the field. Resulting bounding boxes that are too small or in some way inadequate are removed. The final bounding boxes are considered to encompass robots and represent the output of the robot detector. Example outputs are shown in Figure 2.



(a) Initial column scan results for obstacle colors

(b) Final detected obstacles after grouping columns

(c) Robot detection overlayed onto original image

Fig. 2.

Robot Color and Distance Estimates The team color is determined by searching for red regions within the bounding box of the detected robots. If convincing red regions are found within a robots bounding box, the robot is said to be on the red team; blue team otherwise. The distance to a detected robot is calculated by taking the center pixel of the bottom line of its bounding box and projecting a ray through it onto the ground plane. The distance to the intersection point is considered to be the distance to the robot. The robot detector is able to detect robots up to a maximum distance of approximately 1.2 m away. Results of the robot detector along with the predicted team colors are shown in Figure 3.

3 Localization

The CMDASH'07 team continues to use our Monte Carlo localization (MCL) system developed for use on the AIBOs to continually estimate their global location on the field. Our MCL algorithm uses a sensor-based resampling technique to re-localize the robots when they are lost called Sensor Resetting Localization [4]. This algorithm is robust to modelling errors including unmodelled movements, such as robots being pushed or removed for penalties, as well as for systematic errors. The field landmarks, goals, and field lines are used as features for the localization system.

4 World Modeling

To act intelligently in complex and dynamic environments, teams of mobile robots must estimate the position of objects by using information obtained from a wide variety of

Fig. 3. A sequence of images showing robot detection results.

sources. Some examples of such sources include tracked object information from each robot's sensors, models of how each robot is able to affect the environment and manipulate those objects within it, and information obtained from teammates. For any problem of reasonable complexity, teams of mobile robots do not have the sensors necessary to perceive all aspects of a dynamic environment. As a result, the sources of information used in such a state estimator can be corrupted by noise that is extremely difficult if not impossible to fully model. In our research into robust teams of robots capable of operating within dynamic adversarial environments, we have found that not all sources of information about a single quantity of interest can be handled equally. To address this problem, we utilize a method for reasoning over a discontinuous hypothesis space where a strict ordering is imposed on the sources of information. By segmenting the information sources into different classes, a prioritized hierarchy of state estimates are inferred. Using this hierarchy, the decision process that governs each individual robot's actions can easily select the most informative state estimate to use as its input. A robot's actions can affect its perception of the environment as well as the environment. By reasoning about the expected utility of certain classes of estimates over others, the robot can select the "best" estimate from the set to act upon. This will provide more information to the robot that will in turn update the ordered hierarchy of possible estimates.

4.1 Updating Robot Models/Merging Robot Detection Results

Every vision frame presents new robot detection results which must be incorporated into the global model of the game state and potentially matched against previous observations of robot locations. Merging the robot detection results with current models of teammate and opponent robots is necessary to avoid situations where two or more

robot models are created which pertain to the same physical robot. Such situations are undesirable for two reasons:

1. The world model ceases to accurately represent the current game state.
2. The robot models are not able to incorporate new locations into their position history, thus limiting the accuracy of the motion model.

One method of modeling robot locations which does not involve merging subsequent detection results is to project the current field of view of the robots camera onto the playing field of the world model, remove all robot observations contained within the viewing area, and replace them with new robot detection results from the current vision frame if any exist. This method of robot modeling avoids the merging/matching problem by relying on the notion that when the world model determines that there are teammate or opponent robots located in the field of view of the robots camera, the new robot detection results will provide the most updated locations for such robots, and therefore allow for the previous observations to be discarded beforehand. While this method provides a simple mechanism for modeling robot locations and avoids, for the most part, creating multiple models for single physical robots, it does not maintain a position history for the robots being modeled and hence does not provide enough information for velocity estimation/motion modeling. Since we are interested in modeling the motion of teammate and especially opponent robots, we have not adopted this simplified modeling method and instead have implemented a version which maintains a history of recorded positions on the playing field for each modeled robot and hence performs the necessary merging/matching of new robot detection results with those made previously.

