CMPack: A Complete Software System for Autonomous Legged Soccer Robots*

Scott Lenser James Bruce Manuela Veloso

Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213

{slenser,jbruce,mmv}@cs.cmu.edu

Keywords

autonomous robots, lessons learned from deployed agents, multi-agent teams, action selection and planning, real-time performance

ABSTRACT

This paper describes a completely implemented, fully autonomous software system for soccer playing quadruped robots. The system includes real-time color vision, probabilistic localization, quadruped locomotion/motion, and a hierarchical behavior system. Each component was based on well tested algorithms and approaches from other domains. Our design exposed strengths and weaknesses in each component, and led to improvements and extensions that made them more capable in general, as well as better suited for our testing domain. Integrating the components revealed design assumptions that were violated. We describe the problems that arose and how we addressed them.

The integrated system was then used at the annual RoboCup robotic soccer competition where we placed third, losing only a single game. We reflect on how our system addressed its goals and what was learned through implementation and testing on real robots.

1. INTRODUCTION

Developing systems of autonomous robots to address complex tasks is particularly challenging. An autonomous robot needs to perceive its environment, make decisions about selection of its actions, and finally concretely carry its actions ahead. In this paper we report on our research developing autonomous quadruped robots capable of being part of a robotic soccer team. Through several years of working with the Sony quadruped robots, we have gathered a coherent awareness of the complexity of this task, as well as a solid understanding of the approaches to integrate perception, behavior, and actions into effective intelligent autonomous agents.

This paper first briefly overviews the specific RoboCup Sony legged robot league environment [6]. We present our work with several main components of relevance to an autonomous quadruped agent. Perception is gathered through an on-board camera and color thresholding is performed at the hardware level. We have previously developed a vision processing system, CMVision, that learns the necessary thresholds for the color space offline [3]. It provides a color segmented representation to the object recognition system in real time at video rates. Objects to be recognized have varying geometry and may have multiple colors. We present our robust object recognition system developed for RoboCup, and briefly describe the CMVision system on which it builds.

The robots perform their task in a field that includes color-based fixed landmarks. The robot localizes itself in the world using a probabilistic sampling method that updates its locale belief based on its movements and on the perceived objects. We initially developed a probabilistic localization algorithm, Sensor Resetting Localization, [9], that robustly accommodates the poorly modeled movements of the quadruped robot and large errors due to externally caused movements of the robots, such as pushing from other robots, and repositioning done by the human referees during the game. Sensor Resetting Localization is very effective. In this paper, we identify additional challenges that we could face in terms of localization building upon the sensor resetting approach.

The robots need to accomplish the specific task of playing soccer. The task involves three robots in each team. Teamwork is currently addressed as a role assignment, namely one robot is the goalie and the other two robots are attackers. Developing robust behaviors for the robots represents a challenge. We contribute a cognition architecture in which sensors, behaviors, and control are separated in three distinct hierarchies. This architecture supports well the eventual upgrade of sensor processing and the refinement of behaviors and control.

Finally, we address the motion opportunities offered by the quadruped robot. In our previous work with these Sony

*This research was sponsored by Grants Nos. DABT63-99-1-0013, F30602-98-2-0135 and F30602-97-2-0250. The information in this publication does not necessarily reflect the position of the funding agencies and no official endorsement should be inferred.

AGENTS'01, May 28-June 1, 2001, Montréal, Quebec, Canada.

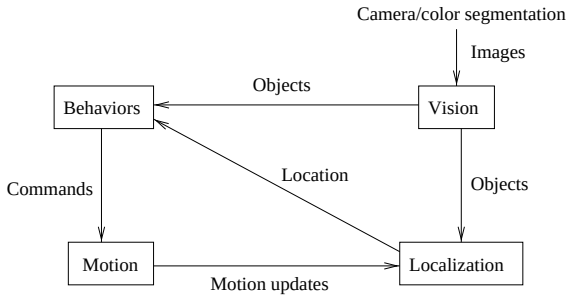

Figure 1: The overview of the main components of our robotic agents.

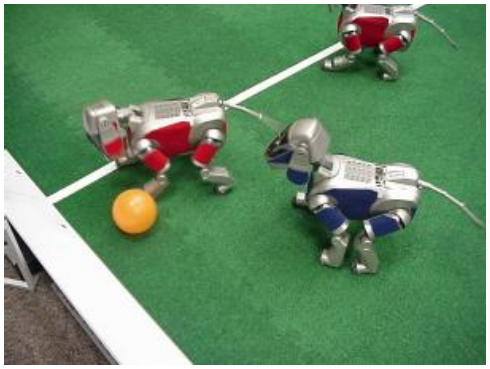

Figure 2: The Sony quadruped robot dogs playing soccer.

robots, we mainly used a walking routine supplied by Sony. In this paper, we report on our development of new motion primitives, including also new kicking movements using the legs and the head. The different kicking modes integrated with the behaviors allowed for a very effective manipulation of the ball.

