From CMDash'05 to CMRoboBits : Transitioning Multi-Agent Research with AIBOs to the Classroom

Paul E. Rybski and Manuela Veloso
Carnegie Mellon University, School of Computer Science
5000 Forbes Ave. Pittsburgh, PA 15213
{prybski,mmv}@cs.cmu.edu

Abstract

introduction

Since 1997, the CORAL research group at Carnegie Mellon, headed by Professor Manuela Veloso, has researched teams of soccer robots using the Sony AIBO robots as the platform (Veloso & Uther 1999; Veloso *et al.* 2000; Lenser, Bruce, & Veloso 2001a; 2001b; Uther *et al.* 2002). Our experience runs across several generations of these four-legged robots and we have met increasing success every year. In the fall of 2003, we created a new course building upon our research experience with the AIBO robots. We have since refined the course and taught it again in 2004 and 2005. The course, which we entitled *CMRoboBits: Creating an Intelligent AIBO Robot*, introduces students to all the concepts needed to create a complete intelligent robot. We focus on the areas of perception, cognition, and action (illustrated in Figure 1), and use the Sony AIBO robots to help the students understand in depth the issues involved in developing such capabilities in a robot. The course has a two-hour weekly lecture and a one-hour weekly lab session. The coursework consists of weekly homeworks and a larger final project. The homework assignments include written questions about the underlying concepts and algorithms as well as programming tasks for the students to implement on the AIBO robots. Evaluation is based on the students' written answers, as well as their level of accomplishment on the programming tasks. All course materials, including student solutions to assignments, are made available on the web. Our goal is for our course materials to be used by other universities in their robotics and AI courses. In this paper, we present the list of topics that were covered in the lectures and include examples of homework assignments as well as the rationale behind them. This curriculum shows how the AIBO and its software resources make it possible for students to investigate and embody an unusually broad variety of AI topics within a one-semester course.

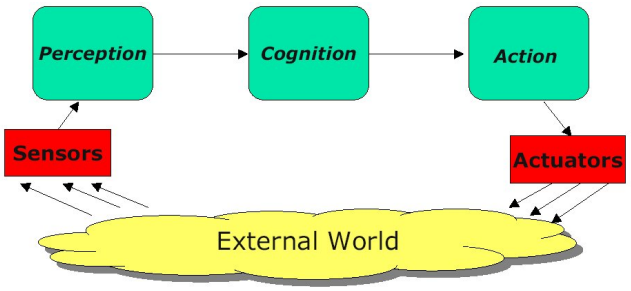

Figure 1: The modules used in the complete robot.

CMRoboBits

The CMRoboBits code distribution provides all the software necessary for complete control of an AIBO. The default code distribution contains all of the functionality necessary to run simple behaviors (such as chasing a ball). Students are able to use the existing codebase for their homeworks and typically write behaviors that make use of the pre-existing systems. For homeworks that expand on a particular aspect of the robot's capabilities (such as vision homeworks), the students are expected to augment the existing CMRoboBits code with the new functionality. Some specific capabilities of the software include:

- Behavior engine for specifying complex robot actions. Behaviors can run in parallel in a priority-based scheme and each behavior can invoke a tree of sub-behaviors.
- Real-time low-level color segmentation algorithm for identifying objects by specific color types. This also includes the tools necessary for learning the segmentation based on labeled training images.
- High-level vision object recognition system for identifying and returning the 3D positions of specific objects such as colored markers, balls, and even other AIBOs.
- Frame-based inverse kinematics engine for performing complex motions with the AIBO's limbs.
- Parameterized walk engine for specifying different trot gaits that the AIBO can use to translate and rotate in any direction.

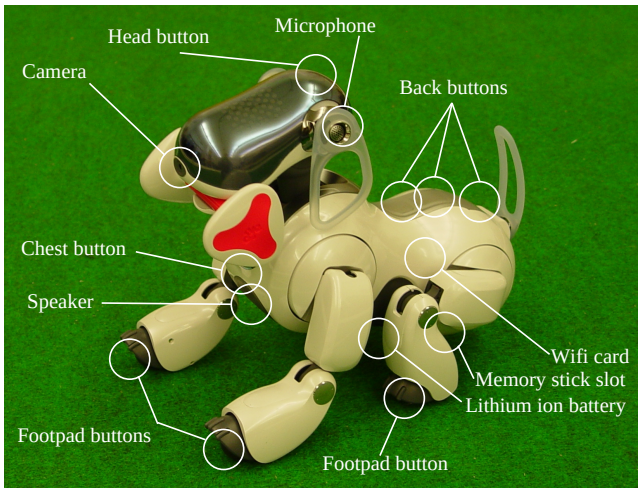

Figure 2: Annotated AIBO ERS-7

- Monte-carlo localization system (particle filter) using landmarks at known positions to track the robot's pose and orientation.
- World modeling system for tracking the positions of detected objects over time.
- Inter-robot communication system for sharing world state with other robots.
- Stream-based debugging tools for viewing debug text strings and frames of color thresholded images in real-time or from log files.

