The University of Pennsylvania Robocup 2005 Legged Soccer Team

Gilad Buchman, David Cohen, Paul Vernaza, and Daniel D. Lee

General Robotics Automation, Sensing and Perception (GRASP) Laboratory
University of Pennsylvania, Philadelphia, PA 19104
WWW home page: http://www.cis.upenn.edu/robocup

Abstract. This paper presents the software design for a team of soccer playing robots developed at the Univ. of Pennsylvania for the 2005 Robocup competition. The software was a port and slight modification from the 2004 competition code. Lower level sensory and motor functions were first prototyped in Matlab. High level behaviors and team coordination were implemented using an embedded Perl interpreter. These developments allowed the development of the team that ultimately resulted in a quarterfinal finish at the international competition. Also presented is the system for extreme locomotion that was demonstrated at the 2005 open challenge, which earned first place for the UPennalizers.

1 Introduction

This paper presents the software structure for a team of soccer playing Aibo robots developed at the Univ. of Pennsylvania. The code itself was a lightly modified port of the 2004 game code, and implemented improvements to vision, motion, obstacle detection, and shock compensation.

Also presented is the system for extreme locomotion that was developed in the months before the 2005 competition and demoed at the Open Challenge. We hope that the system employed will find its way into regular gameplay, as it fundamentally more robust than the methods for locomotion currently employed.

2 Software Architecture

The software architecture for the robots is shown in Figure 1. This archicture is implemented by compiling several OPEN-R modules using the Sony OPEN-R API [1]:

ROBOTCOM.BIN Main module that includes the embedded Perl interpreter, vision and sensor interfaces, walk routine, and world model.

EFFECTOR.BIN Module that accepts arrays of joint angles and sequences the motors and other effectors such as LED's.

SNDCOMM.BIN Module that can play and record PCM sound samples.

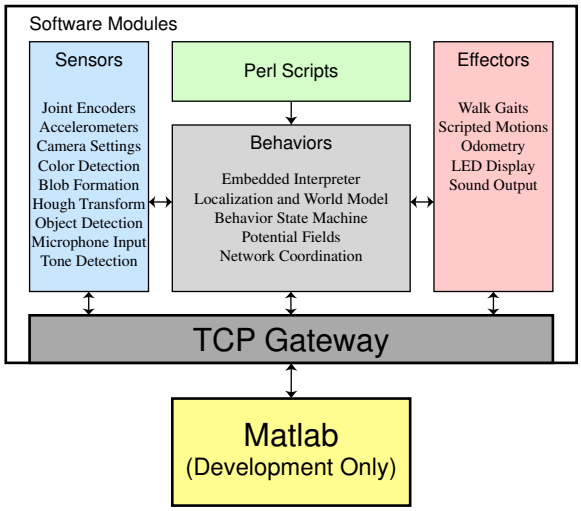


Fig. 1. Univ. of Pennsylvania software architecture for sensory and motor processing, as well as implementation of various behaviors for the Robocup competition.

In order to simplify development, all interprocess communications are performed by passing a shared memory matrix structure between the modules:

```
template <class T>
class ShmArray : public OShmPtrBase {
...
}
```

This structure allows for efficient communication as well as unifying the input and output interfaces of every module that is developed. Modules can also easily pass information to modules on other robots by sending these data structures through the supplied TCPGateway interface.

2.1 Matlab Interface

Rapid development of lower level sensory and motor functions is facilitated by using Matlab to first prototype the algorithms [2]. Matlab is a high-level numerical scripting language that allows complex algorithms to be coded in a small number of lines. Since Matlab uses matrices as its basic data structure, we developed several functions that allowed Matlab to send and receive matrices over the wireless network to the various OPEN-R modules running on the robots.

Motions For example, the joint angles for a new kick are first computed using one of a set of Matlab routines. The routines typically employ inverse kinematics to effect side kicks, bumps, and even combinations of the two in order to quickly

generate a full motion. This results in a $16 \times N$ matrix in Matlab that describes the motion of each of the 3 joints in the 4 legs as well as the 4 head angles (pan, both tilts, mouth). This matrix is then sent to the Effector module on a running robot to be actuated in real-time. In this way, dozens of kicks can be created and tested per hour.

Vision Another example of Matlab's utility is in image processing. Not only are the colors segmented using Matlab, but even the camera's geometric distortions and color inaccuracies can be at least partially corrected, often using freely available software.

An example of a vision problem being solved by Matlab was our correction of the geometric color distortion in the ERS-7 cameras. We found that the distortion in each of the Y,U, and V spaces could be modeled by a quadratic:

$$Y_{actual} - Y_{observed} = A(Y_{actual} - Y_0)r^2 \tag{1}$$

Where A and Y_0 are constants and r is the distance from the center of the image. The constants could be solved by applying Matlab's regression capabilities to a collection of camera images imported to the workspace.

2.2 Perl Interpreter

High level behaviors are implemented using an event-driven state machine. To facilitate rapid development, we decided to implement these behaviors using Perl scripts rather than in the native C++ programming environment. Perl is a high-level "kitchen-sink" programming language that incorporates an optimized interpreter that compiles scripts into intermediate opcodes for efficient performance [3].

To enable this capability on the robots, we embedded a Perl interpreter to run on the robots [4]. Unfortunately, it is not possible to simply configure Perl for the Aibo robots using the normal build process. Due to the limited operating system environment on the robots, our Perl port is based upon the scarcely-documented microperl build in the 5.8.0 release of Perl. The interpreter is built as a static library that is linked into the ROBOTCOMM module. The resulting library references several functions such as "fork" in the standard UNIX API that are not available on the robots, so several dummy functions also needed to be implemented in order for the module to be linked properly.