Figure 1 summarizes the main components that we present in this paper. Section 2 briefly describes the task environment, namely the playing field and the game. Section 3 presents the vision processing for effective object detection. Section 4 discusses the localization approach. Section 5 introduces the behavior architecture. Section 6 discusses the motion primitives.

Section 7 concludes the paper. It reviews the contributions of the work as a fully developed and implemented robotic agent system. We believe that our extensive work on this research task contributes to and advances the area of autonomous robotic agents.

Inevitably and excitingly, there remain many challenges open to our future research. Let us remark that each attacking robot is aware of the task at a basic level, namely the robot searches for the ball, localizes itself on the field, and tries to score into the opponent goal. Collaboration between robots, detection of and response to adversarial robots, and further intelligent awareness of the complete game are challenging topics of our ongoing and future research.

2. ENVIRONMENT

The robots used in this research were generously provided by Sony [6] to be applied to the specific domain of robotic soccer. The robots are the similar to the commercial AIBO robots, but they are provided to us with slightly different hardware and programmable. The robot consists of a quadruped designed to look like a small dog. The robot is approximately 30cm long and 30cm tall including the head. The neck and four legs each have 3 degrees of freedom. The neck can pan almost 90° to each side, allowing the robot to scan around the field for markers. The camera has about a 55° field of view. Figure 2 shows a picture of three dogs playing soccer.

All teams in the RoboCup-00 legged robot league used this same hardware platform. The robots are autonomous, and have onboard cameras. The onboard processor provides image processing, localization and control. The robots are not remotely controlled in any way, and as of now, no communication is possible in this multi-robot system. The only state information available for decision making comes from the robot's onboard colored vision camera and from sensors which report on the state of the robot's body.

The soccer game consists of two ten-minute halves, each begun with a kickoff. Each team consists of three robots. Like our team last year, CMTrio-99 [11], and most of the other eleven RoboCup-00 teams, we addressed the multi-robot aspect of this domain by assigning different behaviors to the robots, namely two attackers and one goaltender. No communication is available and the robots can only see each other through the color of their uniforms. No robot identity can be extracted. As of now, our robot behaviors capture the team aspect of the domain through the different roles.

The acting world for these robots is a playing field of 280cm in length and 180cm in width. The goals are centered on either end of the field, and are each 60cm wide and 30cm tall. Six unique colored landmarks are placed around the edges of the field (one at each corner, and one on each side of the halfway line) to help the robots localize themselves on the field. Figure 3 shows a sketch of the playing field.

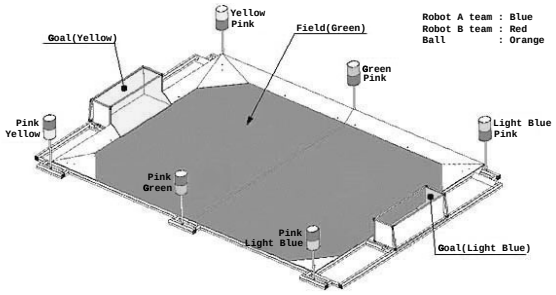

Figure 3: The playing field for the RoboCup-00 Sony legged robot league.

3. VISION

The vision system is responsible for interpreting data from the robots' primary sensor, a camera mounted in the nose. The images are digitized in the YUV color space [10], and color thresholds that were learned offline are then applied to the image in hardware. The low level vision software then carries out the following steps:

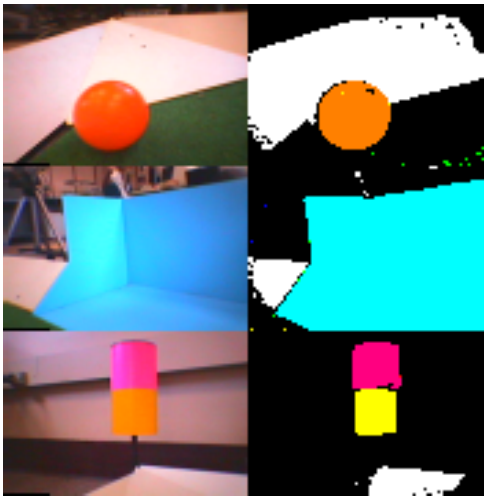

Figure 4: Sample images from the robots' camera, and those images after color thresholding

- Connect like-colored neighboring pixels to make regions.
- Connect like-colored nearby regions into larger regions.
- Calculate statistics for the regions useful for high level vision.