The Goals of the Course and the Schedule

The main goal of the course is to learn how to create an *intelligent* robot, using the AIBO platform as a concrete example. We want the students to understand how *to program* the robots to perform tasks. We view a robot as a complete intelligent agent, in the sense that it includes perception, cognition, and action. Our aim through the course is to *demystify* robot programming so that it becomes clear and accessible to all of our students. A parallel goal of particular interest to us is to bridge from our research code in robot soccer to modular code that can be used for any general robot task. We aim to provide course materials that are modular and well structured so that people at other universities can use the materials in their own courses. We have found that reorganizing and cleaning up our robot soccer code has had several additional positive effects, namely facilitating our research work and easing students' transition into that research.

The AIBO, shown in Figure 2, is a remarkable piece of commercially-available robotic hardware. An AIBO has fifteen degrees of freedom (DOF) in its legs and head, a color CCD camera that can process images at 25-30 frames/second, a 3-axis accelerometer for body pose estimation, buttons on its back, head, and footpads, LEDs for visual debugging, and a wireless Ethernet (802.11b) card for inter-robot and host-computer communication. Students program AIBOs using a free SDK called OPEN-R (found at http://openr.aibo.com/) which lets them compile control code on a workstation with a MIPS cross-compiler (available for GNU Linux, Microsoft Windows, and Mac OSX). The AIBO's low cost (approximately $1,600 US) allows an instructor to purchase several of them for the price of a more traditional research robotic platform.

This 15-week course contains several main components, as we present below. In the first version of the course in the Fall of 2003, we followed these main components. In the second version in the Fall of 2004, we put significantly more emphasis on behaviors, multi-robot cooperation, and learning. Although all the components are relevant to the goals of an AI/robotics course like ours, it is easy to get lost in the low-level details of perception and action, for which there may be other courses specifically addressing these topics. Instead the focus of our course is the complete intelligent agent, hence the emphasis on cognition and learning. We always teach perception and action, but we balance the topics toward cognition. The following descriptions outline how we strike this balance.

Behaviors: This course familiarizes the students with the concept of behaviors for robot control. Every component, from sensors to localization, is cast in the framework of how a mobile robot can use those techniques in its *behaviors*. We re-introduce behaviors at several times in the course, since behaviors are the basic components of virtually any robot task. Initially, we introduce finite-state machines and incrementally address more complex behavioral structures, such as hierarchical behaviors and planning. Figure 3 depicts a decomposition of several parallel and sequential behaviors used in class.

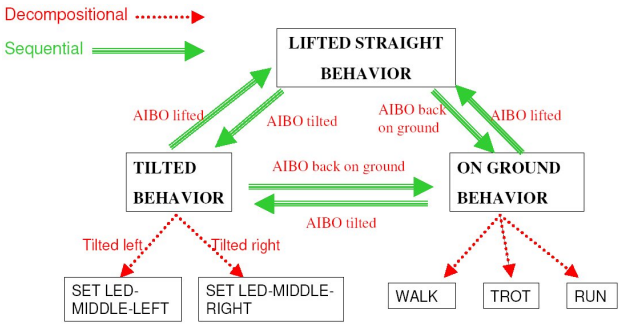

Figure 3: Description of a behavior hierarchy

Sensors and actuators: Robots perceive the world using their *sensors*, and they affect their environment with their *actuators*. Sensors and actuators mediate all interactions between the robot and its environment and are equivalent to input and output operators in computer programming. This component of the course introduces students to the idea of acting in the face of uncertainty. Unlike traditional programming where input values are completely known, robots must perform with only limited, noisy knowledge of their environment. Additionally, robots must cope with noise and uncertainty in the effects of their actions; motors do not always perform the requested movements and

factors such as friction and slip are difficult to take into account when predicting the outcome of actions. Students are introduced to the idea of uncertainty, which is central to robot programming. Figure 4 shows an example plot of the 3-axis accelerometer data that students can use to determine what "state" the robot is in.