To enable the Perl interpreter to interact with the sensory and motor routines implemented in the OPEN-R modules, the interpreter was extended so that it is able to call C functions exported by the modules. A small language called XS available in the Perl distribution was used to automate this task [1]. Through this interface, a Perl script is able to access data variables in the OPEN-R routines and activate functions that set status LED's, sequence motions and walk routines, and communicate with other robots to coordinate team behaviors.

[1] see the perlxs UNIX man page for more information

The use of the Perl interpreter allows us to develop and modify behaviors for the robots using a standard high-level scripting language in the robots' runtime environment. Thus, we could test a new behavior by sending a new script to robots over the network and execute them on demand without having to reboot the robots. Additionally, due to Perl error handling, a bad script will rarely result in a system crash. This dramatically reduces the severity and duration of downtime during development time.

3 Vision

Most of the algorithms used for processing visual information from the robots' CMOS cameras are similar to those used by other teams in the past [5]. Since fast vision is so crucial for the robots' behaviors, these algorithms were implemented from scratch to gain as much performance speed as possible. This speed was critical during the competition since we used several robots with older, slower processors during the preliminary rounds.

3.1 Object Recognition

The main processing pipeline involves segmenting the highest-resolution color images from the camera, forming connected regions, and recognizing various objects from the statistics of the colored regions. The color segmentation routine classifies individual pixels in the image based upon their YCbCr values. Based upon a number of training images, a Gaussian mixture model is used to segment the YCbCr color cube into the following colors:

- Orange (Ball)
- Pink (Marker)
- Cyan (Marker and Goal)
- Yellow (Marker and Goal)
- Blue (Robot)
- Red (Robot)
- Green (Field)
- White (Lines and Border)

Once the pixels in the image are classified according to their colors, they are merged into connected components using techniques that have been previously described [5]. This is accomplished by first run-length encoding the images, and then merging these run-lengths into connected regions.

After the image has been segmented into these connected regions, the regions are classified into relevant objects by comparing various image statistics of the regions. These statistics include the bounding box of the region, the centroid location, and the major and minor axes lengths. In this manner, the location of the ball, markers, and goals are detected. There was not enough time before the competition to implement a robot detection algorithm, and our behaviors are based solely on the reported locations of our robots and the ball.

3.2 Line Recognition

One new aspect of our visual processing routines is the detection of field lines. This decreased the need for our robots to actively search for field markers, enabling them to chase the ball more effectively. The first step in line identification is to find white pixels in the medium resolution camera images that neighbor pixels of field green color. Once these pixels are located, a Hough transform is used to search for relevant line directions.

In the Hough transform, each possible line pixel (x, y) in the image is transformed into a discrete set of points (θ_i, r_i) which satisfy:

$$x \cos \theta_i + y \sin \theta = r_i \tag{2}$$

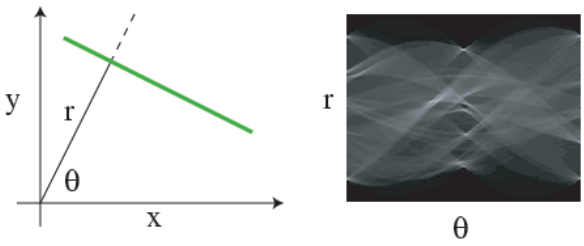


Fig. 2. Hough tranformation for field line detection in images.

The pairs (θ_i, r_i) are accumulated in a matrix structure where lines appear as large values as shown in Figure 2. To speed the search for relevant lines, our implementation only considers possible line directions that are either parallel or perpendicular in the field to the maximal value of the accumulator array. Once these lines are located, they are identified as either interior or exterior field lines based upon their position and then used to aid in localization.

4 Localization

The problem of knowing the location of the robots on the field is handled by a probabilistic model incorporating information from visual landmarks such as markers, goals, and lines, as well as odometry information from the effector module [6]. Recently, probabilistic models for pose estimation such as extended Kalman filters, grid-based Markov models, and Monte Carlo particle filters have been successfully deployed. Unfortunately, complex probabilistic models can be difficult to implement in real-time due to a lack of processing power on board the robots. We address this issue with a new pose estimation algorithm that incorporates a hybrid representation that reduces computational time, while still providing for a high level of accuracy. This new algorithm models the pose uncertainty as a distribution over a *discrete* set of heading angles and *continuous*

translational coordinates. The distribution over poses (x, y, θ), where (x, y) are the two-dimensional translational coordinates of the robot on the field, and θ is the heading angle, is first generically decomposed into the product:

$$P(x, y, \theta) = P(\theta)P(x, y|\theta) = \sum_i P(\theta_i)P(x, y|\theta_i) \tag{3}$$

We model the distribution $P(\theta)$ as a discrete set of weighted samples $\{\theta_i\}$, and the conditional likelihood $P(x, y|\theta)$ as simple two-dimensional Gaussians. This approach has the advantage of combining discrete Markov updates for the heading angle with Kalman filter updates for the translational degrees of freedom.

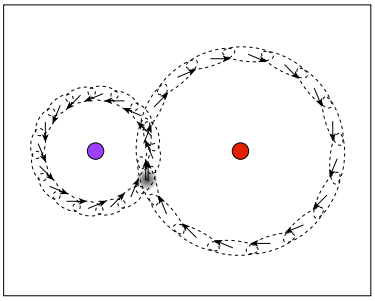


Fig. 3. Hybrid probabilistic representation used for localization.