Here like colored refers to being a member of the same color threshold class (such as orange/ball, or blue/marker), and nearby regions are evaluated using a density threshold. If two regions when merged have a density occupying their bounding area above a threshold for that color class, they are considered as one region. After the low level processing is performed, the high level vision carries out the following steps for each type of object of interest (currently these include the ball, goals, markers, and other robots).

- Scan lists of regions for colors that include the object of interest.
- For each region or regions that may form the object, evaluate domain and geometric constraints to generate a confidence value.
- Perform additional filtering rules to remove false positives.
- Take the hypothesis with the highest remaining confidence value.
- Transform the object from image coordinates to egocentric body coordinates.

The low level vision system uses the CMVision library. It maps pixels into colored regions in several steps using efficient algorithms and highly optimized implementations. The first step is to run-length encode the image, replacing individual pixels with a value and a horizontal length indicating the number of pixels following the first with the same value. This allows a compressed representation that leads to much higher efficiency for later processing. The next step takes the runs and performs connected components merging using four-connectedness. This is implemented as a tree-based union find with path compression. The output is "regions" formed by pixels that were classified as the same color as one of their neighboring pixels.

An additional run over the run-length encoded representation gathers statistics about the regions, including their centroid, bounding box, and pixel area. Finally, the regions are separated by their color and sorted by size, since larger regions are likely to be more salient and the smallest are likely the result of noise. The underlying system is described in more detail in [3], and the library is freely available under the GPL [2].

The high level vision system uses the regions determined by the low level system to detect the objects of interest. Each type of object employs specific detection and processing to find it in the region data, eliminate false positive identifications, and determine its distance from the robot. The processing strategies for each object are as follows:

- *Ball:* The ball is an orange spherical object, thus it can be detected as an orange circular region on the image plane. Large orange regions are processed, factoring their area, the compactness in their bounding box, and the similarity of the width and height of their bounding boxes (indicators of how nearly circular the object is). Objects that appear significantly above the ground plane, or are only a few pixels, are discarded as likely false positives. Distance is determined by similar triangles: the known radius of the ball divided by the distance is equal to the pixel radius on the image divided by the focal distance.
- *Goals:* Goals are detected as large yellow or cyan objects on the image whose regions are near the bottom of the image or when projected into world space are near the ground plane. The region lists for the two possible colors are scanned and confidence values are assigned based on how much the bounding box and area match the possible image geometry given the possible viewpoints on the field. The distance is determined by triangulation based on the height of the goal. The width is not used because it is much more likely to be partially occluded by a robot.
- *Robots:* Robots are detected as red or blue patches (the team colors) that are not significantly above the plane of the robot. Currently, no attempt is made to match which of the color patches is being observed, since each robot has multiple patches in varying amounts of occlusion (see Figure 2). Thus we currently make no orientation or distance estimates, since these are difficult to determine concisely or efficiently.
- *Markers:* Markers consist of a two colored cylinder, with each color occupying a 10cm band on a 10cm diameter cylinder. The resulting image on the image plane is two similarly sized colored regions that are approximately square. Each marker is unique, consisting of a pink band, and a yellow, green, or cyan band above or below the pink one. The markers are found by taking each member of the list pink regions, and then scanning the yellow, green, and blue region lists for pairs which fit the following criteria: The square of

the distance between the centroids of the two regions should be equal to the area of each of those regions. Using a sigmoidal falloff from this ideal model, we can quickly evaluate partial matches and take the best hypothesis. Distance is calculated by triangulation of the distance between the centroids of the two colored regions.

Finally, the ego-centric body coordinates are obtained by mapping the coordinates in the image plane of the camera to the robot's coordinate frame using standard transformation techniques to account for head tilt, pan, and roll, as well as body attitude relative to the ground plane [8].

We found our vision system to be generally robust to noise and highly accurate in object detection and determining object locations at the RoboCup competition. However, like many vision systems it remains sensitive to lighting conditions, and requires a fair amount of time and effort to calibrate. Future work includes further automation of offline threshold calibration using unsupervised learning, and migrating more low and high level calibration parameters to online adaptive systems.

4. LOCALIZATION

Our localization system, Sensor Resetting Localization [9], uses a probabilistic model of the world to estimate the robots location on the field. The robots location is represented as a probability density over the possible positions and orientations, hereafter locations, of the robots. Since the probability density is in general a very complex function, we approximate the probability density by a set of sample points. The samples are chosen such that if $x\%$ of the samples are expected to be found in a particular area then the probability that the robot is in that area is $x\%$. Each sample point represents a particular location on the field at which the robot might be located. Localization is the process of updating this probability density. To make the computation tractable, we make the Markov assumption that the robots future location depends only on its present location, the motions executed, and the sensor readings. Updates are done in such a way that the expected density of sample points in a region is proportional to the probability of the robot's location being within that region. For more detail including a probabilistic derivation of the algorithm, see the Sensor Resetting Localization paper [9].

