The UT Austin Villa 2003 Four-Legged Team

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

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

Abstract. The UT Austin Villa RoboCup 2003 Four-Legged Team was a new entry in the ongoing series of RoboCup legged league competitions. The team development began in mid-January of 2003, at which time none of the team members had any familiarity with the Aibos. Without using any RoboCup-related code from other teams, we entered a team in the American Open competition at the end of April, and met with some success at the annual RoboCup competition that took place in Padova, Italy at the beginning of July. In this paper, we describe aspects of (i) our development process and (ii) the technical details of its end result, the UT Austin Villa team. Complete details in both regards are available in our team technical report [1].

1 Introduction

The UT Austin Villa RoboCup 2003 Four-Legged Team was a new entry in the ongoing series of RoboCup legged league competitions. For the purposes of this paper, we assume familiarity with the specifications of the robots as well as the rules of the RoboCup games. For full details see the legged league[1] and Open-R[2] websites. Here we describe both our development process and the technical details of its end result, the UT Austin Villa team. The main contribution of this paper is a roadmap for new teams entering the competition who are starting from scratch. Full documentation of the algorithms behind our approach can be found in our team technical report [1].

Our team development began in mid-January of 2003, at which time none of the team members had any familiarity with the Aibos. Without using any RoboCup-related code from any other teams, we entered a team in the American Open competition at the end of April, and met with some success at the annual RoboCup competition that took place in Padova, Italy at the beginning of July.

[1] `http://www.openr.org/robocup/index.html`
[2] `http://openr.aibo.com/`

Although our team was not one of the top few at the competition, we view it as a great accomplishment that we were able to develop a competitive team in such a short time. The primary goal of this paper is to document our development process as a guide for new teams in the future.

Our effort began as a graduate research seminar offered as a class during the Spring semester of 2003. The following section outlines the structure of the class. At the end of that section we outline the structure of the remainder of the paper.

2 The Class

The UT Austin Villa 2003 legged robot team began as a focused class effort during the Spring semester of 2003 at the University of Texas at Austin. Nineteen graduate students and one undergraduate were enrolled in the course CS395T: *Multi-Robot Systems: Robotic Soccer with Legged Robots.*[3] All of the authors on this paper participated in the class.

Students in the class studied past approaches, both as described in the literature and as reflected in publicly available source code. However, we developed the entire code base *from scratch* with the goals of learning about all aspects of robot control and of introducing a completely new code base to the community.

Class sessions were devoted to students educating each other about their findings and progress, as well as coordinating the integration of everybody's code. Just nine weeks after their initial introduction to the robots, the students already had preliminary working solutions to vision, localization, fast walking, kicking, and communication.

The concrete goal of the course was to have a completely new working solution by the end of April so that we could participate in the American Open competition, which happened to fall during the last week of the class. After that point, a subset of the students continued working towards RoboCup 2003 in Padova.

The class was organized into three phases. Initially, the students created simple behaviors with the sole aim of becoming familiar with Open-R.

Then, about two weeks into the class we shifted to phase two by identifying key subtasks that were important for creating a complete team. Those subtasks were: vision; movement; fall detection; kicking; localization; communication; general Architecture; and coordination. During this phase, students chose one or more of these subtasks and worked in subgroups on generating initial solutions to these tasks in isolation.

By about the middle of March, we were ready to switch to phase three, during which we emphasized "closing the loop," or creating a single unified code-base that was capable of playing a full game of soccer. We completed this integration process in time to enter a team in the RoboCup American Open competition at the end of April.

---
[3] `http://www.cs.utexas.edu/~pstone/Courses/395Tspring03`

The remainder of the paper is organized as follows. In Section 3 we document some of the initial behaviors that were generated during phase one of the class. Next we document the output of some of the subgroups that were formed in phase two of the class in Sections 4–7. Next, we document the tasks that occupied phase three of the class, namely those that allowed us to put together the above modules into a cohesive code base (Sections 8–11. Section 12 introduces our debugging and development tool. Then in Section 13 we summarize our experiences at the RoboCup 2003 competition, and Section 14 concludes.