When the algorithm is implemented on the robots, they are able to quickly incorporate visual landmarks and motion information to consistently estimate both the heading angle and translational coordinations on the field as shown in Figure 3. Even after the robots are lifted (kidnapped) by the referees, they are able to quickly relocalize their positions when they see new visual cues.

5 Motion

The motion of the robots is controlled by a parameterized walk routine in addition to predetermined scripted motions. The implementation of the walk routine is similar to other teams, in that inverse kinematics is used to generate the motion of the robot feet in a rectangular pattern relative to the body [7]. The four legs are phased according to a dynamic trot gait with a 50% duty cycle. The parameters for the walk were optimized by tracking the motion of the robot using a magnetic tracker [8], and gradually modifying the parameters to maximize speed and stability.

5.1 Kicks

The kicks are fully scripted motions, each starting from a standard position that is compatible with the end of a walk cycle. Matlab was used to tweak the joint angles in order to develop a set of kicks that could be used to kick the ball forward as well as sideways using both the legs and head. The criteria for developing kicks were as follows:

1. They shouldn't damage the robot or turn it off under any circumstances.
2. They had to be very quick to execute (typically under 2 seconds).
3. They featured complete ball control at every instant until the ball was released.
4. If they failed to release the ball, they should return to a state where the robot maintained control of the ball.
5. They were optimized for either power or accuracy.

6 Behaviors

The behaviors of the robots on the field are implemented using an object-oriented Perl script. This script incorporated an event-driven state machine that generates different actions depending upon one of possible four roles that the robot was assigned:

Attack The attacking robot goes directly to the ball, as long as it wasn't in the defensive penalty box.

Defend The defending robot positions itself between the ball and defensive goal area.

Support The supporting robot positions itself in an area away from the ball that would assist in scoring.

Goalie The goalie stays near the defensive goal to clear the ball when it comes close.

6.1 Potential Fields

Potential fields are used in all the roles to guide the robots to their optimal positions on the field. As shown in Figure 4, the positioning of the robots are dependent upon the ball location relative to predetermined field locations. For example, the optimal position for the Support role is on an intermediate point between the ball and an offensive field position. Similarly, the Defend role is situated on an intermediate point between the ball and a defensive field position. In Robocup 2005, the offensive field position was on the centroid of the opponent's half, and, in order not to be in the way of the Attacker, a potential field was used to repel the Supporter from the ball when the ball was too close. The Defender was situated on the y-centered line of its team's half but followed the ball horizontally along the field. With the exception of the Goalie, a repulsive potential field in the defensive penalty area prevented the robots from becoming illegal defenders. The potential fields were also used to direct the robots to their initial kick-off positions.

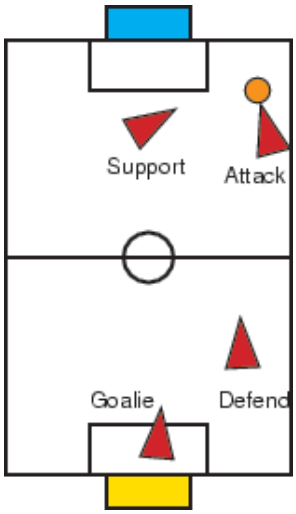


Fig. 4. Potential fields draw robots to their relevant field positions around the ball according to their roles.

6.2 Role Switching

Although the Goalie remains statically assigned, the three robots that are initially assigned the Attack, Support, and Defend roles can fluidly switch roles. The role switching mechanism is based on the GermanTeam Dynamic Role Assignment [13]. The three field players negotiate who is going to be the Attacker, based on the Estimated Time of Arrival (ETA) to the ball. Each robot compute its own ETA based on the following code:

```
$eta = 3*$Ball->distance
    + 100*$Ball->uncertainty
    + 750*$not_attack; # approx 1 sec hysteresis
if (abs($targetA) > deg_to_rad(90)) { # penalize coming back on back
    $eta += 1000*(abs($targetA) - deg_to_rad(90));
}
```

The ETA is a weighted sum of the distance from the player to the ball plus an uncertainty measure that increases with the time the player is unable to see the ball. An hysteresis term is added to prevent continuous switching between two players that are approximately the same distance from the ball. Another advantage is given to a player that is all ready facing the opponent goal, since a dog that is turned away from the goal will need to turn with the ball, a maneuver that is costly in terms of time and accuracy of odometry.

The players send each other their own ETA, and the player with the smallest ETA takes the Attacker role by sending a message to his teammates. If the teammates believe that the Attacker ETA is smaller than their own ETA, they

will chose one of the remaining rules. The Supporter is then chosen based on the Y position on the field of the players: the one closer to the opponent goal becomes the Supporter and the other becomes the Defender. If only two players are on the field (the third player is penalized), the player that is not the Attacker will take the Defender role. Using this scheme we guarantee that we will always have an attacker and a defender. In some cases due to network delays or mis-localization of the robots, two robots could fight over the Attacker role but there are no situations where no one is going after the ball. However, this conflict in the role switching rarely occurred during the competition, and if it did it was only for a very short period of time (less then 2 seconds.)

