In *The University of Texas at Austin, Department of Computer Sciences, AI Technical Report 06-337*,
December 2006.

The UT Austin Villa 2006 RoboCup Four-Legged Team

Peter Stone, Peggy Fidelman, Nate Kohl, Gregory Kuhlmann,
Tekin Meriçli, Mohan Sridharan, Shao-en Yu

Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500
Austin, Texas 78712-1188
{pstone,peggy,nate,kuhlmann
tmericli,smohan,shawnyu}@cs.utexas.edu
`http://www.cs.utexas.edu/~AustinVilla`

Technical Report UT-AI-TR-06-337

December 17, 2006

Abstract

The UT Austin Villa Four-Legged Team for RoboCup 2006 was a fourth-time entry in the ongoing series of RoboCup legged league competitions. The team development began in mid-January of 2003 without any prior familiarity with the Aibos. After entering a fairly non-competitive team in RoboCup 2003, the team made several important advances. By the July 2004 competition that took place in Lisbon, Portugal, it was one of the top few teams. After those first two years of intense development, the team's third and fourth years were devoted more to spin-off research than to development related to the competition. Building off of the team's previous three technical reports [14, 15, 16], this report details the changes made to the team between RoboCup 2005 and RoboCup 2006 in Bremen. Taken together, this and the previous technical reports provide the history and details of the UT Austin Villa RoboCup Four-Legged team and the associated research in our lab.

Contents

1 Introduction

RoboCup, or the Robot Soccer World Cup, is an international research initiative designed to advance the fields of robotics and artificial intelligence, using the game of soccer as a substrate challenge domain. The long-term goal of RoboCup is, by the year 2050, to build a team of 11 humanoid robot soccer players that can beat the best human soccer team on a real soccer field [6].

RoboCup is organized into several leagues, including a computer simulation league and two leagues that use wheeled robots. This technical report concerns the development of a team for the Sony four-legged league[1] in which all competitors use identical Sony Aibo ERS-210A and/or ERS-7 robots and the OPEN-R software development kit.[2]

Since all teams use the same commercial robots, the four-legged league is essentially a software competition. In this report, we detail the development of our team, UT Austin Villa,[3] from the Department of Computer Sciences at the University of Texas at Austin.

For this report, we assume familiarity with the robots' specifications and the rules of the RoboCup games. For full details, see the legged league and OPEN-R sites footnoted above. Here we describe both our development process and the technical details of its end result, the UT Austin Villa team. In conjunction with our previous technical reports [14, 15, 16] this paper provides full documentation of the algorithms behind our approach. In particular, this report is *not* intended to stand alone; a full understanding requires familiarity with our previous technical reports. Taken together, they provide the history and details of a relatively young RoboCup team that has quickly grown from a nascent project to an ongoing source of diverse and plentiful research results while becoming a highly-seeded team in competitions.

Our team development began in mid-January of 2003, without any prior familiarity with the Aibos. After entering a fairly non-competitive team in RoboCup 2003, the team made several important advances. By the July 2004 competition in Lisbon, Portugal, it was one of the top few teams. After those first two years of intense development, the team's third and fourth years were devoted more to spin-off research than to development related to the competition. In this document we fully describe the changes to our 2005 code base, which we previously documented in full detail [16]. We include all of our advances as of RoboCup 2006 in June.

The remainder of the report is organized as follows. Section 2 gives an overview of our completely-rewritten vision module. The main functional change is that the code is more efficient computationally. Section 3 is a brief recap of our largely-unchanged approach to localization. Section 4 gives an overview of our revamped motion module. Section 5 represents the most substantial change to our team, namely a move to an interpreted language (Lua) with the interpreter running on-board the Aibo. Section 6 summarizes our changes to the robots' behaviors. Section 7 summarizes our recent research results. Section 8 presents our results in the competitions and Section 9 concludes.

2 Vision

The entire vision module was rewritten this year both to speed up the process and to clean it up to fit better with the modified code architecture (see Section 5). But, in principle, the basic structure of the vision code remained the same as before [15]: the input image was segmented, the segmented pixels were organized into run-lengths and bounding boxes, and objects and lines were found based on the known structure of the robot's environment.