3 Initial Behaviors

The first task for the students in the class was to learn enough about the Aibo to be able to compile and run any simple program on the Aibo.

The open source release of Open-R came with several sample programs that could be compiled and loaded onto the Aibo right away. These programs could do simple tasks such as:

L-Master-R-Slave: Cause the right legs to mirror manual movements of the left legs.

Ball-Tracking-Head: Cause the head to turn such that the pink ball is always in the center of the visual image (if possible).

PIDcontrol: Move a joint to a position specified by the user by typing in a telnet window.

The students were to pick any program and modify it, or combine two programs in any way. The main objective was to make sure that everyone was familiar with the process for compiling and running programs on the Aibos. Some of the resulting programs included:

- Variations on L-Master-R-Slave in which different joints were used to control each other. For example, one student used the tail as the master to control all 4 legs, which resulted in a swimming type motion. Doing so required scaling the range of the tail joints to those of the leg joints appropriately.
- Variations on Ball-Tracking-Head in which a different color was tracked. Two students teamed up to cause the robot to play different sounds when it found or lost the ball.
- Variations on PIDcontrol such that more than one joint could be controlled by the same input string.

After becoming familiar with the compiling and uploading process, the next task for the students was to become more familiar with the Aibo's operating system and the Open-R interface. To that end, they were required to create a program that added at least one new subject-observer connection to the code.[4]

[4] A subject-observer connection is a pipe by which different Open-R objects can communicate and be made interdependent. For example, one Open-R object could send a message to a second object whenever the back sensor is pressed, causing the second object to, for example, suspend its current task or change to a new mode of operation.

The students were encouraged to create a new Open-R object from scratch. Pattern-matching from the sample code was encouraged, but creating an object as different as possible from the sample code was preferred.

Some of the responses to this assignment included:

- The ability to turn on and off LEDs by pressing one of the robots' sensors.
- A primitive walking program that walks forward when it sees the ball.
- A program that alternates blinking the LEDs and flapping the ears.

After this assignment, which was due after just the second week of the class, the students were familiar enough with the robots and the coding environment to move on to their more directed tasks with the aim of creating useful functionality.

4 Vision

The ability of the robot to sense its environment is a prerequisite for any decision making on the Aibo. As such, we placed a strong emphasis on the vision component of our team. The vision module processes the images taken by the CMOS camera located on the Aibo. The module identifies colors in order to recognize objects, which are then used to localize the robot and to plan its operation.

Our visual processing is done using the established procedure of color segmentation followed by object recognition. Color segmentation is the process of classifying each pixel in an input image as belonging to one of a number of predefined color classes based on the knowledge of the ground truth on a few training images. Though the fundamental methods employed in this module have been applied previously (both in RoboCup and in other domains), it has been built from scratch like all the other modules in our team. Hence, the implementation details [1] provide our own solutions to the problems we faced along the way. We have drawn some of the ideas from the previous technical reports of CMU [2] and UNSW [3]. This module can be broadly divided into two stages: (i) low-level vision, where the color segmentation and region building operations are performed, and (ii) high-level vision, wherein object recognition is accomplished and the position and bearing of the various objects in the visual field are determined.

5 Movement

Enabling the Aibos to move precisely and quickly is equally as essential to the overall RoboCup task as the vision task. In this section, we introduce our approach to Aibo movement, including walking and the interfaces from walking to the higher level control modules.

The Aibo comes with a stable but slow walk. From watching the videos of past RoboCups, and from reading the available technical reports, it became clear that a fast walk is an essential part of any RoboCup team. The walk is perhaps the most feasible component to borrow from another team's code base, since it can be separated out into its own module. Nonetheless, we decided to create our own walk in the hopes of ending up with something at least as good, if not better, than that of other teams, while retaining the ability to fine tune it on our own.

The movement component of our team can be separated into two parts. First, the walking motion itself, and second, an interface module between the low level control of the joints (including both walking and kicking) and the decision-making components.