6.3 Time Synchronization

The players communicate their ETAs over the WLAN. In order to account for network delays a synchronization mechanism was implemented. When a player j sends a message at time t, it attaches to it its local timestamp. When player i receives this message (sent using broadcast) it saves the sender's timestamp alongside its own local timestamp. The difference between these two timestamps is equal to the clock difference between the robots plus the time it took the message to travel between the robots due to network delay.

$$TotalDiff^t_{j,i} = LocalTS^t_i - LocalTS^t_j = NetDelay + ClockDiff_{i,j} \quad (4)$$

The sender also sends with every message the latest time difference between itself and its other teammates. Player i receives the message and updates its belief about the network delay and the difference between its clock and the sender clock.

$$ClockDiff_{i,j} = -ClockDiff_{j,i} \Rightarrow \quad (5)$$

$$NetDelay = \frac{TotalDiff^t_{j,i} + TotalDiff^{t-1}_{i,j}}{2} \quad (6)$$

$$ClockDiff_{i,j} = TotalDiff^t_{j,i} - NetDelay \quad (7)$$

Since the Network Delay changes all the time and the actual clock differences are unknown and can change during the game if one of the robots is rebooted, player i updates its clock difference from player j by:

$$RealClockDiff_{i,j} = \alpha(ClockDiff_{i,j}) + (1-\alpha)(ClockDiff_{i,j}), 0 \leq \alpha \leq 1. \quad (8)$$

6.4 Kick Selection

A cost function was designed for kick selection. Each time a player was to perform a kick, it calculated what is the most effective kick for that given situation given the angle and distance to the opponent goal. A set of kicks were developed optimized for either power or accuracy and for different angles. Statistics were collected for each kick in the set, including the median distance and median angle

of the kick and their standard deviations. A chosen set of kicks were written to a file, which was automatically loaded and parsed by the robot. This way changing sets of kicks for different stragegies was very easy. The cost function calculated the utility of using each kick in the kick set, the kick with the minimal cost was chosen and performed. For maximizing performance the robot aligned itself in such a way that the desired angle would match the kick angle. If in order to align the robot needed to perform a big turn with the ball, then after the turn the robot recalculated a desired kick. The following terms were used in calculating the cost:

- Number of frames it takes to perform the kick.
- Difference between the desired distance and the kick's mean distance.
- Difference between the desired angle and the kick's mean angle.
- Standard deviation of the kick's distance.
- Standard deviation of the kick's angle.
- Penalize kick that overshoots.
- Penalize kick that requires turning to the opposite direction from the desired kick direction.

The weights on the on the different terms need tuning and should be learned offline. The statistics of the kicks depends heavily on the specific field.

7 Extreme Locomotion

A major research goal in the field of robotics is to develop capacity for "extreme" walks while keeping computations tractable. In order to create a class of agile legged robots capable of assisting troops in the field, DARPA (Defense Advanced Research Projects Agency) has issued solicitation BAA 05-25, which has as its primary objective the development of learning techniques that will permit a quadruped to walk across an obstacle-strewn terrain.

This study was designed to create a method for making a Sony Aibo walk up a 1-inch step consistently, in preparation for the associated DARPA project. The Sony Aibo was selected because, although the DARPA-supplied robot is not currently available, the Aibo has a similar size and structure. Project activities focused on finding ways to apply simple models for generating walks onto the step. The rationale behind developing the simple models was that the models would eventually be machine learned. To address the present objective, hand-tuning alone was sufficient.

As a result of the work completed in this study, both a 35 mm and a 50 mm step were scaled successfully by the Sony Aibo. The method used will be explained fully in the following sections. Areas requiring future work – such as footfall order determination and primitive switching – will also be described.

7.1 Overview of Step Method

The method used for scaling the obstacles mixed a potential field formulation for torso placement with pre-set geometries for footpaths during steps. The torso

was subjected to rigid-body translation and rotation from fields that were determined by leg extension, leg orientation, and body balance. At each frame, the step function would attempt to find a local minimum in the potential field by iteratively "riding" the field, and the torso would move toward this minimum as quickly as possible. The feet were pre-sequenced to place themselves at certain end positions in turn and, as such, were not affected by the fields.

Interspersed between the footfalls were periods in which the robot would shift its center of balance over the three feet that were to be stationary over the next cycle. This insured that the robot did not immediately fall over when the next foot was raised.

A set of primitives (namely, front-up move-forward and rear-up) were found that allowed the robot to more effectively negotiate large obstacles. By separating the larger task of obstacle-scaling into these primitives, field parameters could be tuned more specifically and different footfall sequences could be determined for different situations.

Part 3 will explain the types of potential fields exerted on the torso. Part 4 will explain the rigid-body dynamics used to evaluate the effect of these fields. Part 5 will briefly explain the implementation of trapezoidal step paths. Part 6 will discuss the various primitives' parameters and the footfall sequences for these primitives.

7.2 Explanation of Potential Fields

Early on, it was recognized that it would be necessary to have the torso displace and pitch in order to keep the robot's feet stable and within their configuration spaces. The original study proposal [9] called for the use of explicit primitives, like "pitching" and "climbing", to accomplish these tasks. This, however, would have required complex tuning and switching. The use of potential fields allowed vast simplification.

There were three main types of fields applied to the torso: radial leg fields, angular leg fields, and a balance field. The leg fields were applied to the hips of each leg, and resulted in both translation and rotation for the torso. The balance field was applied at the center of mass of the robot, and thus allowed for only translation. As will be explained in section 4, these fields determined instantaneous momentum, not force.

The radial leg and angular leg potential fields were modeled to keep the feet within their configuration spaces. The balance field was added to keep the dog from falling over. These fields are explained in greater detail below.