The sample locations are represented as an array of n locations on the soccer field. High sample counts tend to result in better accuracy and robustness at the cost of additional computation. We used $n = 400$ in our system as a reasonable compromise. There are three basic types of updates we perform on the sample set: movement updates, sensor updates, and sensor based resetting. The basic update cycle is: move robot, update for movement, read sensors, update for sensors, reset if necessary, repeat. The mean of the samples is taken to be the best estimate for the location of the robot and the standard deviation of the samples is used to estimate the uncertainty.

The goal of the movement update is to convolve the robot's belief state with the probabilistic model of the robots movements. This goal is accomplished by taking each location in the sample set and randomly choosing a new location according to the probabilistic movement model. This new location is a possible final location of the robot after executing the movement from the sample location. Note that this operation spreads out samples that are overlapping since each sample may choose a different location to end up in.

The goal of the sensor update is to multiply the robot's belief state by the probability of the locations given the sensor readings. Updates for sensors are done in a two step process. The first step weights all the samples according to the probability of being in a location given the sensor readings. The second step renormalizes the samples to uniform weight. This normalization is done by creating a new sample set. The new sample set is generated by drawing samples with replacement from the old sample set with probability proportional to their weights. This has the effect of copying some samples and deleting others. Note that this step never creates new samples. In our case, distances and angles to landmarks serve as our sensor readings.

The goal of sensor based resetting is to return the probability density to a sane state whenever it appears that the algorithm may be failing. Here, a sane state means one that reflects all likely locations of the robot. The algorithm may fail for a number of reasons: inaccurate models of movement and sensor noise, external disturbance of the robot's location, or insufficient sample count to accurately approximate the probability density. We restore the probability density to a sane state by replacing the density with one consistent with the robot's current sensor readings and the robot's prior belief of likely locations to be located on the field (we use a uniform distribution for this). We apply this resetting step any time we find that our belief state is inconsistent with the current sensor readings. Inconsistency can be measured by using the sensor probability model to estimate the likelihood of the sensor readings given our current belief state. See Figure 5(right) for an example of the result of a reset event. Without this step, the algorithm is the same as the popular Monte Carlo Localization technique [5]. This step was added to solve a problem recovering our position from an initially unknown location (see Figure 5 left). With only 400 samples (to keep computational load low) there is only 1 sample for every 100cm^2 ignoring orientation for the moment. Usually only 1–4 of the 400 samples are consistent with a single landmark observation but there are infinitely many locations that are consistent with this observation (see Figure 5 middle). Resetting solves this problem by concentrating the samples in locations that are consistent with the landmark observation (Figure 5 right). Adding sensor based resetting also solved several other problems. By resetting, we automatically recover from errors in the movement model and external disturbances. External disturbances are caused by the referee moving the robot or collisions with other robots.

Although this algorithm is highly effective, there are still additional challenges to address. A limitation of the system is that the localization technique requires probabilistic models of the robot's motions and sensors. A good probabilistic model of motions requires excellent calibration. Developing a probabilistic model for the sensors is even more problematic. Besides calibration, the sensor model must be such that field locations consistent with the sensor reading can be found in an efficient manner to allow for resetting. This can be difficult to do for sensor readings that entail complex distributions. We are looking into solutions to these difficult research problems.

We made the following observations and enhancements in

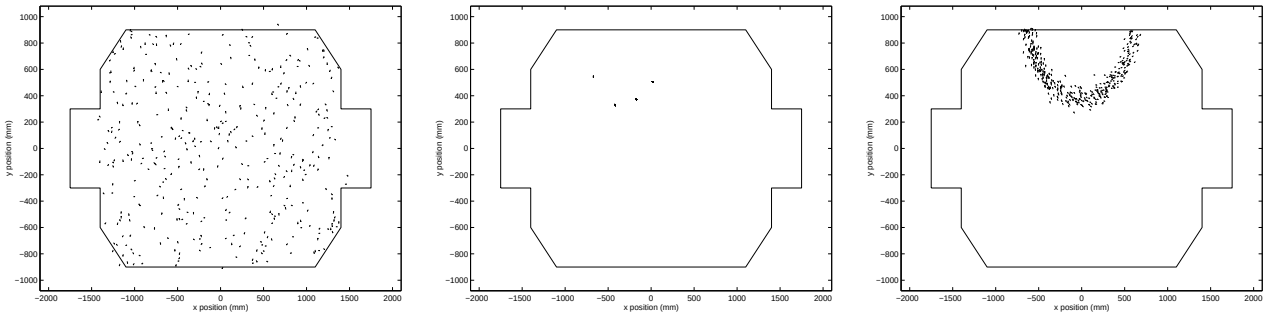

Figure 5: The sequence of belief states resulting from a robot in an unknown initial location seeing one landmark.

our work with the robots:

- Localization must be robust to external disturbances.
- Localization must be fault tolerant.
- Localization should incorporate sensor reading that are hard to describe analytically.
- Sensor based resetting allows for poorer approximations and models to be used.
- Sensor based resetting provides fault tolerance and failure recovery in the robot localization.
- Sensor Resetting Localization is robust to external disturbances.