Update Algorithm Merging new robot observations with previous observations from the world model involves matching robot models with new robot position estimates, or rather, identifying which robot model, if any, should get updated with a new position estimate. Robots are represented in the world model as (x,y) locations in the global coordinate system of the soccer playing field, however for matching purposes it is also necessary to take into account the physical dimensions of the robot. The modeled robot radius is double the actual physical robot radius to account for error in the location estimates. The following is an outline of the robot model update algorithm:

The time-to-live counter is decremented for all robot models in the current field of view of the robot projected onto the playing field. The time-to-live counter represents the number of frames allowed that contain the robot model inside the projected field of view, but not detected from vision, and is initially set to 10 frames. Once the time-to-live counter reaches zero, the robot observation is removed from the world model, as it is deemed inaccurate. This field of view reasoning is similar to the one used in the simplistic method presented earlier, however instead of immediately removing robot observations that are within the projected field of view, the time-to-live counter is introduced to provide robustness against errors in the visual detection method and field of view estimation.

All robot observations returned by vision are compared against the robot models already existing in the world model, in order to match those observations that correspond to the same physical robot. In finding a match for a new observation, all potential

matches of robot models are evaluated and the best of them all, the one most resembling the new observation, is updated with the new position estimate. A robot model is considered a potential match with a new observation from vision when the distance between their respective locations on the field is less than two robot radius lengths. The procedure iterates through the robot models and the first potential match found is considered to be the best match until another potential match is found, at which point the two are compared against one another and the one that is the better fit with the new observation is made the new best match. This procedure continues until all robot models that are potential matches have been evaluated, and the best among them determined. The algorithm for comparing two potential matches takes into account the distance between both robot models and the visual observation location, the team color of each and that of the visual robot, and the position history size of the robot models (larger is better). If both robot models match up equally with the visual observation, then the best match is kept the same.

If a robot model is found to match the new observation returned by vision, its position is updated in the world model. However, if no existing robot model matches the visual observation, then a new robot model is created in the world model at the given location and initialized with a timeout value of 4 seconds. The timeout value is provided in order to remove robot models considered out of date. It should be noted that every time a visual robot observation is matched with an existing robot model, the timestamp is updated and the timeout is essentially reset, so it is possible for any given robot model to exist within the world model for longer than 4 seconds, as long as its position is updated and existence reaffirmed from visual observations.

4.2 Robot Motion Tracking

Tracking the motion of robots in the world model is useful for many aspects of the game, but most importantly, by providing information about the movement of robots on the field the current situation of the game can be more readily assessed. For example, by having a robot observe the motion of an opponent robot walking towards the ball and correctly predict that the opponent will arrive to the ball first, the robot can decide to assume a more defensive role that would try to prevent the opponent from getting an accurate shot on goal, rather than to continue walking towards the ball and potentially getting scored on. In order to achieve this motion and speed assessment of robots on the playing field, velocity estimates are calculated based on the observed sequence of locations through time of the robots being tracked.

Velocity Estimation Each time a new visual robot observation is matched with an existing robot model, the updated position of the robot is stored in the position history list for that robot model. In order to facilitate the velocity calculation on a robot model, the robot position history is actually separated into two lists, the x-position list and the y-position list. Each x or y position entry in the lists are associated with the time it was recorded. To avoid the calculation of spurious or noisy velocity estimates, the position history lists must contain the minimum number of recorded positions before any velocity calculation is performed for the robot model. The minimum number of

positions is equal to 10, and is intended to represent about one-third of a second of motion information. The velocity calculation is performed as follows:

x-velocity: The slope of the least-squares fitting line through the (timestamp, x-position) data points is calculated and represents $\Delta x/\Delta t = vx$, the estimated velocity of the robot in the x-direction.

y-velocity: The slope of the least-squares fitting line through the (timestamp, y-position) data points is calculated and represents $\Delta y/\Delta t = vy$, the estimated velocity of the robot in the y-direction.

The calculated x and y velocities are combined to produce the velocity vector estimate for the robot model, and this is the vector that is reported to behaviors. The velocity vector, once attached to the current position of the robot model, can be used to predict the motion of the robot and enhance situational awareness during game play. Another method would be to predict the future location of the robot based on the calculated least-squares line fit over the data, by finding the x and y positions along the line at a certain time in the future. It should be noted, however, that velocity estimates are only available for robot models that are currently within the field of view of the robot, since no assumptions can be made about robots moving outside of view and we want to minimize error in the estimates as much as possible so as not to provide false information to behaviors.