Radial Leg Fields The radial leg fields, depicted in Figure 5 as springs, were nicknamed "shock-absorber fields" since their effect was similar to attaching shock absorbers between the hip and foot of each leg. Stretching a leg near the limit of its configuration space tugged the torso after it; likewise, bringing a leg close in to the torso tended to push the torso away. The purpose of these fields was to avoid the imaginary angles and odometry problems associated with attempts to position the leg beyond the singularities.

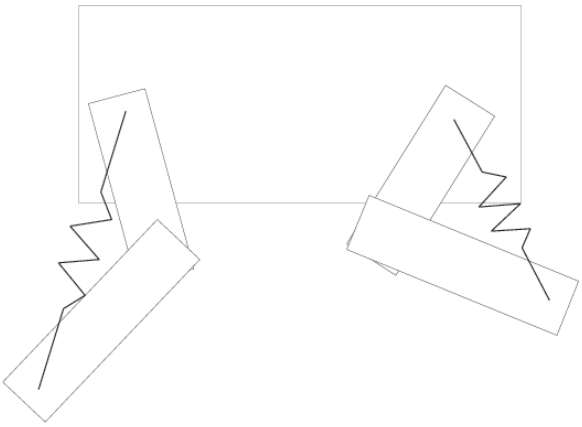


Fig. 5. Potential fields act as springs, forcing the torso relative to the robot's feet.

The characteristic force-displacement response, though, was quite dissimilar to the classical linear spring. The study showed that it was advantageous to create a large "dead zone" within which the robot was unaffected by the shock-absorber field. Therefore, the field was modified to be the product of a line with the sum of two exponentials:

$$Force = A(x - len_{nat})(\exp(B(x - len_{max})) + \exp(C(len_{min} - x))) \quad (9)$$

In the function above, 'x' is the extension between the hip and foot. Note that, by modulating the constants B and C, the operator can make the force-displacement profile asymmetric – an advantage not afforded by other functions with "dead zones" (such as hyperbolic sine or high-degree polynomials). An example of the force-displacement profile of this function, with real parameters taken from the code, is given in Figure 6. Note that, since the constants B and C are equal, the asymmetry is not fully employed.

Angular Leg Fields The radial leg fields served to keep the legs within their extension limits. However, using radial fields alone led to situations in which the robot would try to position its legs outside their angular limits. Therefore, it was necessary to add in angular leg fields, so that extreme angles could be avoided.

The angular fields were decomposed into two fields for each leg: a "flap" field, and a "swing" field. Figure 7 shows the axis convention employed for the robot. Figure 8 and Figure 9 show the "swing" and "flap" angles, respectively. The axis convention is consistent across the three figures.

The "swing" angle, marked α on Figure 8, is the angle between the projection of $\mathbf{R}$ (the vector from the hip to the foot) on the Y-Z plane and the $-\mathbf{Z}$ vector. The sign convention used made the angle depicted negative, so it is marked as such. Likewise, the "flap" angle, marked β on Figure 9, is the angle between the

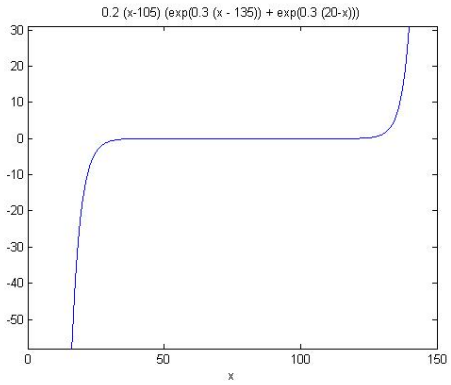


Fig. 6. Linear-exponential force profile of the radial leg field.

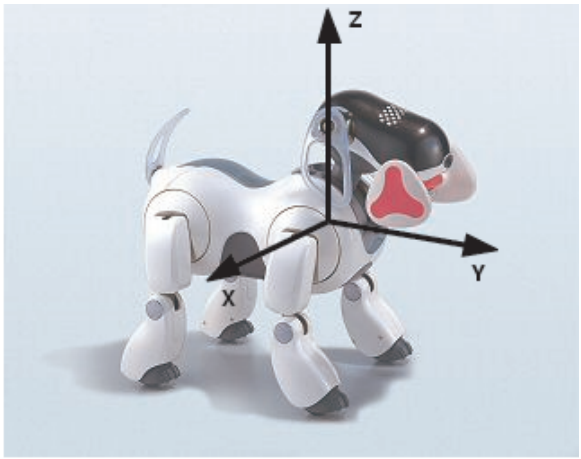


Fig. 7. Axis convention employed for the Sony Aibo.

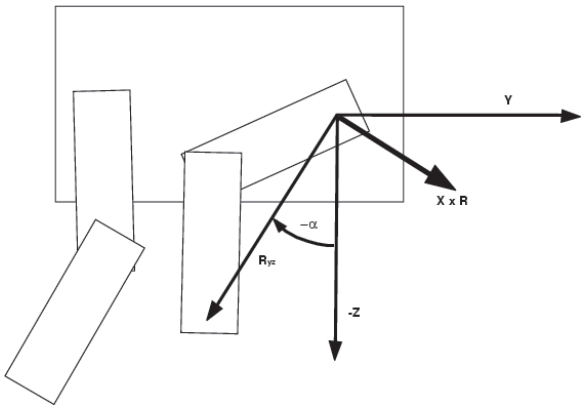


Fig. 8. Y-Z Plane of Robot with "Swing" Angle Indicated (Front Right Leg).