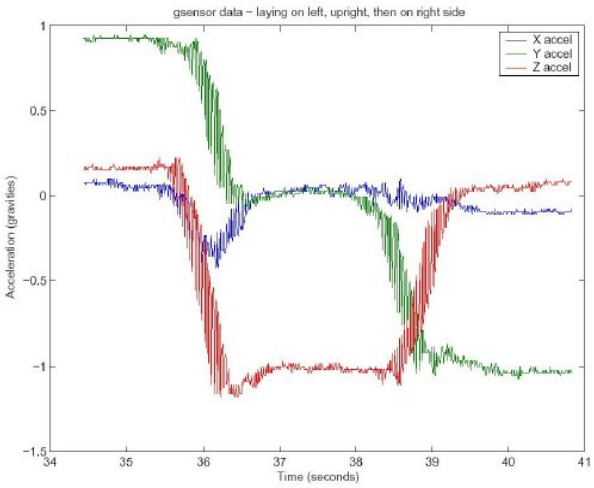

Figure 4: 3-axis accelerometer signature for a robot that starts on its left side, is rotated to an upright position, and then is rotated to its right side.

Motion: The AIBO robots offer an interesting and challenging platform for exploring robot motion. AIBOs differ from most educational robots because they are a legged platform with fifteen degrees of freedom (DOF) in their head and legs. Each of the four legs has three DOF and the head has pan, tilt, and roll joints. In addition to these major joints, students can actuate the tail, mouth, ears, and eye LEDs to create more expressive behaviors. The legged kinematics system makes use of a frame-based motion system and a parameterized walk engine (Bruce, Lenser, & Veloso 2002). In this unit, we introduce students to the ideas of forward and inverse kinematics. We also include a practical introduction to our motion system on the AIBO. We describe our parameterized walk engine, which uses approximately fifty numeric parameters to specify an entire gait for the robot. These parameters include factors such as robot body height, body angle, lift heights for each leg, and timings. We also introduce students to the idea of frame-based motion where all joint angles are specified for a few *key frames* and the robot interpolates between them. This type of motion is useful for scripting kicking motions for soccer, dance motions, climbing, and other predefined motions.

Vision: The AIBO robots use vision as their primary sensor. Color images in the YUV color space arrive at a framerate of 25Hz. As in previous years, we use CMVision (Bruce, Balch, & Veloso 2000;) for our low level vision processing. The high level vision module uses these regions to find objects of interest in the image. The vision unit of the course acquaints students with the basics of robot visual

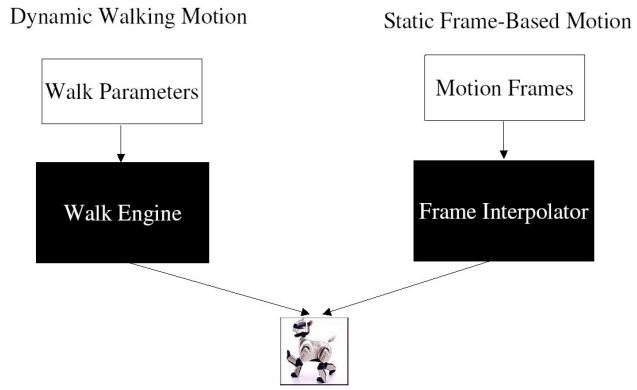

Figure 5: The two motion control systems in the CMRoboBits code base.

processing. Students briefly learn about the YUV color space commonly used by image capture hardware, as well as Real time color segmentation, and camera calibration. The final topics include higher level concepts such as object recognition from the color segmented images, including weeding out false positives. Students also learn how kinematics ties back to vision for calculating the real world position of objects in the vision frames. Figure 6 shows an example frame of post-processed AIBO video. This image illustrates the output of the AIBO's real-time color segmentation algorithm.

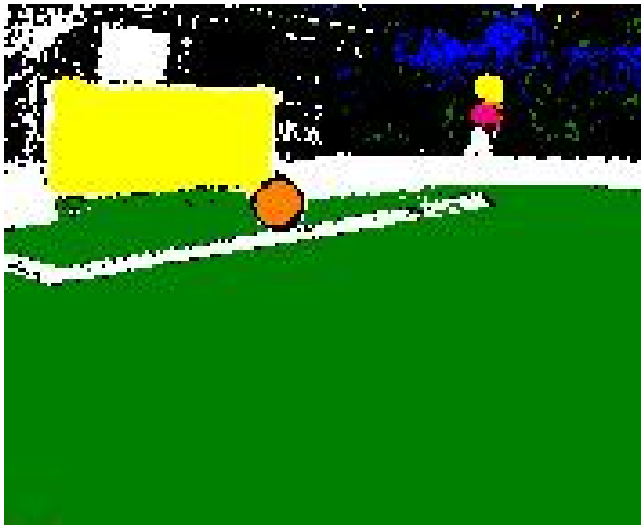

Figure 6: An example frame of video from the AIBO's camera after color-segmentation post-processing.