5. BEHAVIOR SYSTEM

Our behavior system is a hierarchical behavior-based system. The system is primarily reactive, but some behaviors have internal state to enable sequencing of behaviors. The input to the system is information about the objects seen (from the vision) and an estimate of the robots location (from the localization). The output of the behaviors is a motion to be executed (see Figure 1). The motion can be a type of kick to execute (discrete) or a direction to walk (continuous) for example. The behavior system consists of three interconnected hierarchies for sensors, behaviors, and control (see Figure 6). The sensor hierarchy represents all that is known about the world. The behavior hierarchy makes all of the robot's choices. The control hierarchy encodes everything the robot can do. Our system is similar to the system used by the FU-Fighters small size RoboCup team [1].

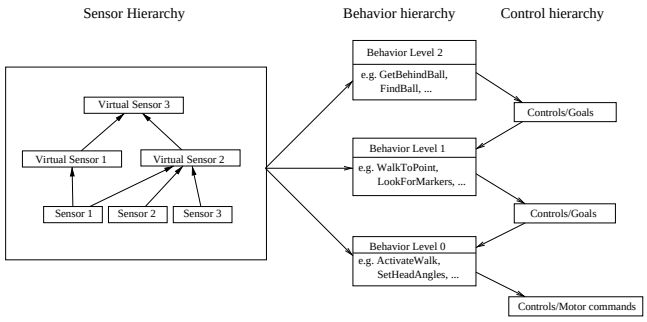

Figure 6: Overview of the behavior system.

5.1 Sensors, behaviors, and control

The sensor hierarchy represents the knowledge that the robot has about the world. We divide the sensors into two categories, real sensors and virtual sensors. Virtual sensors represent a processed version of the real sensors that includes information that the behaviors are directly interested in. For example, a real sensor is the position of the ball reported by the vision, a virtual sensor is the estimated position of the ball including when it is not directly seen. The virtual sensors serve to aggregate information and act as a model of the world complete with memory. Virtual sensors also avoid computing the same information twice in separate behaviors. The sensor hierarchy has a data structure for storing the sensor values and code to update the virtual sensors in the data structure from the real sensors. The data structure output by the sensor hierarchy provides the input to the behavior hierarchy.

The behavior hierarchy and control hierarchies are tightly coupled and cooperate to decide on the action to perform. The behavior hierarchy consists of n behavior sets. Each behavior set represents a set of behaviors. Different behavior sets operate at different levels of detail. The control hierarchy consists of n control sets. Each control set is a data structure representing all the actions the robot could perform. Each control set represents the actions at different levels of detail. A control set can be viewed as a set of goals for low level behaviors to achieve. Alternatively, it can be viewed as a virtual actuator that the high level behaviors can control. A behavior set at level k receives input from the the control set at level $k+1$ and the sensor hierarchy. The behavior set decides what action to perform and writes the decision into the more detailed control set at level k. For example, a behavior set might contain a behavior for getting behind the ball. This behavior could be activated when the control set above this level indicates that it is good to get behind the ball. The behavior would then use the sensors to find out where the ball is and set a goal for the next level down to run along an arc that intersects a point behind the ball. Each behavior set takes an abstract description of the task to be performed and creates a more concrete description of actions to be performed. The control hierarchy provides storage for each level of the behavior hierarchy to write outputs and read inputs. The lowest level of the control hierarchy, level 0, is simply the set of low level actuators available to the robot, in our case the commands exported from the motion module.

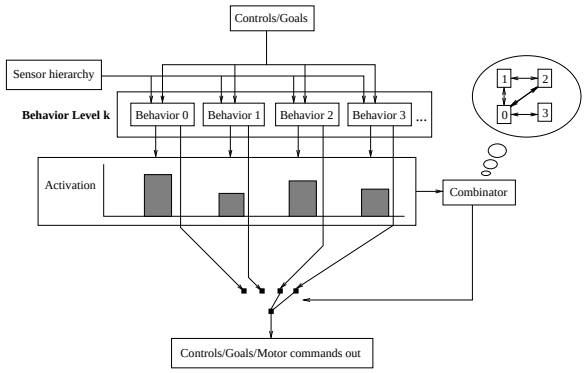

Figure 7: Detail of a level in the behavior hierarchy.