Motion Tracking Results Figure 4 shows visual results of the robot modeling and velocity estimation systems. The velocity estimate is represented by a vector, which is the visual representation of the direction and magnitude (speed) of motion of the robot to which it is attached. The visual robot detection results are also provided for comparison/evaluation. The example illustrates the velocity estimation fairly accurately capturing the actual path of motion of the robot.

5 Behaviors

The introduction of visual detection and modeling of teammate and opponent robots allows for the creation of a variety of different game behaviors that without such information were either impossible or very hard to implement. Behaviors are able to gain more information about the current state of the game, which allows for better situational assessment and strategy building. Teammate robots can actually react to the presence of opponent robots and reason about the appropriate actions to take based on the situation, instead of blindly reacting only to the ball. For example, consider the situation when a robot walks towards the ball and it suddenly disappears from view when another robot walks in front of it. If the robot has no knowledge on the location of other robots on the field, it would most likely decide to start spinning in place to look for the ball since it has no idea where it could be and all directions of view are equally likely to contain the ball. However, if the robot had information about the locations of robots on the field, it could reason that the ball is being occluded by another robot and that is why it cannot be seen. The robot could then decide to try and move around the robot in question and search for the ball there. Many more behaviors are possible with the introduction of

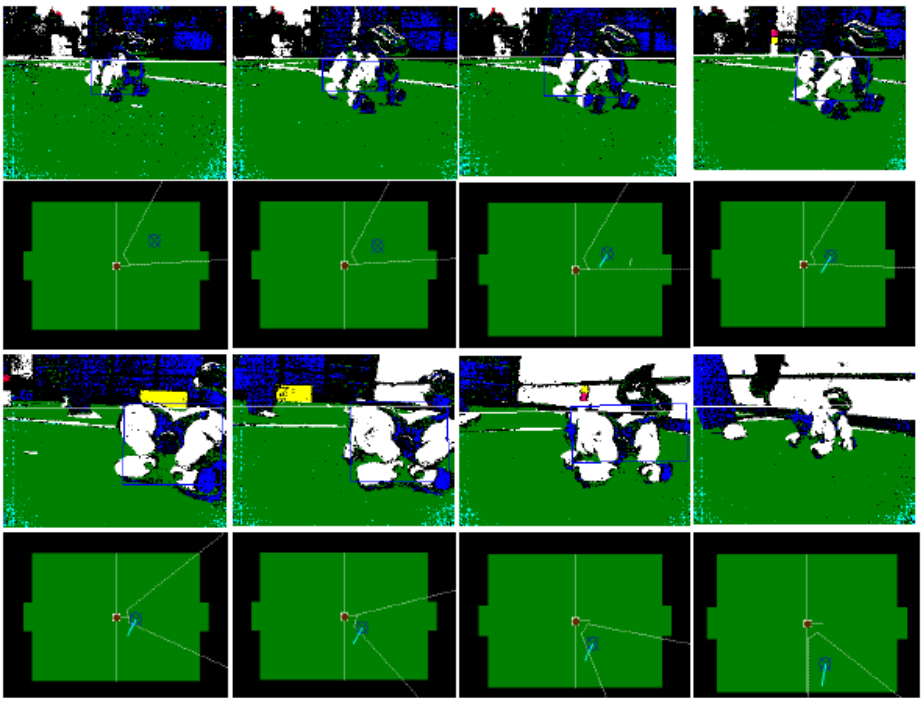


Fig. 4. Visual robot detection/modeling/velocity results from standing robot viewing moving robot walking in nearly linear fashion from left to right

robot modeling, some of which have successfully been implemented and two of these are presented in this section.

5.1 Navigation with Obstacle Avoidance

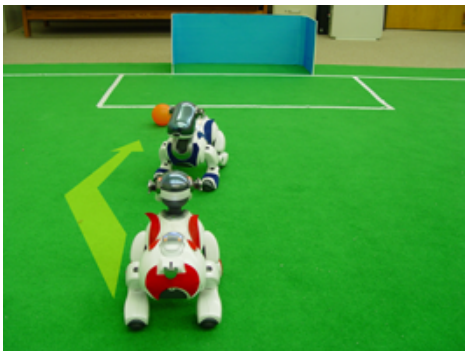
One of the most helpful behaviors as far as improving game play has to be the ability to navigate towards a desired target, whether it be an object like the ball or a global point on the field, while avoiding obstacles. Obstacles on the field are other robots, so having robot locations within the world model and being able to easily access that information greatly facilitates the creation of such a behavior. Not only is it desirable to avoid robots when traveling towards a desired location to prevent running into them, which wastes valuable time and makes reaching the target much more difficult, but also to avoid penalties called against pushing robots. Creating an effective obstacle avoidance navigation behavior will help to minimize these penalties, in addition to helping the robots get to where they want to go.