Localization: In order to act effectively, a robot often needs to know its location in the environment. Localization becomes an essential component that interacts with perception, decision making, and motion. This unit introduces the ideas of probabilistic localization beginning with the basic ideas of probabilistic localization and including different methods of representing locale belief, such as

Kalman filters and particle filters. We also cover more advanced localization challenges such as recovering from errors in motion models (e.g., the kidnapped robot problem) through sensor-based resampling algorithms, and the study of various tradeoffs that may be made between computational cost and resource consumption.

Multi-Robot Cooperation: Once the students understand how to program a single AIBO to do interesting behaviors, we teach them how to use the AIBO's on-board 802.11b wireless Ethernet system. This feature allows the robots to communicate among themselves. We teach the students how to solve problems with multi-robot behaviors, discuss the challenges, and present several approaches for multi-robot communication and coordination, including market-based approaches, and cognitive architectures, such as the skills-tactics-plays architecture.

Figure 7: Students learn some of the challenges with programming behaviors for cooperative multi-robot tasks.

Learning: Throughout the course we explain how robot programming involves a considerable amount of parameter tuning in a variety of algorithms. We introduce learning approaches for motion optimization, control learning, and team play adaptation to an opponent team. Learning is a theme that we have taught in the lectures, but students do not currently investigate it deeply in their programming assignments. We plan on continuing to understand better in the next versions of the course how to make learning more present in the specific robot programming assignments.

Upcoming Capabilities

Every year, the CMRoboBits codebase is derived from our lab's RoboCup AIBO code. This allows the class to be taught with the latest research code and gives the students the opportunity to work with state of the art robotic algorithms and techniques. The new capabilities that will be available in the CMRoboBits 2005 courses and beyond include the following.

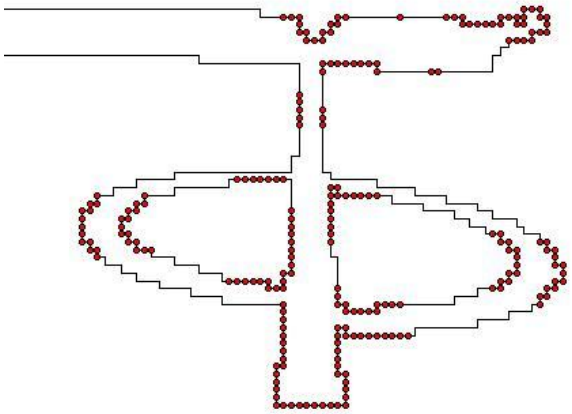

Figure 8: The small circles identify areas of high curvature found on the lines.

Line Vision

In 2005, we extended our vision system with a line vision module that is able to recognize straight lines, corners and the center circle at 30 fops. For all features, both position and orientation are recovered with high accuracy. For the center circle, the orientation is obtained by determining the orientation of the center line that passes through the circle. Corners and the center circle can only be detected up to a distance of approximately 1.5 meters from the robot. These detected features are used by our localization framework.

The line vision module is fast enough to run in parallel with the CMVision color segmentation system. The overall processing is similar to the vision system of the FU-Fighter's MidSize (von Hundelshausen 2004). We use the region tracker (von Hundelshausen & Rojas 2003) to extract the boundary contours of all white regions that are below the horizon. Here we run the region tracking on top of our segmentation results, determining the boundary contours of the white areas in the segmented images. Then we process the contours similar as described in (von Hundelshausen 2004). We split the contours at high curvature points and classify between straight and curved segments.

Parallel straight segments spaced at a small distance are identified to represent the two borders of a field line and are grouped together. Finally both sides of a hypothetical field line are checked for green. Corners are detected on the base of the straight lines. The center circle detection is different from (von Hundelshausen 2004). Because of the larger perspective distortion we could not directly use the MidSize method. The modified method is based on detecting the two halves of the center circles. By using the region tracking algorithm the half circles can detected extremely efficiently because the sense of rotation of the boundary contours of the half-circles is different from the other lines. In each hypothetical half of the center circle we match a triangle, two points being at the intersection of the center line and the circle and the third point located on the curved part of the half

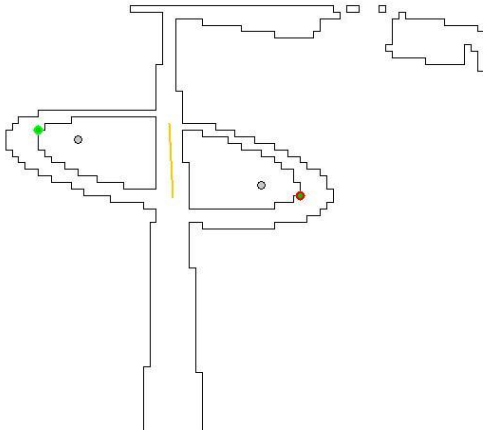

Figure 9: The center circle as identified by the AIBO vision system.