5.2 Behavior selection and coordination

We can look in more detail at the behavior sets which comprise the levels of the behavior hierarchy (see Figure 7). Each behavior set takes as input the controls/goals from the control set of the level above it. Running the behavior set produces the control set that will be used by the behavior set at the next lower level. If this is the lowest level, then the control set produced is sent directly to the motion module. How is this accomplished? Each behavior set has a set of behaviors that all operate at the same level of detail. Behaviors have a functions for calculating activation levels and outputs given the sensors and high level controls. These functions are part of the domain knowledge given to the robot. Each behavior looks at the sensors and the goals and decides how good it would perform. This measure of goodness is used as an activation value for the behavior (see middle of Figure 7). Note that these activations can also be thought of as predictions of the future reward that will result if this behavior is run. Each behavior drives a set of low level actuators/controls. Some behaviors within the behavior set will drive the same actuator, and thus conflict, while some will drive different actuators and can be run in parallel. This occurs frequently in our domain as we may want to walk somewhere while doing something useful with the head like scanning for the ball. Behaviors that are in conflict must never be run together. We capture this constraint by constructing a graph where behaviors are nodes and edges connect behaviors that conflict (see upper right part of Figure 7). We use a special combinator (described below) to choose a set of non-conflicting behaviors with near maximal total expected reward. So for the example activations in the figure, the combinator has chosen behaviors 2 and 3 since this has more reward than executing behavior 0 alone. The behaviors that are chosen for activation are then run and write there results directly into the control set for the next level of behaviors. The behaviors that are run use the sensors, the control set from above, and their memory of what they were doing to choose the controls to write into the goals of the next lower level.

The goal of the combinator is to find a set of non-conflicting behaviors that result in maximal reward. Since the reward estimates(activations) and conflict net is given, this is the problem of finding maximal weight cliques in the dual of the conflict graph. Since this problem is NP-complete, we use an approximation algorithm. The basic idea is to find a suitably good approximation iteratively by suppressing weakly activated behaviors with many conflicts and reinforcing behaviors with few conflicts. We first produce an optimistic estimate for the total reward that can be achieved while running behavior k by assuming that all behaviors that are not in direct conflict with behavior k can be run in parallel. This is calculated by finding the total activation of all behaviors and subtracting the activation of all behaviors in direct conflict. This estimate is then used as a gradient to change the activation of the k^{th} behavior. Behaviors that might be runnable with more reward than the behaviors they conflict with are reinforced while other behaviors are suppressed. Usually, at least one of the behaviors will have a negative gradient. We follow this gradient until the activation of one of the behaviors becomes 0. Any behavior whose activation becomes 0 is removed from consideration. This process is repeated until the set of behaviors with non-zero activation contains no conflicts. Small random perturbations are added to the activations to break any ties. In case all the gradients are positive, we double the amount subtracted for conflicts until one of the gradients becomes negative. In the worst case, it may take $O(n \lg n)$ steps for the combinator to converge (where there are n behaviors) but for most cases it converges in a few iterations. The set of behaviors with non-zero activation at the end of this process are run completing the execution of the behavior set.

By making each behavior responsible for its own activation, we keep the only interaction between behavior levels through the control sets. This allows us to change one level of behaviors without having to change any other levels of behaviors. For example, we could replace a behavior for getting behind the ball with two different behaviors without changing the layer that chose to get behind the ball. By allowing our behaviors to have some memory, individual behaviors can sequence actions. We use this when searching for the ball to allow us to remember where on the field we decided to go look for the ball at. Memory also allows us to add hysteresis to our behaviors where needed.

We made the following observations/enhancements in implementing this system:

- Oscillation is a major problem for behavior based systems due to lack of accounting of future costs. We believe this can be solved through activations that accurately reflect future reward (perhaps using reinforcement learning?).
- Behavior based systems perform better with primitives parameterized to form a continuous system. This is largely due to the difficulty of sequencing behaviors.
- Complexity can be reduced by making hierarchical levels that are more separate. Separation can be achieved by making behaviors responsible for their own activations.
- Providing behaviors with some limited memory allows for sequencing and oscillation removal through hysteresis.