One change that we incorporated this year was to modify the vision code to better handle image rotations. As the robot moves about playing soccer on the field, the legged motion and the three degrees-of-freedom in the robot's head cause the images to be rotated such that the objects in the image are no longer parallel to the horizontal plane. Previously, we used to handle this by heuristically compensating for the estimated tilt and roll of the robot's camera coordinate frame. This year, we implemented two strategies to handle this. The first one estimated the principal axes of the detected blobs (after run-length encoding and region merging) using the blob statistics, similar to the technique used by the team from The University of Pennsylvania [4]. The second determines the convex boundary of the blobs in the image using methods developed by the graphics community, based on convex hulls. Both techniques work equally

[1] http://www.tzi.de/4legged/
[2] http://openr.aibo.com/
[3] http://www.cs.utexas.edu/~AustinVilla

well on the robots but, in the code executed at run-time, we only use the former technique. Also, this compensation is required only for the beacon and goal colored blobs.

The high-level object recognition code was rewritten to speed up the process and also make it feasible to apply the same module to different platforms. For example, the basic vision module involving color segmentation, region formation and object recognition was also tested on a Segway robotic mobility platform (RMP) [2] with very small changes. All the parameters involved in the code were extracted out into a single file that could be modified at run-time without having to reboot the robot. This helped make experimentation with the parameters easier to perform and also helped speed up the debugging process. The resultant module was faster and worked better than in previous years.

This year, visual feedback was also used for the estimation of the robot's body tilt and roll. Typically the robot's accelerometer values are used for the estimation of the robot's body tilt and roll - our approach is described in [15]. Since this process is noisy, most teams average the values over a sliding window to prevent sudden fluctuations in the estimated values. This is still quite noisy and also does not remove an inherent bias. Instead, we used visual feedback to better estimate these values. The robot started out standing in place and looking at a fixed object; here the position of the object in the image, and the robot's body tilt and roll were calculated. Next, the robot walked towards the object and as it moved, it detected the offsets in the position of the object in the image. We made the assumption that while the object is at a distance, the noise in the image offsets is small and hence the offset of the object in the image is an estimate of the robot's body tilt and roll over time. Then we performed weighted linear regression using the values of the accelerometers (which generated the heuristic estimates of the tilt and roll) as inputs, while the image offsets were used as ground truth. Finally, to verify our assumption, we also took actual measurements of the robot's body tilt and roll using an external motion capture system. We found that using the image offsets was a valid assumption, and that this regression process helped remove the bias in the estimation and reduced the error in estimation of the body tilt (and roll) by at least 25%.

3 Localization

The basic localization module was rewritten to conform to the new architecture (see Section 5 below). But, we still retained the Particle Filtering implementation from previous years [15] and except for some bug-fixes the code remains essentially unchanged.

4 Motion

For an intelligent *mobile* robot, the ability to get from one place to another is one of the fundamental skills. However, locomotion on legged robots is a challenging multidimensional control problem. It requires the specification and coordination of motions in all of the robot's legs while accounting for factors such as stability and surface friction. Rapid locomotion becomes crucial specifically when the robot's aim is to chase and dribble a ball as fast as it can and shoot it towards the opponent goal.

In the robot soccer domain, the speed and maneuverability of individual robots are very important factors in determining the overall success of the team. Since the default gait developed by Sony for Aibo is fairly slow, there has been much research within the RoboCup community to develop improved locomotion for these robots.

At the lowest level, the Aibo's gait is determined by a series of joint positions for the three joints in each of its legs. More recently, trajectories of the Aibo's four feet through three-dimensional space have been used to develop a higher level representation for Aibo's gait. A series of inverse kinematics calculations are then performed to convert these trajectories into joint angles. Among higher-level approaches, most of the differences between gaits that have been developed for the Aibo stem from the shape of the loci through which the feet pass and the exact parameterization of those loci.