circle. On the basis of these triangles we perform several consistency checks: We have to find two triangles that have two sides that are parallel and close to each other and also we run different checks on the angles of both triangles. Finally, we check whether both halves of the center circle are green. In contrast to the FU-Fighters Middle Size feature detection, the center circle detection will fail in the presence of occlusion. Figure 9 shows the circle identified by an AIBO standing on the center line of the field.

World Modeling

Similar to previous years, the CMDASH'05 world model estimates the positions of the ball, opponents, and teammates on the soccer field and provides a single unified interface to the behaviors. Uncertainty estimates for each of these quantities are calculated as well. New features for this year are different ball uncertainty estimates based on a high-level notion of ball state.

Multiple Ball Hypothesis Tracking An estimate of the ball position is provided by a Multiple Hypothesis Tracking Kalman Filter (MHTKF) which uses a bank of Kalman Filters to track several different hypothesis of the ball's position. Ball hypotheses are generated by the robot's perceptual system, from its own activities (such as kicking), and from teammate information. Because of the potential for erroneous ball information caused by poorly localized teammates, the robot's maintains two separate estimates of the ball based. The first is based on its own information and the second is an estimate returned by its teammates. When a behavior requests the position of the most likely ball, the model with the least uncertainty is chosen and that position and uncertainty estimate is returned to the behavior. The robot trusts its own ball estimates first and track them before using a hypothesis generated from its teammates.

Ball Velocity Estimation The World Model contains a method that uses linear regression on the past history of ball positions to estimate the current velocity of the ball. The primary use of ball trajectory estimation is to determine appropriate times for the robot to dive or block in order to capture the ball. The trajectory estimation is done by performing multiple linear regressions, with varying amounts of ball history being used for each regression. The chi-squared goodness-of-fit values of the linear regressions are used to select the best one. The velocity, intercept, and χ^2 values calculated by this regression are then used in the implementation of behaviors.

Opponent Robot Tracking Each visual observation of an opponent robot is stored for a maximum of 4 seconds after which time it is removed from consideration. Behaviors can query arbitrary rectangles in space for the existence of opponent robot observations. When a robot receives robot observations from its teammates, these observations are incorporated to the robot's own set of readings and are queried in the same fashion. Similarly, the observations are removed from consideration after four seconds.

Behaviors

Over the past few years, teams have experimented with different methods of team coordination. Many of these strategies involve attempting to keep teammates from running into each other as well as placing teammates in good locations on the field so that they can be in good positions to receive passes or go after a free ball. While there have been many good approaches, no one strategy seems to have emerged as being clearly superior to all others. One reason for this is that several different coordination strategies are likely to be applicable in a single situation. Since some strategies may work better than others, a team that selects the superior strategy will be at an advantage. Thus, one of the most important problems to address when designing a multi-robot soccer team is in selecting the kind of coordination strategy that they will use during the game. Teams may choose to use a fixed coordination strategy defined *a priori*, but if chosen poorly, such a fixed strategy may not fare well against the strategy of the other team. Thus, an important extension to the research problem of coordination strategies is the ability for a team to dynamically change their strategy at runtime to adapt to their opponents' strengths and weaknesses.

Dynamically selecting a different strategy depending on the situation can be very powerful technique, but can be very challenging to implement well. Robots that use a dynamic coordination system must be able to perceive and properly evaluate the state of the world as well as the state of their own progress. This information is vital when making the decision to switch from a poorly performing strategy to one that could potentially work better.

We have identified several different levels for dynamic coordination that can be applied to a robotic team. These include:

- A "first-order" approach, where the robots use a fixed coordination strategy and each robot modifies the parameters of its behavior according to the world state.
- A "second-order" approach, where the robots have multiple ways of handling different situations. In order to uti-

lize a second-order strategy, the robots must be able to evaluate the world state so that they can choose between the different behaviors they have at their disposal.

- A "third-order" approach, where the robots have several different team strategies, or "plays," which describe the coordinated actions of all of the robots together. Depending on the world state, different plays may apply; the team collectively decides upon the right behavior to apply in a given situation.

Changing Single Robot Parameters We define the first-order coordination strategy as the ability for the robots to set their own behavior based on the state of the world. In this kind of system, each robot is programmed with a single behavior set which is used to control the robot's behavior in its environment.