6. MOTIONS

The motion component allows the robot to walk, kick the ball, and get up after falling. The desired action is chosen by the behavior system, and the motion system determines

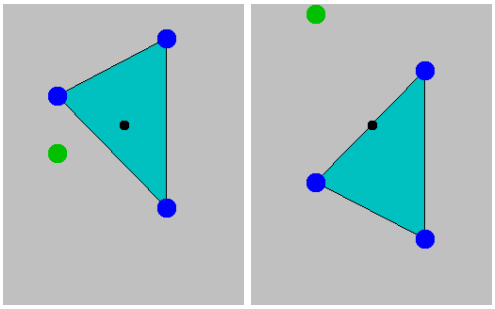

Figure 8: Visualization of walk at two points during a forward walk cycle. The front of the robot is at the top, and the dots represent foot positions projected down onto the ground plane. The basis of support is indicated as a shaded polygon defined by the feet on the ground. The left figure is at a point right before the rear left foot is to touch down, and the second is at a "critical transition point," where the front foot is to about to touch down while center of mass crosses the basis of support.

the required angles and PID values for each of the joints to carry out the command. A successful walking motion must allow the robot to travel on a specified path, while maintaining stability in the process. Also, the walking motion must transition smoothly to and from other motions. One way to look at the stability of the robot while walking is to consider the robot's center of mass and basis of support (Figure 6). The basis of support is the area on the ground within the polygon formed by the points of contact (in our case either a triangle or a quadrilateral). The points of contact are the locations where the feet are in stable contact with the ground. In the absence of momentum, the robot can stand in any position that results in the center of gravity of the robot (when projected onto the ground plane) being within the basis of support. The robot will fall from any other position. Our walk uses a quasi-static crawl gait. A crawl is a gait in which at least 3 feet are on the ground at all times. Quasi-static refers to the type of stability of the system, and means that the center of gravity of the robot is always within or on the edge of the basis of support. It is called quasi-static because with actuation and environment noise the center of gravity of the robot may temporarily be outside the basis of support. Our particular walk is based loosely on the walk used by the RoboCup team from University of Paris VI [7]. We use the same foot ordering and similar techniques to assure stability when the center of gravity of the robot is over the basis of support. However, our implementation differs in that it uses higher level geometric primitive and calculations to better address the problem of smooth transitions and composable target motions. The first requires that transitioning from motion along one path to another should be smooth (such as arcing to the left and to the right without a abrupt change in velocity), and composable motions are the ability to sum primitive motions into a more complicated path (such as turning while walking forward, or walking sideways while turning).

Several key observations led to our approach to implementing the walk. The first is that in a stable crawl, the body should be kept at nearly a fixed attitude, to prevent unwanted jolting or oscillation on other parts of the robot, such as the head. The second is that a specific path can then be constrained entirely by the motion of the body, and that when feet are on the ground their motion is fully constrained by the motion of the body during that time. Then the main parameters left open are the following:

- what path the robot should follow
- the order of lifting the feet
- the path the feet should follow in the air (air path)
- the target point the foot should move to while in the air (air target)
- fraction of a total walk cycle the foot should be in the air (air fraction)
- the time period of a walk cycle

Additional parameters, such as the body attitude, and the "neutral" or origin position of the legs relative to the body were determined and fixed in our implementation to maximize stability, while the parameters above were determined online through calculations. First, the path followed by the robot was taken as a high level command from the behavior system, and consisted of a velocity request for x, y, and θ. These are translated into a cycle period, body path, and air cycle using linear interpolation. The fastest walk had a forward speed of 120mm per cycle, a cycle time of 1200ms, and an air fraction of 0.25, and the slowest had a velocity of 0, and cycle of 1500ms, and an air fraction of 0. Thus our behaviors could select any speed depending on what the situation required. Additionally, the walk also scaled these parameters based on requested sideways motion and angular velocity, leading to speed adaptive walking for these also.

The path of the body was then determined by the velocity request, and modeled using a Hermite spline [4]. These splines take the initial and target points as parameters, as well as derivatives at each, thus allowing smooth trajectories through points at a specified speed and direction. The calculated world coordinates were then use to perform inverse kinematics to determine the required joint angles. Using the splines we also implemented continuous transitioning from one path to another. Four times each cycle (once every time a foot was picked up), the motion system would take the most recent target walk request from the behavior system. It would then determine the current position and velocity of the body (which is made simple by using the splines), and plot a new spline path from the current position and velocity to the target position and velocity one cycle later. Thus we had fully continuous body paths and velocity regardless of the sequence of paths requested by the behaviors, and latency of only a single step before beginning to execute a command. Traditional approaches require a whole cycle before transitioning to another path, and may require special properties on leg position or velocity to allow continuity and smooth motion. The spline based motion guarantees continuity without restrictive assumptions.

The order of the feet was determined based on the dominant requirement of the motion target specified by the behaviors. Turning, and walking have different optimal foot patterns for stability. For turning, a clockwise or counterclockwise pattern that is the same as the direction of motion is most stable, while for forward motion the most stable order is to lift a back foot followed by the front foot on the

same side, and then the same for the other side. Changing the order of the feet requires the robot to stop and transition through a standing state in our model, in order to guarantee stability and continuity, and keep the implementation of different walks modular.