The biggest change in the motion module of UT Austin Villa in 2006 was switching to the Cerberus motion engine as the base model [3, 8] which was more flexible in terms of providing a smooth omnidirectional motion and a set of different locus representations including rectangular, elliptic, and Hermite curve shaped trajectories. Although those representations provided reasonable results, the biggest constraints they had in common were being 2D representations

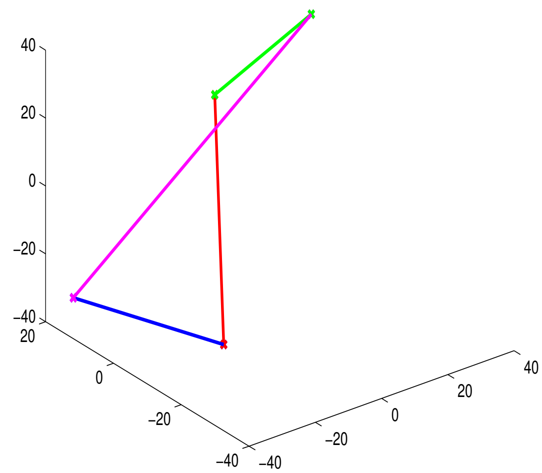

Figure 1: An optimized 3D polygon that the front right leg traces during forward walk.

and a lack of joint behavior modeling. The German Team showed that representing loci in 3D yields a much better result in both walking speeds in various directions and maneuverability [5].

The most problematic part of designing the motion module, especially the walking engine, is having to deal with the difference between the specified motor behavior and the actual motor behavior. This difference may be caused by various factors such as the response time of individual motors, coordination of several motors, or some kinematic errors. Most of the approaches so far either did not take joint modeling issue into account at all (i.e. using static inverse kinematics, not worrying about dynamic effects) or had a separate module for handling this often-computationally-expensive operation. However, using a polygon defined in three dimensional space to represent the loci makes it possible to implicitly take joint modeling into account. After training is completed, the resulting polygons and the time necessary to trace them make the legs move in such a way as to produce optimal loci (even if the polygons look contorted when plotted). Figure 1 illustrates an optimized 3D polygon that front right leg traces during forward walk.

In order to make the robot move in various directions, the legs are treated as wheels and the polygons are rotated in the desired direction of walk. A walk command is defined with four components; the type of the walk — walk with ball or without ball — and forward, sideways, and turn vector length values. Every combination of those three vectors creates an imaginary point on the field where the robot *rotates* about. Each leg moves in the direction of the line that is perpendicular to the line connecting the paw and the center of rotation. Walking speed is defined by the size of the polygons. In order to walk faster, the steps of the robot become longer although the time for completing one step remains the same.

During the 2006 competitions, in order to get the maximum benefit from the 3D polygon approach, we dynamically switched between our Cerberus-based motion engine and the German Team motion engine [5] based on the requested walk type; that is, *walkWithBall* if the robot is to walk while keeping the ball under its chin, or just *walk* otherwise. Since our Cerberus-based motion engine provided a safer posture for ball control, the motion commands were processed using that engine if the requested walk type was *walkWithBall*, and the commands were sent to German Team engine if the requested walk type was not related to ball control. Currently ongoing work is to use our own engine's capability of representing various locus types as an extension to the base Locus class, and to let our own engine execute it rather than the German Team engine.

Unlike the motion module used until 2005 in which the gait definition parameters were learned using the policy gradient algorithm [7], the parameters in our new, Cerberus-based walking engine were learned using GAs [8], and the parameters from the German Team motion engine, which are the 3D coordinates of 4 vertices of the polygon and the amount of time needed for the foot to travel from one vertex to the next, were learned using a (μ, λ) evolution strategy with self-adapting mutation. Different polygons are used for front and rear legs for a certain walking direction and those polygon definitions differ for each of the main directions that the robot walks towards [5].

5 Code Architecture

In early 2006, the code for UT Austin Villa underwent a significant rewrite. The main results of this change were improved code organization between the simulator and the robot and the introduction of a new behavior system written in the Lua programming language. This section describes the revamped architecture, the most significant change in the 2006 UT Austin Villa code base.