We have tried two different methods for representing first-order coordination strategies. The first is a potential fields approach and the other is an approach that we call constraint-based positioning. In previous work (Vail & Veloso 2003 in press), we give a detailed description of our implementation of potential field-based coordination. In this approach, we use potential fields both to determine the role that each robot plays (attacker, supporting attacker, and defender) and also to determine where the supporting robots position themselves on the field of play. On efficiency issue with potential fields occurs when they are used to coordinate the actions of a team of robots in a very dynamic world. In this situation, the fields may need to be recomputed for each every new sensor reading. This does not tend to be true for implementations of potential fields that are used for navigation in more static environments. In general, however, it's possible for minor disturbances in the positions or strengths of individual attraction and repulsion fields to cause fairly significant changes in the local gradient surrounding the robot.

Constraint-based positioning is an approach to robot positioning that we have developed in the CMPack'04 team for the 2004 RoboCup competition. Under this regime, robots are still assigned roles using a potential function, but the field positions chosen by the supporting robots are subject to a set of constraints. This approach was developed because there are several hard constraints that we would like to enforce on the robots' positions which are difficult to specify clearly with potential fields. For instance, defender robots need to avoid their own goalie's defense box, because entering the defense box is a violation which will cause the robot to be removed from play for 30 seconds. Other constraints that we would like to enforce include not crossing in front of a robot that is about to take a shot on goal, not coming within a certain minimum distance of a teammate, and so on. Consider a situation in which a robot is near the defense zone and a teammate is directly approaching it. Should the robot move toward the goal, violating the defense-zone constraint, or stand still, violating the teammate-distance constraint? Our implementation of constraint-based positioning allows us to prioritize the constraints, so that the robot knows that entering the defense zone is a more serious violation than coming near a teammate. In theory, the priorities of these constraints could be represented as a potential field, but we have found that debugging the complex potential fields that result can be difficult. If no constraints are in danger of being violated, the robot can choose to move to a specific point that is chosen based on the current state of the world. In this case, the robot can still use potential fields to choose an open area on the field or to choose a path to navigate around local obstacles.

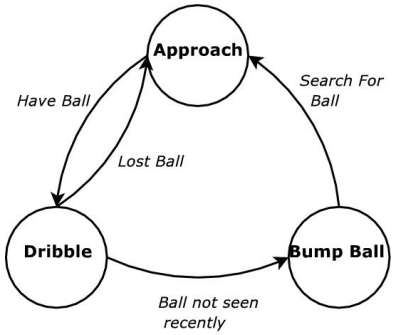

Figure 10: Finite state machine for the dribble behavior.

Our experience with RoboCup has been that a single positioning function defined for a particular role tends to be too limiting. Trying to capture all of the possible actions that a robot might accomplish can cause the complexity of the positioning function to grow beyond what is manageable. A soccer-playing robot might have multiple ways of approaching the goal, each of which has advantages depending on the relative position of the goalie and/or his other players. In some situations, the robot may want to try one approach and if it fails, try a different approach. Behaviors like these may be mutually exclusive and as such could be very difficult for a single function to capture. Experimental validation of these methods can be found in (McMillen, Rybski, & Veloso 2005).

Finite State Machine As in previous years, individual behaviors are written as finite state machines using a standard class. Each behavior is defined as a set of explicit states and state transitions and is able to monitor statistics such as the time spent in each state. This well formed framework leads to improved performance and debugging by providing methods to detect infinite loops, oscillation and other unwanted behaviors easily. Figure 10 shows an example state machine for a simple dribble behavior.

Python Interpreter The CMDASH'05 behavior subsystem has been completely replaced from previous years. In CMPack'04 and previous years, the behavior subsystem was written in C++. We make use of patches to Python 2.3.3 that were originally used by rUNSWift in the RoboCup 2004 competition. This year, the CMDASH'05 behavior system consists of the Python interpreter running directly on the robot and all of our behavior code has been completely rewritten in the Python language. Programming in python offers several important advantages.

- As an interpreted language, run-time errors do not crash

the robot (as C++-based behaviors would), but rather provide informative debug information which can be captured in a log or read directly over the wireless Ethernet.

- Specifying what code is running on the dog at any point can easily be accomplished by ftping the new Python files to the robot and restarting the interpreter remotely. This does not require rebooting the AIBO.
- The interpreter can be interacted with directly by using a remote python shell utility called *dogshell*. Parameters and variables can be changed dynamically without restarting the interpreter.

Odometry Calibration

New for this year is a simple but robust way for calibrating the robot's odometry. A specific walk's odometry parameters must be empirically determined in order to be used effectively by the behaviors. Our walk engine returns a theoretical maximum speed for the walk as well as odometric information. However, this assumes noise-free contact with the world which is impossible to achieve. Our odometry correction process consists of two steps: (1) computing the maximum speed of the walk, and (2) computing any correction factors for the surface on which the robot walks.