Navigation Behavior The navigation behavior that was developed consists of two states and has the following state machine, beginning with the initial state:

Go To Point: The robot is directed to walk in a straight-line path directly towards the desired target. If the robot has been in this state for at least 0.3 seconds, is roughly facing the target and there is a robot blocking the path in front, pick the most appropriate side to move to, left or right, depending on whether or not its open and its proximity to the target, and transition to *Avoid*.

Avoid: The robot is directed to move to the side chosen in the previous state while maintaining a minimum distance away from the robot in front. Therefore, the robot need not move directly sideways, but can add a forward walk component as long as the distance to the other robot remains above or equal to the minimum allowable. If the robot has been in this state for at least 0.5 seconds and there is no robot blocking the path in front, or the robot has been in this state for more than 2 seconds, transition to *Go To Point*.

The time constants are to avoid oscillations in the behavior and in the case of the *Avoid* state, also to encourage continuing progress towards the desired target. The behavior is very simple, thanks to the information provided about robot locations in the world model, yet it is very effective during actual game situations. Figure 5 provides an example of a situation where the navigation algorithm is used. The red robot detects the presence of the blue robot in its path towards the ball and decides to avoid it by moving towards the left side, as it is open and closest to the target.



(a) Photo of the obstacle avoidance situation with the path chosen by the algorithm shown in light green

(b) Segmented image taken during situation with robot detector result shown

Fig. 5.

Results The navigation behavior has been successfully integrated into the CMDash team game play, and has helped to improve the team performance during actual competition. Figure 6 provides video capture images taken during a game that show the execution of the navigation behavior and how it allowed our robot to gain control of the ball even though two separate robots from the opposing team were in its way initially.

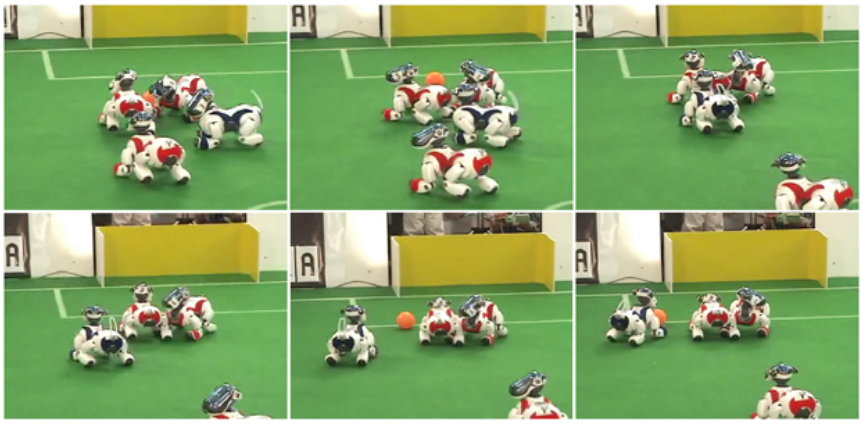


Fig. 6. Video capture images of blue robot performing navigation towards ball with obstacle avoidance during a game

5.2 Dodging Opponents when Shooting

The behavior to avoid, or dodge, opponent robots when attempting to shoot on goal or down the field is a similar yet slightly different behavior from that of navigating while avoiding obstacles. This behavior was first implemented and used with great success by UTS in 2004, when the team reached the finals of the international competition. The dodge behavior only executes after the robot has gained control of the ball beneath its chin and wants to shoot, or kick the ball, towards a specific location. Once the robot grabs the ball, the Turn behavior executes, which attempts to change the orientation of the robot so that it is facing the desired target. Once the Turn has reached the desired orientation, it checks for the existence of opponent robots in front of the robot by accessing all robot locations from the world model, and seeing if any of them fall within the region in front of the robot. If so, the Turn behavior then calls the Dodge behavior, whose job it is to try and avoid the opponent robot before shooting towards the target, so as to get a clear shot off.