5.1 The Abstract Robot

In the legged league, it has often been seen as desirable for a team to have the ability to debug code in simulation before undergoing costly real-world evaluations on the Aibo. The simulator that our team created has proven to be useful in previous years for tasks like debugging localization, but due to a lack of integration with the rest of our code, it was used for little else.

In order to improve the situation, we reorganized our code to incorporate the notion of an *abstract robot*. The notion of this abstract robot allowed us to separate all of the robot-specific code (things like OPEN-R function calls and data types) from more general high-level code (algorithms like localization and vision).

One instance of this abstract robot was created for the actual Aibo robot, which would deal with real-world percepts via the normal OPEN-R architecture. Another instance of the abstract robot was created for a simulated robot, which received information from the simulator.

Abstracting all of our high-level code required quite a bit of work, but once done we were free to run our entire code-base in simulation. This frequently allowed us to sidestep the costly Aibo development cycle, since we were able to now run our code in simulation to catch simple mistakes.

5.2 Using Lua for Behaviors

In previous years, the behavior of our team (like that of many others) was controlled by pure C++ code. Since the native language of the Aibo is C++, it is a natural first step to write all of the code that controls the Aibo in C++. However, there are several problems with writing Aibo code in C++. When the code crashes, the entire OPEN-R object crashes, often resulting in a dead Aibo. Furthermore, making changes to the code is a time-intensive process, often requiring a trip to the memory stick reader and a reboot of the robot.

To address these problems, we incorporated a Lua interpreter into our team code. The Lua language is a highly portable interpreted language that was designed to compile easily on a multitude of different systems. A package called SWIG allowed us to interface the native language of the Aibo (C++) with the Lua interpreter, which meant that we could call C++ functions and access C++ data from Lua.

The main loop of our abstract robot was actually written in Lua, as was much of the other functionality of the robot. Certain pieces of code that needed to run quickly (like vision and localization) were still written in C++, but were called from the Lua main loop. The behavior code for the robot was written entirely in Lua.

In addition, we created several scripts that allowed us to transfer new Lua files to the Aibo in real-time. If we had written a particular part of the robot code in Lua, we now had the ability to change that code and send it to the robot, resulting in a near-instantaneous development cycle. This ability to quickly change and evaluate the code proved especially useful for designing and debugging behaviors.

6 Team Strategy and Behavior

As mentioned in the previous section, we changed the code architecture so that all the strategy and behavior was programmed in Lua. The first thing we did was to rewrite a lot of the primitive behaviors in Lua (such as the basic motions, kicks etc), modeling it on the C++ code we had from previous years. Our motion engine had undergone a change from previous years (see Section 4 above) and the existing behavior module, which had been built over a couple of years, had become difficult to manage, prompting us to rewrite the whole high-level behavior module rather than trying to create the Lua equivalent of the existing C++ code. For the same reason, we also decided to rewrite the team strategy module.

6.1 Behavior Changes

Ball Following This year we had different approaches for the roles of *Attacker* and *Defender*. The *Attacker*, as in our code from previous years, goes straight to the ball with the maximum possible speed. However, the *Defender*'s main aim is to block a shot to the goal. Going straight to the ball might leave an open line to the goal for a robot from the other team. Therefore, the *Defender* tends to use a combination of sideways and forward motion to keep itself in a suitable position (between the ball and the goal) as it moves to the ball. When it is just supposed to block a shot at the goal (some other robot from the team is closer to the ball and is moving towards it), it just positions itself suitably, without moving towards the ball. In either case the target position changes dynamically as a function of the current robot pose and ball position.

For both the *Attacker* and *Defender*, it does not make sense to just focus on the ball while it is still some distance away from the robot (we use a threshold distance of 1m). Instead, while going after a ball (or keeping track of a ball) more than this threshold distance away from the robot, the robot periodically looks in the direction where it expects to see markers, in an attempt to stay localized, so that it can react quickly when it gets to the ball (or if it needs to change its position with respect to the ball). Since this requires looking away from the ball, the robot uses this behavior (known as *Active Localization* and introduced into our team in 2004 [15]) *iff* it has high confidence in its ball estimate.