To calculate the maximum speed, the robot runs a simple warmup behavior which moves it in specific directions for a set amount of time at the maximum speed possible for that walk. The distance that the robot travels in that time is used to compute the true maximum speed. The same process is repeated for turning.

To calculate the odometric corrections, the robot is told to translate and specific distance as well as rotate about a specific angle. If the robot overshoots, then a correction factor is computed that allows it to better predict the distance that it travels or the angle that it turns.

Logging and Debug GUI

Capturing sensor logs for analysis is an important part of the AIBO research process. The CMDASH'05 team is capable of logging a wide variety of sensor information and inferred state information. Some of the more important information includes:

- raw vision video frames
- color segmented video frames
- inferred positions and confidences of objects identified by high-level vision (ball, markers, goals, other robots)
- estimated positions of objects based on integrating multiple video frames (i.e. world model estimate of the ball position)
- robot's own position based on the MCL localization
- general debug information

This information can be stored in a logfile that is saved to the memory stick after the robot's operation, as well as transmitted over the wireless Ethernet during operation. Our graphical visualization program, called *chokechain*, allows someone to view this data in real-time directly from the robot, or to step forward and backward through it if captured to a logfile. Figure 11 shows a sample view of the interface.

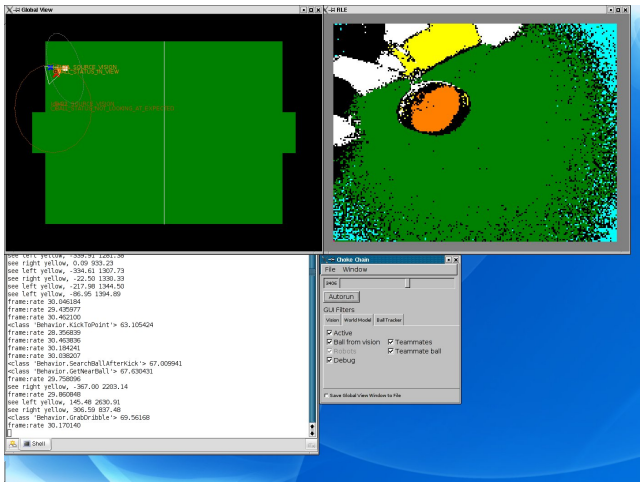

Figure 11: The chokechain GUI illustrating a top-down world model view of the world in the upper left, the color-segmented image on the upper right, debug information shown in the terminal window in the lower left, and the window control panel with information filters in the lower right.

Calibrated Overhead Camera for Ground Truth Obtaining ground truth of the robot's position is extremely important for validating the accuracy of our algorithms. In particular, having the ability to compare the robot's estimated location on the field and its estimate of the ball to the real world is very useful. In order to facilitate this, we have mounted a camera over our field to obtain an overhead view of the robots. Radial distortion of the image caused by our camera's wide field of view is corrected using a Matlab camera calibration toolbox (Bouguet). Once the intrinsic parameters of the camera are obtained, the position of the field in the camera images is obtained by having the user click on the screen to identify the outer boundaries of the field. Once aligned, any world model information generated by the robot can be overlayed on the corrected camera image in order to compare it against ground truth. Figure 12 shows the global view in *chokechain* but using the overlay on the live video feed.

Conclusion

We are very interested in teaching Artificial Intelligence concepts within the context of creating a complete intelligent robot. We believe that programming robots to be embedded in real tasks illustrates some of the most important concepts in Artificial Intelligence and Robotics, namely sensing uncertainty, reactive and deliberative behaviors, and real-time communication and motion.

With CMDASH'05 , 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

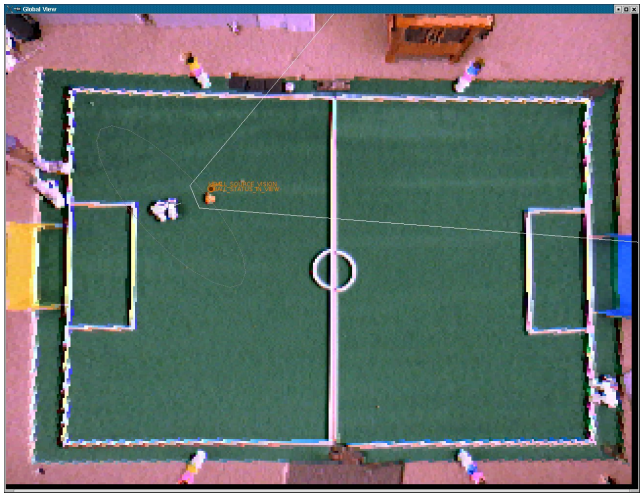

Figure 12: Overhead camera view with world state information superimposed on it. This image shows the robot's position with uncertainty, the field of view of the camera, and an estimate of the ball's position in the image.