Dodging Behavior The dodging behavior has three states:

Initial: The direction of motion is chosen, left or right, in order to avoid the opponent robot in front. The dodge direction is chosen simply based on which side is closer to the inside of the field, because dodging towards the outside of the field and accidentally leaving the field causes loss of possession of the ball. Transition to *Dodge*.

Dodge: The robot is directed to move to the side chosen in the previous state at full speed, with no forward walking component. If the robot has been in this state for at least 0.3 seconds and there are no longer robots in front, then transition to *Kick*. If the robot has been holding the ball for approximately 3 seconds, then abort the behavior to avoid penalty.

Kick: Kick the ball forwards towards the target. The behavior is then reset, and ready to be called again by the Turn behavior if necessary.

Results The dodging behavior, like the navigation behavior, has been successfully integrated into our team game play, and has helped tremendously in increasing the chances of our team scoring after gaining control of the ball while near the opponent goal area. The dodging behavior is also very effective in avoiding opponent robots when trying to clear the ball down the field. Figure 7 shows video capture images of the dodge behavior being executed during competition. The red robot after grabbing the ball detects the presence of the opponent goalie robot, then quickly side steps to its right side until the goalie is no longer impeding the shot, then kicks forward and scores a goal.

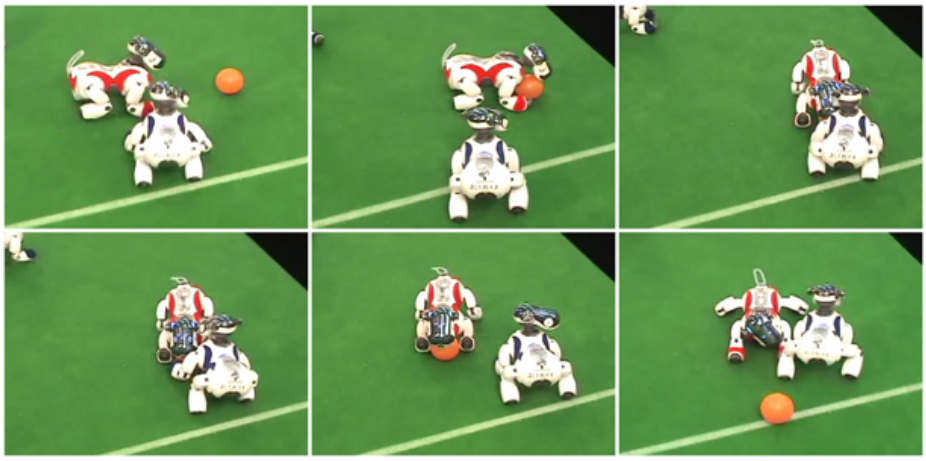


Fig. 7. Video capture images of red robot dodging opponent blue robot before shooting and scoring goal

6 Upcoming Research

The focus of our upcoming research will be on improving our team game strategy for individual robots and among teammates. Currently the robot vision and modelling is only used for navigation and dodging, however we would like to incorporate it into the behaviors to increase situational awareness and to make better strategic decisions. There are various more behaviors that can be created to take advantage of the information provided by the robot detection and modeling procedures, all of which can add a new dimension to the functionality, performance, and teamwork of our robot soccer team. Some of the behaviors which are particularly interesting include: passing to visual teammates, using robot vision for teammate coordination when approaching the

ball or during a designed play, communicating information about moving opponents to teammates that are beyond the visible range of detection, and positioning around the field so as to remain in open regions away from opponent robots.

7 Conclusion

With CMDASH'07 , 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 have built upon our experiences from last year and enhanced our AIBO software with new visual features, more extensive modeling and estimation of the world, improved behaviors which are reactive to the location of opponent robots, and we are working to improve our teammate coordination and game strategy.

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. 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. Lenser, J. Bruce, and M. Veloso. CMPack: A complete software system for autonomous legged soccer robots. In *Autonomous Agents*, 2001.
4. S. Lenser and M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Proceedings of ICRA-2000*, 2000.
5. 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.
6. 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.
7. 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.