Chin-Pinching We ported the existing chin-pinch learning code [16] from C++ to Lua and rewrote a training task for single robot to pinch the ball with the experimental parameter sets and used those parameter sets and the resultant chin-pinching performance as inputs to our policy gradient algorithm to generate the next output set. During the US Open this year, we observed that the chest sensor provided false positives regarding the presence of the ball under the chin. So, similar to the method used by other RoboCup teams, we added an additional check to determine the presence of the ball under the chin: we requested a head angle that would put the head closer to the ground than would have been possible if the ball were actually under the chin. If the head is unable to reach the requested position, we used it as an additional indication of the fact that the ball is really under the chin.

Shooting Towards Open Space Our initial attacker behavior had the robot shooting in the direction of the goal based on localization. But the lack of walls around the field leads to a lot of missed open shots at the goal because of even small localization errors. Therefore, we rewrote the behavior to have the robot function in a more reactive fashion when it is closer to the opponent's goal. More specifically, when the robot, with the ball under its chin, is within a certain threshold range of the goal (based on localization), it tries to shoot in the direction of the biggest goal-colored blob in vision that satisfies certain basic sanity checks (to filter out noise from the background). This simple change proved to be fairly effective in real games since it also enables the robot to avoid the goal keeper and shoot at the most *open* region of the target goal. Using this behavior also enabled us to win the *New Goal Challenge* this year, with just the basic attacker behavior, since the robot just aimed at the biggest yellow blob that satisfied the sanity checks once it was within our predefined shooting range of the goal.

When the robot turns at full speed towards the goal (with the ball held under its chin), by the time it stops in response to the visual sighting of a goal-colored blob, it tends to overshoot by $15 - 20$ degrees. We solved this problem using the following strategy. The robot turns at full speed searching for the goal-colored blob. Once one is found, the robot assumes that it has overshot the desired direction and turns in the opposite direction at one-half of the earlier speed. If a suitable blob is found, it tries to align and shoot (at one-third the maximum speed) at the space that is most open. If not, it takes a shot in the direction of the previous goal-colored blob sighting. The whole action also has a time threshold to ensure that the robot does not receive a penalty for holding the ball for too long.

Dribbling Because of the overhaul to our behavior infrastructure (to Lua), we introduced the dribbling action in the code within a week of the competition. Since we lacked time to test it in real games, we implemented a simple strategy. The robot used its knowledge of its surroundings, using its own visual information and the information communicated by its teammates, to determine if there was an opponent in its intended direction of motion. If an opponent, i.e. obstacle, existed in this direction, the robot performed a sideways dodging motion and then moved forward dribbling the ball. While dodging, the robot tries to move into the field in order to avoid running the ball out of bounds. Our fast forwards walk enables the robot to keep control of the ball during its forward motion by just putting its head above the ball. The *Ball Holding* rules prevent us from controlling the ball (under the chin) and moving it for more than 50cm or 3 seconds. Therefore, the robot times out of this behavior just before it would incur the penalty and ends the motion, typically with a slide forward kick that moves the ball further in the intended direction. We also enabled the robot, while dribbling, to react to the visual input. If, for example, a suitable target goal-colored blob is seen, the robot suitably adjusts its dribbling direction.

Kick Decisions In previous years, the decisions about where to perform which types of kicks (and dribbles) were spread throughout various parts of the behavior code. To enable more straightforward experimentation with the kick decisions, we refactored the code this year so that all the decisions about kick preparation, direction to aim the ball, and kick type were encapsulated into a single function. This function is consulted as soon as the attacking robot gets "close" to the ball, since certain kick preparations (such as a chin-pinch) will need to begin soon after this if they are selected. However, at the moment when the preparation ends and the aiming stage (or the kick itself) is about to begin, there is a chance for the robot to reevaluate this decision. If localization information that has been gathered since the original kick decision indicates that a different kick direction or type *that requires the same preparation* would be better, the decision is revised and the new aim / kick is carried out. Otherwise, the robot executes the original decision.