AIBO software with new visual features, a Python interpreter for behaviors with new behavior semantics, more extensive modeling and estimation of the world, better odometry calibration, and more advanced debugging and visualization tools.

We seek to find a good balance between the theoretical aspects of the in-class lectures and the hands-on labwork. We feel very strongly that both are required to achieve a well-rounded learning experience. As of the 2004 class the students did not have any in-class midterm or final exam in lieu of a more in-depth final project, we have decided to include them the next time the course is taught. There are too many concepts to fit into a cumulative final project. A more traditional exam schedule will fill this gap.

It is a testament to the AIBO's capabilities that the course explores too many AI topics to capture in a single final project. Indeed, we are seeking to better integrate topics such as machine learning and (perhaps there are others) in future offerings. We have found that, with the AIBO's hardware and software resources, undergraduate students can efficiently and effectively investigate the broad and growing connections among AI robotics topics.

Acknowledgements

We would like to thank Sony for their remarkable support of our research, specifically by making the AIBO robots accessible to us since their first conception in 1997. Sony has continued their support through these years, and is very interested in following the impact of an AIBO-based course. We would like to thank all of the students who have helped develop the AIBO soccer codebase, including Scott Lenser, James Bruce, Doug Vail, Sonia Chernova, Colin McMillen, Juan Fasola, and Sabine Hauert. Thanks also to Dr. Felix von Hundelshausen. We would also like to thank the Carnegie Mellon Computer Science Department for approving this new course in the Fall of 2003 and providing the lab facilities.

References

Bouguet, J. Y. Camera calibration toolbox for matlab. Website: http://www.vision.caltech.edu/bouguetj/calib_doc/.

Bruce, J.; Balch, T.; and Veloso, M. CMVision, www.cs.cmu.edu/~jbruce/cmvision/.

Bruce, J.; Balch, T.; and Veloso, M. 2000. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of IROS-2000*.

Bruce, J.; Lenser, S.; and Veloso, M. 2002. Fast parametric transitions for smooth quadrupedal motion. In Birk, A.; Coradeschi, S.; and Tadokoro, S., eds., *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag.

Lenser, S.; Bruce, J.; and Veloso, M. 2001a. CMPack'00. In Stone, P.; Balch, T.; and Kraetzschmar, G., eds., *RoboCup-2000: Robot Soccer World Cup IV*. Berlin: Springer Verlag. 623–626.

Lenser, S.; Bruce, J.; and Veloso, M. 2001b. CMPack: A complete software system for autonomous legged soccer robots. In *Proceedings of the Fifth International Conference on Autonomous Agents*. Best Paper Award in the Software Prototypes Track, Honorary Mention.

McMillen, C.; Rybski, P.; and Veloso, M. 2005. Levels of multi-robot coordination for dynamic environments. In *Multi-Robot Systems Workshop*.

Uther, W.; Lenser, S.; Bruce, J.; Hock, M.; and Veloso, M. 2002. CM-Pack'01: Fast legged robot walking, robust localization, and team behaviors. In Birk, A.; Coradeschi, S.; and Tadokoro, S., eds., *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Berlin: Springer Verlag.

Vail, D., and Veloso, M. 2003, in press. Dynamic multi-robot coordination. In *Multi-Robot Systems*. Kluwer.

Veloso, M., and Uther, W. 1999. The CMTrio-98 Sony legged robot team. In Asada, M., and Kitano, H., eds., *RoboCup-98: Robot Soccer World Cup II*. Berlin: Springer Verlag. 491–497.

Veloso, M.; Lenser, S.; Winner, E.; and Bruce, J. 2000. CM-Trio-99. In Veloso, M.; Pagello, E.; and Kitano, H., eds., *RoboCup-99: Robot Soccer World Cup III*. Berlin: Springer Verlag. 766–769.

von Hundelshausen, F., and Rojas, R. 2003. Tracking regions. In *Proceedings of the RoboCup 2003 International Symposium, Padova, Italy*.

von Hundelshausen, F. 2004. *Computer Vision for Autonomous Mobile Robots*. Ph.D. Dissertation, Department of Computer Science, Free University of Berlin, Takustr. 9, 14195 Berlin, Germany. http://www.diss.fu-berlin.de/2004/243/indexe.html.