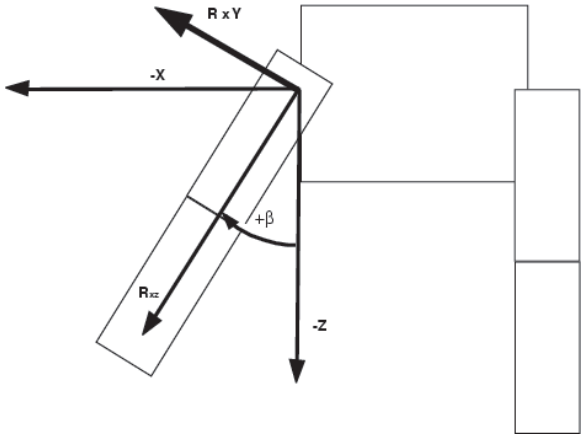


Fig. 9. X-Z Plane of Robot with "Flap" Angle Indicated (Front Right Leg).

projection of **R** on the X-Z plane and the –**Z** vector (the sign convention makes the angle depicted positive).

For each field a force function similar to those used by the radial fields was implemented, since it was found that a "dead zone" was advantageous with the angular fields as well. Thus, both "flap" and "swing" had a natural angle, a minimum angle, and a maximum angle associated with it.

The direction along which the force was applied to the torso, however, was different between the two fields and distinct, obviously, from the radial fields as well. The direction for the force application due to the "swing" field was along the cross product of the **X** vector and **R**, and the direction for the force application due to the "flap" field was along the cross product of **R** and the **Y** vector. Both directions are denoted on their respective figures with the heavier arrows. The rationale behind using the cross products for the directions of application was that they would provide the most efficient angular change without radial change, given that the foot position remained constant.

Balance Field The final potential field force implemented on the torso was a balance field. In order to keep the robot statically stable (the gaits generated were all static, not dynamic, gaits) it was necessary both to keep at least three legs on the ground at a time and to ensure that the robot's center of mass was within the polygon determined by the feet on the ground. Therefore, a field was implemented that drew the torso towards the centroid of the polygon determined by the planted feet.

Figure 10 shows a schematic of the balance field for the case where the front right foot is off the ground. The planted feet are shown and labeled (no legs are shown, since they are unnecessary for the calculation of the force or direction). The force runs from the centroid of the torso to the centroid of what is, in this case, a triangle whose vertices are the positions of the planted front left foot, rear left foot, and rear right foot.

The magnitude of the force is calculated using a hyperbolic tangent:

$$Force = A \tanh(Bx) \tag{10}$$

Variable 'x' is the magnitude of the vector from the torso centroid to the foot centroid.

7.3 Explanation of the Rigid-Body Reactions

The torso acted as a 4-DOF rigid body, limited to x-y-z translation and pitching. The "forces" at the hips of the four legs and at the center of mass of the torso were summed by standard techniques to provide a net resultant force and a net moment (it was assumed that the center of geometry of the torso was also the center of mass). The way the force and the moment were applied, however, were different from the standard convention:

$$Force = (mass)(velocity) \tag{11}$$

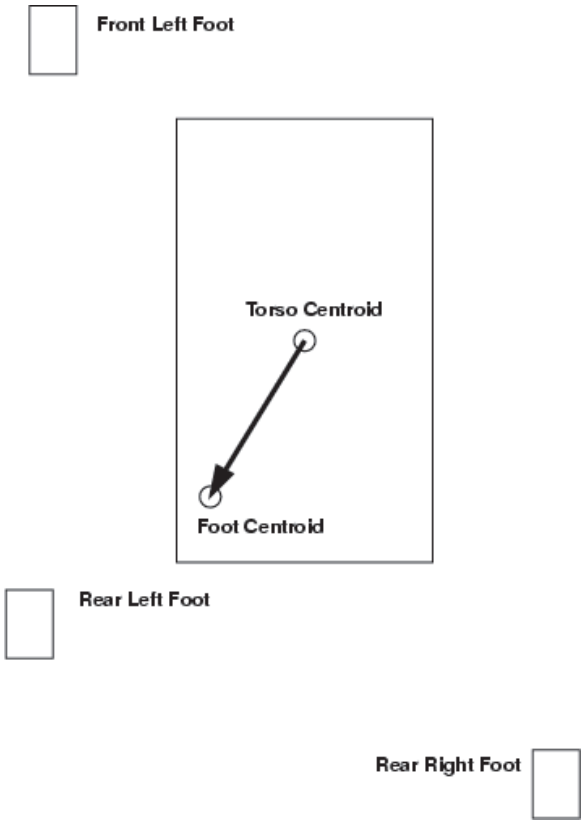


Fig. 10. The Balance Field (Bird's Eye View).

$$Moment = (Moment\ of\ Inertia)(Angular\ Velocity) \tag{12}$$

Thus, the fields implemented are probably most appropriately called "momentum fields". In the interest of simplicity, these "momentum fields" were implemented instead of "force fields". Since the function was aiming for the local minimum, it didn't seem to make a difference whether velocity was integrated from accelerations or not.

The mass and moment of inertia were defined as parameters in the step function. Since the system was 4-DOF, the moments of inertia corresponding to yaw and roll were effectively set to infinity. There were also caps on the maximum angular and translational velocities that could be achieved.

7.4 Explanation of the Leg Paths

The footpaths used were inspired by the trapezoidal steps simultaneously developed by Newcastle University and the University of New South Wales for the Robocup 2003 competition [10], [11]. The rationale behind using a trapezoidal step in robot soccer was the speed advantage. Disengaging the claw of the Sony Aibo from the field material yielded a speed increase of about 10%. While, for the case at hand, speed was not important, it was crucial that the robot's feet not be hindered by the material. Consequently the trapezoidal method was employed.