6.2 Strategy Changes

Once the basic behavior was in place for a single robot, we rewrote the team strategy for the team of robots to function together. The first iteration happened on the flight to the US-Open and was very simple, having the team of robots use the communicated ball knowledge to decide on the robot that would actually go to the ball while the other two robots (the keeper always guards the goal and makes its own decisions) positioned themselves along the other two vertices of a triangle around the ball. The *Defender* would always position itself in a such a position as to block a shot at the goal it was guarding while the *Supporter* would always place itself on the other side of the field from the *Attacker* in a position where it could quickly get to the ball if the attacker missed in its attempt to score on the goal. We also added in our customary features that allowed the robots to smoothly trade roles and dynamically adjust positions based on the positions of the individual robots and the ball [15].

In principle the basic strategy was quite similar to the one we had from the previous year (see [16]). Writing it in Lua helped us make it cleaner and easier to understand and modify. Unfortunately, we could not get much time to test the modifications made to our strategy in mock games because most of our time between the two competitions was spent debugging the low-level modules and the essential single robot behaviors. Still, we managed to incorporate the communicated ball, opponents' and teammates' positions in the decision process to achieve smooth role allocation and role switching. A drawback of our current strategy is that our robots tend to be more conservative in staying out of each other's paths than in previous years. We observed that the more successful teams are more aggressive in terms of how far apart the robots stay from each other and we plan to change our strategy accordingly.

7 Research Results

In this section, we briefly describe the research projects that we performed using the Aibo as the test platform. For most of these projects, detailed description, links to appropriate papers, and supporting materials such as images and videos can be found online on our team's website [1].

7.1 Generalized Planned Color Learning

In previous work [11], we had enabled the robot to learn the colors on the robot soccer field, modeling colors as 3D Gaussians, using a pre-defined motion sequence. This year, we extended the approach in two significant ways. The color learning works both in the controlled lab setting and in un-engineered indoor corridors by proposing a hybrid color model. We also enabled the robot to plan a motion sequence appropriate for learning colors, using the known model of its color-coded world. The algorithm is described in [12] and detailed experimental results can be found online:`www.cs.utexas.edu/users/AustinVilla/?p=research/gen_color`.

7.2 Adapting to Changing Illumination Conditions

In previous work [10], we had shown that if the robot is provided suitable color maps and image statistics for different illumination conditions, it can transition smoothly between the color maps based on a comparison of the image characteristics. We aim to have the entire color learning algorithm to execute autonomously under changing illumination conditions. We extended our approach by enabling the robot to detect changes in illumination conditions automatically. If an illumination change is detected, the robot automatically adapts to the change by revising its color knowledge be re-learning the colors. Complete details, including the algorithm and experimental results, are available in [13] and supporting images are available for viewing online:`www.cs.utexas.edu/users/AustinVilla/?p=research/illuminvar_colorlearn`.

7.3 Learning a More Stable Walk

A fast gait is an essential component of any successful team in the RoboCup 4-legged league. However, quickly moving quadruped robots, including those with learned gaits, often move in such a way so as to cause unsteady camera motions which degrade the robot's visual capabilities. In previous research, we presented a method for automatically learning a *fast* gait [7]. This year, we presented an implementation of the policy gradient machine learning algorithm that searches for a parameterized walk while optimizing for both speed and stability [9]. To the best of our knowledge, previous learned walks have all focused exclusively on speed. Our method is fully implemented and tested on the Sony Aibo ERS-7 robot platform. The resulting gait is reasonably fast and considerably more stable compared to our previous fast gaits. We demonstrate that this stability can significantly improve the robot's visual object recognition. Videos are available on-line at `www.cs.utexas.edu/~AustinVilla/?p=research/learned_walk`.

7.4 Selective Visual Attention for Object Detection

Autonomous robots can use a variety of sensors, such as sonar, laser range finders, and bump sensors, to sense their environments. Visual information from an on-board camera can provide particularly rich sensor data. However, processing all the pixels in every image, even with simple operations, can be computationally taxing for robots equipped with cameras of reasonable resolution and frame rate. We present a novel method for a legged robot equipped with a camera to use selective visual attention to efficiently recognize objects in its environment [18]. The resulting attention-based approach is fully implemented and validated on an Aibo ERS-7. It effectively processes incoming images 50 times faster than a baseline approach, with no significant difference in the efficacy of its object detection. More information and a video is available on-line at `www.cs.utexas.edu/~AustinVilla/?p=research/model-based_vision`.