5.1 Walking

At the lowest level, walking is effected on the Aibo by controlling the joint angles of the legs. Each of the four legs has three joints known as the rotator, abductor, and knee. The rotator is a shoulder joint that rotates the entire leg (including the other two joints) around an axis that runs horizontally from left to right. The abductor is the shoulder joint responsible for rotating the leg out from the body. Finally, the knee allows the lower link of the leg to bend forwards or backwards, although the knees on the front legs primarily bend the feet forwards while the ones on the back legs bend primarily backwards. Each joint is controlled by a PID mechanism [5].

The problem of compelling the robot to walk is greatly simplified by a technique called inverse kinematics. This technique allows the trajectory of a leg to be specified in terms of a three-dimensional trajectory for the foot. The inverse kinematics converts the location of the foot into the corresponding settings for the three joint angles. A precursor to deriving inverse kinematics formulas is having a model of the forward kinematics, the function that takes the three joint angles to a three-dimensional foot position. This is effectively our mathematical model of the leg.

Our walk uses a trot-like gait in which diagonally opposite legs step together. That is, first one pair of diagonally opposite legs steps forward while the other pair is stationary on the ground. Then the pairs reverse roles so that the first pair of legs is planted while the other one steps forward. As the Aibo walks forward, the two pairs of diagonally opposite legs continue to alternate between being on the ground and being in the air. For a brief period of time at the start of our developmental process, we explored the possibility of other gait patterns, such as a walking gait where the legs step one at a time. We settled on the trot gait after watching video of RoboCup teams from previous years.

While the Aibo is walking forwards, if two feet are to be stationary on the ground, that means that they have to move backwards with respect to the Aibo. In order for the Aibo's body to move forwards in a straight line, each foot should move backwards in a straight line for this portion of its trajectory. For the remainder of its trajectory, the foot must move forward in a curve through the air. We opted to use a half ellipse for the shape of this curve.

5.2 General Movement

Control of the Aibo's movements occurs at three levels of abstraction.

1. The lowest level, the "movement module," resides in a separate Open-R object from the rest of our code (as described in the context of our general architecture in Section 8) and is responsible for sending the joint values to *OVirtualRobotComm*, the provided Open-R object that serves as an interface to the Aibo's motors.
2. One level above the movement module is the "movement interface," which handles the work of calculating many of the parameters particular to the current internal state and sensor values. It also manages the inter-object communication between the movement module and the rest of the code.
3. The highest level occurs in the behavior module itself (Section 10), where the decisions to initiate or continue entire types of movement are made.

Details of all three of these levels can be found in our technical report [1].

6 Kicking

The robot's kick is specified by a sequence of poses. A $Pose = (j_1, \ldots, j_n)$, $j_i \in \Re$, where j represents the positions of the n joints of the robot. The robot uses its PID mechanism to move joints 1 through n from one $Pose$ to another over a time interval t. We specify each kick as a series of moves $\{Move_1, \ldots, Move_m\}$ where a $Move = (Pose_i, Pose_f, \Delta t)$ and $Move_{jPose_f} = Move_{(j+1)Pose_i}$, $\forall j \in [1, m-1]$. All of our kicks only used 16 of the robot's joints (leg, head, and mouth).

We soon realized that we would need to create several different kicks for different purposes. To that end, we started thinking of the kick-generation process in more general terms. In this section we formalize that process.

The kick is an example of a fine-motor control motion where small errors matter. Creation of a kick requires special attention to each *Pose*. A few angles' difference could affect whether the robot makes contact with the ball. Even a small difference in Δt in a *Move* could affect the success of a kick. To make matters more complicated, our team needed the kick to transition from and to a walk. More consideration had to be taken to ensure that neither the walk nor the kick disrupted the operation of the other.

We devised a two-step technique for kick-generation:

1. Creating the kick in isolation from the walk.
2. Integrating the kick into the walk.

Details of these steps as well as how this process was used to create more than a half-dozen kicks can be found in our technical report [1].