A schematic of the trapezoidal step is provided in Figure 11. There was no bottom stroke to the step. Traversal was accomplished by torso repositioning across different foot positions. In fact, the only way to estimate the end position of the robot was to sum the frame-by-frame torso displacements. The parameters ϕ, γ, and 'minimum clearance' determine the shape of the step. Additional parameters determine the speed with which the step is executed in the Rise, Traverse, and Lower Stages.

In the case that there is an x-displacement in the step, the path must skew in the X-Y plane. In this case, both the Rise and Lower Stages have no component in the x-direction. This is to ensure that there is still a clean disengagement from the field material. Figure 12 gives a top down view of what a skewed step might look like.

7.5 Primitive Parameters and Footfall Sequences

Once all the fields were in place, the final challenges were to find a suitable sequence of footfalls to climb the step and to tune the field parameters.

The footfall sequence was a list of 4 element vectors, which denoted the x, y, and z displacements of the step as well as an index of the leg that was stepping. The step function assumed that the foot was just resting on a surface at the end of each footfall. The function was completely open-loop and this "foot resting" condition was necessary so that the robot's orientation could be accurately calculated. Unfortunately, it was the general case that for the scaling footfalls some experimentation was required to find the exact z-displacement

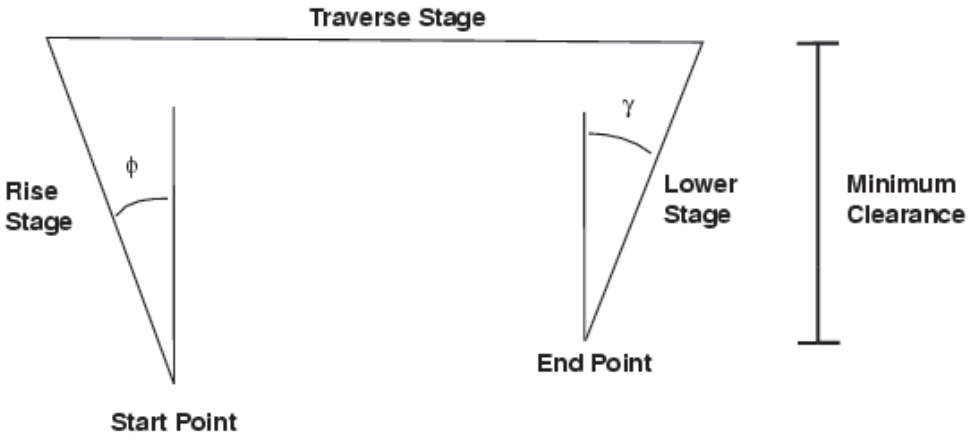


Fig. 11. Trapezoidal Step Path (Side View).

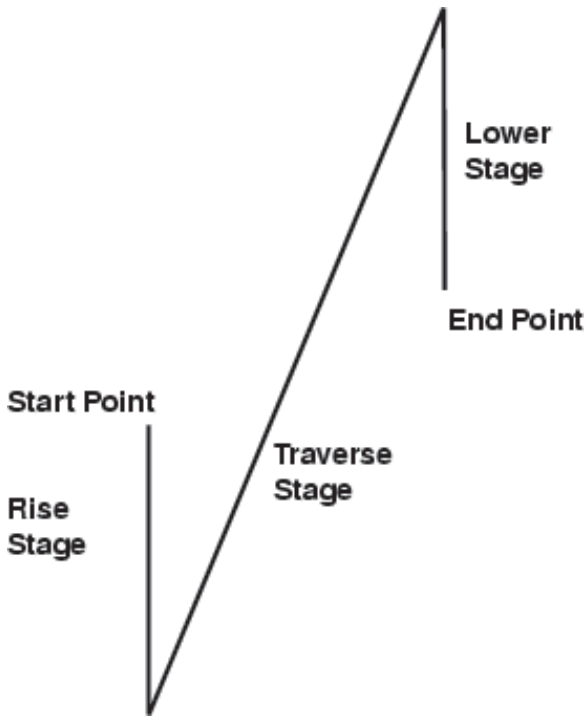


Fig. 12. Trapezoidal Step Path (Bird's Eye View).

that would allow the foot to just set down. This was because the Sony Aibo has large plastic casings around its forepaws, making it very difficult to get exact measurements for the point of contact.

The method for parameter tuning, likewise, was trial and error: any set of parameters that led the robot to try to effect positions outside its configuration space was deemed unsuitable, as was any set that resulted in the robot's losing its balance. Tuning the parameters thus proved exceptionally difficult, as the robot seemed to inevitably violate at least one of the two criteria. This was remedied, however, by the development of a shifting foot-cycle: while traditional non-extreme static gaits use 4 step foot-cycles, it was found that an irregular cycle led to a large qualitative improvement in the robot's performance. Once the irregular cycle was implemented, it was not long before the parameters were tuned appropriately, and the robot was able to successfully scale a 35 mm step.

The next step after scaling a 35 mm, or 0.25 L (L = leg length) step was to scale a 50 mm (0.35 L) step. Unfortunately, the advances that resulted in the scaling of the smaller step were not quite sufficient to permit scaling the larger one. In general, parameters that allowed good performance for one portion of the step led to significant failure in other portions.