7.5 Autonomous Sensor and Actuator Model Induction

We presented a novel methodology for a robot to autonomously induce models of its actions and sensors called ASAMI (Autonomous Sensor and Actuator Model Induction) [17]. While previous approaches to model learning rely on an independent source of training data, we show how a robot can induce action and sensor models without any well-calibrated feedback. Specifically, the only inputs to the ASAMI learning process are the data the robot would naturally have access to: its raw sensations and knowledge of its own action selections. From the perspective of developmental robotics, our robot's goal is to obtain self-consistent internal models, rather than to perform any externally defined tasks. Furthermore, the target function of each model-learning process comes from within the system, namely the most current version of another internal system model. Concretely realizing this model-learning methodology presents a number of challenges, and we introduce a broad class of settings in which solutions to these challenges are presented. ASAMI is fully implemented and tested, and empirical results validate our approach in a robotic testbed domain using a Sony Aibo ERS-7 robot. Videos of the learning process are available on-line at `www.cs.utexas.edu/~AustinVilla/?p=research/learned_walk`.

8 The Competitions

In the RoboCup initiative, periodic competitions create fixed deadlines that serve as important motivators. In 2006, we entered the Fourth U.S. Open competition as well as the Tenth International RoboCup Competition. This section describes our results and experiences at those events.

8.1 U.S. Open, 2006

The Fourth U.S. Open RoboCup Competition was held in Atlanta, GA from April 22nd to 25th, 2006. The results of our games are shown in Table 1 below.

Opponent	**Score (us-them)**	**Notes**
Brown	7–0	Pool Play
Bowdoin	6–0	Pool Play
Dortmund	0–6	Pool Play
CMU	0–1	Semi-final
UPenn	1–3	Third place

Table 1: The scores of our five games at the U.S. Open.

At the time of this competition, our revamped Lua-based code was still in its final debugging stages. In an attempt to get everything working, many last-minute changes were made at the competition site, which is always a dangerous endeavor. In the end, the team did not play up to its potential, but, as is the purpose of the opens, much progress was made towards completing the new architecture, which then put us in a good position to move forward towards the international RoboCup.

8.2 RoboCup 2006

The Tenth International RoboCup Competition was held in Bremen, Germany, from June 14th to 18th, 2006.[4] Twenty-four teams competed in the four-legged league and were divided into eight groups of three. The results of our first round and second round games are tabulated in Table 2 shown below.

Opponent	**Score (us-them)**	**Notes**
BabyTigers	6–0	First Round
EagleKnights	3–0	First Round
German Team	1–3	Second Round
Dutch Aibo	0–5	Second Round
FC-Twaves	7–1	Second Round

Table 2: The scores of our five games at RoboCup.

[4] `http://www.robocup2006.org`

At RoboCup, our team was stronger than it had been the previous year. But, as should always be expected, the general level of play of all the other teams also improved. Our team scored some clear victories, but also came up against some tough competition. Ultimately, we failed to advance to the quarterfinals for the first time in three years. Nonetheless, it was a positive experience for our new team members who had not previously experienced the excitement of RoboCup.

9 Conclusions and Future Work

The experiences and algorithms reported in this technical report document the third year of the UT Austin Villa legged-league robot team.

There are still many directions for future improvements to our team, as noted throughout this report. We plan to continue our development toward future RoboCup competitions. But more importantly, we continue to make use of this code base as a fully functional research platform and are using it for investigations in various directions as summarized in Section 7.

Overall, developing a competitive RoboCup soccer team has been a rewarding learning experience. We look forward to building from it in the future and continuing to contribute to the RoboCup initiative.

Acknowledgments

The authors would like to thank Sony for developing the robots, as well as the previous RoboCup legged-league teams for forging the way and providing their source code and technical reports as documentation. This research is supported in part by NSF CAREER award IIS-0237699 and ONR YIP award N00014-04-1-0545.