7 Localization

Since it requires at least vision and preferably locomotion to already be in place, localization was a relatively late emphasis in our efforts. In fact, it did not truly come into place until after the American Open Competition at the end of April. Before that time, we had been working on a preliminary approach that was eventually discarded and replaced by the current one.

For self-localization, the Austin Villa team implemented a Monte-Carlo localization approach similar to the one used by the German Team [4]. This approach uses a collection of particles to estimate the global position and orientation of the robot. These estimates are updated by visual percepts of fixed landmarks and odometry data from the robot's movement module. The particles are averaged to find a best guess of the robot's pose.

We have extended the approach of the German Team to improve the accuracy of the observation updates. Rather than using only the most current landmark observations, our approach maintains a history of recent observations that are averaged according to their estimated accuracy. Because it is rare for the robot to gather sufficient information in a single camera frame to triangulate its position, it is important to incorporate visual information from the recent past. At the same time, if visual data is inaccurate, reusing it again and again can aggravate the problem. Our approach is able to leverage past data while, in most situations, robustly tolerating occasional bad inputs.

8 General Architecture

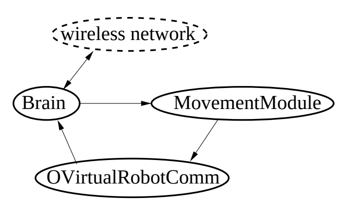

Fig. 1. A high level view of the main Open-R objects in our agent. The robot sends visual data to the Brain object, which sends movement commands to the Movement-Module object, which sends set points to the PID controllers in the robot. The Brain object also has network connections to teammates' Brain objects, the Robocup game controller, and our UT Assist client (Section 12). Note that this figure omits sensor readings obtained via direct Open-R API calls.

Due to our bottom-up approach, we did not address the issue of general architecture until some important modules had already taken shape. We had some code that cobbled together our vision and movement components to produce a rudimentary but functional goal-scoring behavior. Although this agent worked, we realized that we would need a more structured architecture to develop a more sophisticated agent, particularly with the number of programmers working concurrently on the project. The decision to adopt the architecture described below did not come easily, since we already had something that worked. Implementing a cleaner approach stopped our momentum in the short term and required some team members to rewrite their code, but we feel the effort proved worthwhile as we continued to combine more independently-developed modules.

We designed a framework for the modules with the aim of facilitating further development. We considered taking advantage of the operating system's inherent distributed nature and giving each module its own process. However, we decided that the task did not require such a high degree of concurrent computation, so we organized our code into just two separate concurrent objects (Figure 1).

9 Global Map

Early in the development of our soccer playing agent, particularly before we had functioning localization and communication, we chose our actions using a simple finite state machine. Our sensory input and feedback from the Movement Module dictated state transitions, so sensations had a relatively direct influence on behavior. However, once we developed the capability to locate our agents and the ball on the field and to communicate this information, such a direct mapping became impossible. We created the global map to satisfy the need for an intermediate level of reasoning. The global map combines the outputs of Localization from Vision and from Communication into a coherent picture of what is happening in the game, and it provides methods that interpret this map in meaningful ways to the code that governs behavior.

10 Behaviors

In this section we describe the robot's soccer-playing behaviors. In our development, we had relatively little time to focus on behaviors, spending much more of our time building up the low-level modules such as walking, vision, and localization. As such, the behaviors we describe here are far from ideal. We anticipate overhauling this component of our code base should we participate in future competitions.

One of the most important skills for a soccer-playing robot is the ability to score, at least on an empty goal. Our behaviors are implemented by a Finite State Machine (FSM), wherein at any time the Aibo is in one of a finite number of states. The states correspond roughly to primitive behaviors, and the transitions between states depend on input from vision, localization, the global map, and joint angles. A pictorial overview of the FSM is given in Figure 2. As we developed our strategy more fully, this became the behavior of the attacker (see Section 11).