The air path and air target must be chosen to keep the robot stable and keep the foot position within its reachability space during the entire walk cycle. The air target was chosen so that the average position of the foot during the ground fraction of the cycle would have an average position approximately at the neural position. This can be calculated by evaluating the future velocity at the leg's neutral position, and setting the air target as the sum of the neutral location and half of the velocity. In reality, the walk parameters may change the next time the foot is on the ground, but this simple predictive approximation keeps the foot nearly where it should be without an expensive calculation or knowledge of future walk commands. The air path was a Catmull-Rom spline (multiple segment Hermite spline), with the interpolated points being the current location, the air target, and an intermediate point averaging the two but elevated in z so that the foot will be off of the ground. The velocity at the center point is determined as the average velocity needed to get the foot to the target, and the initial and end points have the velocity of the ground plane but with a discontinuous z velocity component in order to quickly make or break traction with the ground.

The high level motion system consisted of walks with separate stepping patterns (walking forward, turning left, turning right), several kicks used to manipulate the ball with the body, and four routines for the robot to get up when it falls down, each of which was tailored to recovering from a fall on a different side of the body (front, back, left side, right side). The motion system was constructed as a state machine, with all states transitioning to or from standing at neutral, except for the fall recovery routines, which passed through an intermediate state of lying down on the robot's belly. The non-walking motions were specified as time variant functions to determine raw joint angles and kinematic targets for the legs.

Overall we found the walk to perform significantly better than other implementations of the same gait at the competition, as well as allowing for a simplified software architecture thanks to high level primitives such as splines. We demonstrated the fastest and most stable walk using the crawling gait. Future goals include adapting our approach to other gaits such as trotting (two feet in the air at a time), which allow much higher speeds but which require dynamic stability (the center of mass is not over the support basis, which is only a line for trotting gaits).

7. CONCLUSION

Sony provided us with fully operational quadruped robots to support our research on teams of fully autonomous robotic agents. In this paper, we reported on components developed to create effective fully autonomous quadruped robots in a robotic soccer domain. The team was entered the RoboCup 2000 international competition, and placed third out of twelve teams in the soccer tournament, losing only one game to the eventual champion. We performed very well in overall level of play, demonstrating reliable vision, localization, behaviors, and motions. In one of the physical challenges, the reliability of the overall system allowed us traverse a preset trajectory on the field and score the fastest of all the teams, demonstrating precise sensing, localization, and motions.

This paper presented the main technical contributions underlying the software of this fully implemented robotic system. We presented the object recognition algorithm that allows for the robot to reliably detect objects with multiple colors. We then described the localization algorithm and the behavior system composed of well defined and separated hierarchies for the sensors, the behaviors, and the controls. We also overviewed our sophisticated walking and kicking motions for quadruped robots. Note that several of the algorithms presented in this paper were designed and implemented with a great degree of generality and abstraction. Indeed we have applied the vision processing and the localization approaches to several other robotic platforms.

Finally there is clearly future work ahead of us within the broad and challenging area of multirobot systems. In the near future, we aim at developing and contributing more elaborate team work among the robots, and the application of the quadruped robots, in conjunction with other of our robots, to other tasks, such as robotic search and rescue.

8. REFERENCES

[1] S. Behnke, B. Frötschl, R. Rojas, et al. Using hierarchical dynamical systems to control reactive behavior. In *Proceedings of IJCAI-99*, pages 28–33, 1999.

[2] J. Bruce, T. Balch, and M. Veloso. CMVision (http://www.coral.cs.cmu.edu/cmvision/).

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

[4] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics, Principles and Practice.* Addison-Wesley, Reading, Massachusetts, second edition, 1990.

[5] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte Carlo localization: Efficient position estimation for mobile robots. In *Proceedings of AAAI-99*, 1999.

[6] M. Fujita, M. Veloso, W. Uther, M. Asada, H. Kitano, V. Hugel, P. Bonnin, J.-C. Bouramoue, and P. Blazevic. Vision, strategy, and localization using the Sony legged robots at RoboCup-98. *AI Magazine*, 1999.

[7] V. Hugel. *Contribution a la commande de robots hexapode et quadrupede.* PhD thesis, Universite Paris VI, 1999.

[8] R. Jain, R. Kasturi, and B. Schunk. *Machine Vision.* McGraw-Hill, New York, 1995.

[9] S. Lenser and M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Proceedings of ICRA-2000*, 2000.

[10] C. Poynton. Poynton's color FAQ (http://www.inforamp.net/ poynton/notes/colour_and_gamma/colorfaq.html).

[11] 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 AIPS-00*, 2000.