References

[1] The UTAustinVilla research website, 2004. `http://www.cs.utexas.edu/users/AustinVilla`.

[2] The Segway Robots, 2006. `http://www.segway.com/products/rmp/`.

[3] H. Levent Akın, Çetin Meriçli, Tekin Meriçli, Kemal Kaplan, and Buluç Çelik. Cerberus'05 team report. Technical report, Artificial Intelligence Laboratory, Department of Computer Engineering, Boğaziçi University, October 2005.

[4] G. Buchman, D. Cohen, P. Vernaza, and D. D. Lee. *RoboCup-2005: The Ninth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2006.

[5] Thomas Rofer et al. Germanteam 2005 team report. Technical report, October 2005.

[6] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: The robot world cup initiative. In *Proceedings of the First International Conference on Autonomous Agents*, pages 340–347, Marina Del Rey, California, February 1997.

[7] Nate Kohl and Peter Stone. Machine learning for fast quadrupedal locomotion. In *The Nineteenth National Conference on Artificial Intelligence*, pages 611–616, July 2004.

[8] Tekin Meriçli. Developing a rapid and stable parametric quadruped locomotion for aibo robots. Technical report, Department of Computer Engineering, Marmara University, October 2005.

[9] Manish Saggar, Thomas D'Silva, Nate Kohl, and Peter Stone. Autonomous learning of stable quadruped locomotion. In Gerhard Lakemeyer, Elizabeth Sklar, Domenico Sorenti, and Tomoichi Takahashi, editors, *RoboCup-2006: Robot Soccer World Cup X*. Springer Verlag, Berlin, 2007. To appear.

[10] M. Sridharan and P. Stone. Towards illumination invariance in the legged league. In *The International RoboCup Symposium*, 2004.

[11] M. Sridharan and P. Stone. Autonomous color learning on a mobile robot. In *The Twentieth National Conference on Artificial Intelligence (AAAI)*, 2005.

[12] M. Sridharan and P. Stone. Autonomous planned color learning on a mobile robot without labeled data. In *The Ninth IEEE International Conference on Control, Automation, Robotics and Vision (ICARCV)*, December 2006.

[13] M. Sridharan and P. Stone. Color learning on a mobile robot: Towards full autonomy under changing illumination. In *The International Joint Conference on Artificial Intelligence (IJCAI)*, Hyderabad, India, January 2007.

[14] Peter Stone, Kurt Dresner, Selim T. Erdoğan, Peggy Fidelman, Nicholas K. Jong, Nate Kohl, Gregory Kuhlmann, Ellie Lin, Mohan Sridharan, Daniel Stronger, and Gurushyam Hariharan. UT Austin Villa 2003: A new RoboCup four-legged team. Technical Report UT-AI-TR-03-304, The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, 2003.

[15] Peter Stone, Kurt Dresner, Peggy Fidelman, Nicholas K. Jong, Nate Kohl, Gregory Kuhlmann, Mohan Sridharan, and Daniel Stronger. The UT Austin Villa 2004 RoboCup four-legged team: Coming of age. Technical Report UT-AI-TR-04-313, The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, October 2004.

[16] Peter Stone, Kurt Dresner, Peggy Fidelman, Nate Kohl, Gregory Kuhlmann, Mohan Sridharan, and Daniel Stronger. The UT Austin Villa 2005 RoboCup four-legged team. Technical Report UT-AI-TR-05-325, The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, November 2005.

[17] Daniel Stronger and Peter Stone. Towards autonomous sensor and actuator model induction on a mobile robot. *Connection Science*, 18(2):1–23, 2006. Special Issue on Developmental Robotics.

[18] Daniel Stronger and Peter Stone. Selective visual attention for object detection on a legged robot. In Gerhard Lakemeyer, Elizabeth Sklar, Domenico Sorenti, and Tomoichi Takahashi, editors, *RoboCup-2006: Robot Soccer World Cup X*. Springer Verlag, Berlin, 2007. To appear.