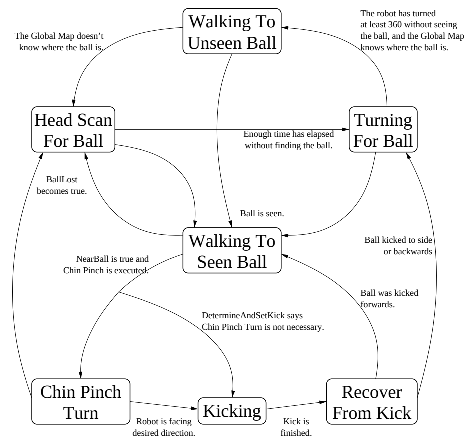

Fig. 2. The finite state machine that governs the behavior of the attacking robot.

11 Coordination

In this section we describe our initial and eventual solutions to coordinating multiple soccer-playing robots.

Our first efforts to make the players cooperate resulted directly from our attempts to play games with 8 players. Every game would wind up with six robots crowded around the ball, wrestling for control. At this point, we only had 2 weeks before our first competition, and thus needed a solution that did not depend on localization, which was not yet functional. Our solution was a process we called *Dibs*, which used local communication among the robots to determine which is closest to the ball, and thus which should go after it.

During the last week or so before RoboCup 2003, we developed a new, more sophisticated coordination strategy. In particular, it takes advantage of both localization and global maps. This strategy uses a dynamic system of roles to coordinate the robots. In this system, each robot has one of three roles: *attacker*, *supporter*, and *defender*. The goalie does not participate in the role system. For full details on the player's behaviors, see our technical report [1].

12 UT Assist

During the course of our development, we developed a valuable tool to help us debug our robot behaviors and modules. This tool, which we called UT Assist, allowed us to experience the world from the perspective of our Aibos and monitor their internal states in real-time. One use of UT Assist is to extract debugging data from the Aibos, including visual output, localization output, and behavior output. We also use it for vision calibration.

13 The Competition

The Seventh International RoboCup Competition was held in Padova, Italy from July 2nd to 9th, 2003. 24 teams competed in the four-legged league, eight of which, including us, were teams competing at the international event for the first time. The 24 teams were divided into four groups of six for a round robin competition to determine the top two teams which would advance to the quarter-finals. The teams in our group were the German Team from University of Bremen, TU Darmstadt, Humboldt University, and University of Dortmund, all in Germany; ASURA from Kyushu Institute of Technology and Fukuoka Institute of Technology in Japan; UPennalizers from the University of Pennsylvania; Essex Rovers from the University of Essex in the UK; and UTS Unleashed! from the University of Technology at Sydney. Essex ended up being unable to compete and dropped out of the competition. The results of our four games are shown in Table 3.

Meanwhile, we made sure to arrange some practice games in Italy. The results are shown in Table 4.

Opponent	**Score (us-them)**
UTS Unleashed!	1–7
German Team	0–9
UPennalizers	0-6
ASURA	1-4

Fig. 3. The scores of our four official games at RoboCup.

Opponent	**Score (us-them)**
CMU	2–2
U. Washington	0–1
Team Sweden	3–0
U. Washington	0–4
Metrobots	3–1
Team Upsalla	4–0

Fig. 4. The scores of our six unofficial games at RoboCup.

Based on all of our practice matches, we seemed to be one of the better new teams at the competition. We were in a particularly hard group, but we were able to compete at a reasonable level with even the best teams (despite the lopsided scores).

The Challenge Events

In addition to the actual games, there was a parallel "challenge event" competition in which teams programmed robots to do three special-purpose tasks:

1. Locating and shooting a black and white (instead of an orange) ball;
2. Localizing without the aid of the standard 6 colored field markers; and
3. Navigating from one end of the field to the other as quickly as possible without running into any obstacles.

Given how much effort we needed to put in just to create an operational team in time for the competition, we did not focus very much attention on the challenges until we arrived in Italy. Nonetheless, we were able to do quite well, which we take as a testament to the strengths of our overall team design.

On the first challenge, we finished in the middle of the pack. Our robot did not succeed at getting all the way to the black and white ball (only eight teams succeeded at that), but of all the teams that did not get to the ball, our robot was one of the closest to it, which was the tie-breaking scoring criterion. Our rank in this event was 12th.

In the localization challenge, the robot was given five previously unknown points on the field and had to navigate precisely to them without the help of the beacons. Our robot used the goals to localize initially, and then relied largely on odometry to find the points. Our robot successfully navigated to only one of the five points, but the large majority of teams failed to do even that. Our score was sufficient to rank us 5th place on this event. Unfortunately we were disqualified on a technicality. We had initially programmed the robot with the wrong coordinate system (a mere sign change). Rather than running the robot toward mirror images of the actual target points, we decided to fix the code and accept the disqualification.

Finally, on challenge 3, the robot was to move from one side of the field to the other as quickly as possible without touching any of seven stationary robots placed in previously unknown positions. Our robot used an attraction and repulsion approach which pulled it toward the target location but repelled it from any observed obstacle. The resulting vector forces were added to determine the instantaneous direction of motion for the robot. Since speed was of the essence, our robot would switch to our fastest gait (ParamWalk) when no obstacles were in sight. A slower gait that allowed omnidirectional movement (SplineWalk) was used for all other movement.

Our robot was one of only four to make it all the way across the field without touching an obstacle, and it did so in only 63.38 seconds. The German team succeeded in just 35.76 seconds, but the next closest competitor, ARAIBO, took 104.45 seconds. Thus we ranked 2nd in this event.

Officially, we finished 13th in the challenge events. However the unofficial results, which did not take into account our disqualification in event 2, nor one for the University of Washington, placed UT Austin Villa in fourth place. Given that 16 of the 24 RoboCup teams were returning after having competed before, and several of them had spent more effort preparing for the challenges than we had, we were quite proud of this result and are encouraged by what it indicates about the general robustness of our code base.

14 Conclusions and Future Work

The experiences and algorithms reported in this paper and the accompanying technical report comprise the birth of the UT Austin Villa legged-league robot team. Thanks to a focussed effort over the course of 5 1/2 months, we were able to create an entirely new code base for the Aibos from scratch and develop it to the point of competing in RoboCup 2003.

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 now have a fully functional research platform and are ready to use it for investigations in various directions. One current effort involves automatically learning to improve the walking parameters; other investigations are likely to begin shortly.

Overall, developing a competitive RoboCup soccer team in such a short period of time 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

Thanks to other members of the original class from the spring of 2003, all of whom contributed to the design process of our team. Particular thanks to Nabeel Ahmed, Pradeep Desai, Gregory Jay, Chris Lundberg, and Aniket Murarka. The authors would also like to thank Sony for developing the robots and sponsoring the four-legged league, 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.

References

1. 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. The University of Texas at Austin, Department of Computer Sciences, AI Laboratory Technical Report UT-AI-TR-03-304, 2003. `http://www.cs.utexas.edu/home/department/pubsforms.shtml`
2. Manuela Veloso, Scott Lenser, Douglas Vail, Maayan Roth, Ashley Stroupe, and Sonia Chernova. CMPack-02: CMU's Legged Robot Soccer Team. 2002. `http://www.openr.org/robocup/code2002SDK/CMU/cmu_teamdesc.pdf`.
3. Spencer Chen, Martin Siu, Thomas Vogelgesang, Tak Fai Yik, Bernhard Hengst, Son Bao Pham, and Claude Sammut. The UNSW RoboCup 2001 Sony Legged League Team. 2001 `http://www.cse.unsw.edu.au/ robocup/2002site/`
4. H.-D. Burkhard, U. Düffert, J. Hoffmann, M. Jüngel, M. Lötzsch, R. Brunn, M. Kallnik, N. Kuntze, M. Kunz, S. Petters, M. Risler, O. v. Stryk, N. Koschmieder, T. Laue, T. Röfer, Spiess, A. Cesarz, I. Dahm, M. Hebbel, W. Nowak, J. Ziegler. 2002. German Team 2002. `http://www.robocup.de/germanteam/GTeng/index.html`.
5. Control Tutorials for Matlab: PID Tutorial. `http://rclsgi.eng.ohio-state.edu/matlab/PID/PID.html`.