The solution was to simply use different parameters for different parts of the step. The three primitives that resulted (front-up, move-forward, and rear-up) each had its own footfall sequence and parameters – thus the irregular cycle used to scale the 35 mm step became a concatenation of three distinct cycles.

Once the primitives were set in place, it was a simple matter of re-tuning the parameters and footfalls to scale the new step. The footfall sequence remained unchanged from the 35 mm step to the 50 mm step. Indeed, the ease with which the robot was retuned suggests the robustness of the field method, although it should be noted that the motion returned from the front-up primitive had to be smoothed and sped up. This was, however, a problem for finding the minimum of the force field and not a problem inherent in the fields themselves. Figures 9 and 10 give information on the primitive parameters and footfall sequences employed for the 50 mm step.

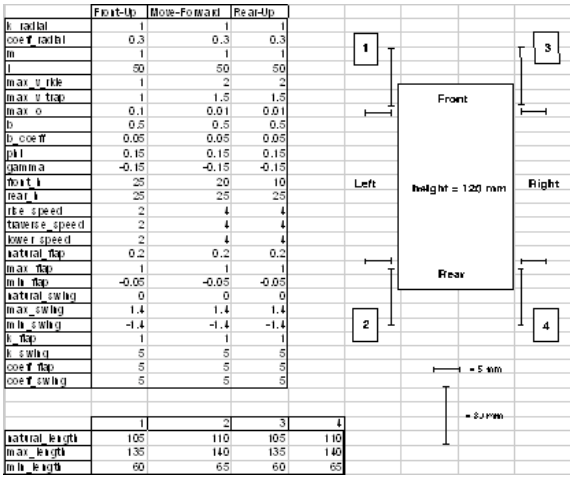
The picture in the upper right of Figure 13 identifies the four legs by the numerical convention employed in the footfall sequence. The lower table in Figure 13 uses this convention. Figure 13 also identifies the initial stance of the robot.

The natural_length, max_length, and min_length given in the lower table refer to the associated radial field values (all in mm). The radial parameter k_radial serves as A and coeff_radial serves as both B and C in Equation 9.

Likewise, k_flap and k_swing are both A in their respective functions, and coeff_flap and coeff_swing are each both B and C. The angles natural_flap, max_flap, etc. are all given in radians.

The parameter b serves as A and b_coeff serves as B in Equation 10.

The parameter max_v_ride refers, in mm/frame, to the maximum speed the torso is allowed to move during while "riding" the potential field (i.e. with all four feet on the ground); likewise, max_v_trap is the maximum speed the torso



	Front-Up	Move-Forward	Rear-Up
k_radial	1	1	1
coeff_radial	0.3	0.3	0.3
m	1	1	1
l	50	50	50
max_y_ride	1	2	2
max_y_flap	1	1.5	1.5
max_o	0.1	0.01	0.01
b	0.5	0.5	0.5
b_coeff	0.05	0.05	0.05
phi	0.15	0.15	0.15
gamma	-0.15	-0.15	-0.15
front_h	25	20	10
rear_h	25	25	25
rise_speed	2	4	4
traverse_speed	2	4	4
lower_speed	2	4	4
natural_flap	0.2	0.2	0.2
max_flap	1	1	1
min_flap	-0.05	-0.05	-0.05
natural_swing	0	0	0
max_swing	1.4	1.4	1.4
min_swing	-1.4	-1.4	-1.4
k_flap	1	1	1
k_swing	5	5	5
coeff_flap	5	5	5
coeff_swing	5	5	5

	1	2	3	4
natural_length	105	110	105	110
max_length	135	140	135	140
min_length	60	65	60	65

Fig. 13. 50 mm Parameter Summary.

is allowed to move while stepping (three feet on the ground). Similarly, max_o is the maximum angular speed the robot is allowed to pitch (in rad/frame).

Phi and gamma are as noted in Figure 11. Rise_speed, traverse_speed and lower_speed are the maximum mm/frame a stepping foot is allowed to move in each of the three stages (noted in Figure 11 and Figure 12).

Certainly, a quick examination of Figure 14 reveals that the three primitives are quite distinct, not only in footfall order but in the characteristics of their steps. Note that the Rear-Up z-displacements for the rear left and rear right legs are much less than 50 mm. This factor is attributable to the previously mentioned forepaw casings. Also, across all three primitives the y displacements of leg 4 (rear right) seem to lag behind the others. It is possible that this asymmetry makes it necessary to bring foot 3 down 10 mm in the Rear-Up primitive.

7.6 Recommendations and Future Work

Extreme Legged Locomotion is an important and very difficult problem. The huge number of dimensions faced by a researcher attempting to solve the problem makes most solution techniques worthless. In the interest of limiting the problem so that part of it could be solved using traditional (gradient based) learning techniques, a field/pre-set geometry system was set up to create an appropriate class of extreme steps. The system was tested on two obstacles (a 35 mm step and a 50 mm step), and was successful at negotiating both.

Problems remain, however, with determining the proper footfall sequence as well as foot displacements during these sequences. Currently, these sequences and displacements are tailored to specific primitives and are associated with specific obstacles. There is no algorithm in place to find a footfall sequence or set of displacements for an arbitrary obstacle field. In addition, the primitive

	Front-Up			
x displacement	0	0	0	0
y displacement	65	70	70	60
z displacement	0	50	50	0
leg index	2	1	3	4
	Move-Forward			
x displacement	0	0	0	0
y displacement	70	70	70	60
z displacement	0	0	0	0
leg index	1	3	2	4
	Rear-Up			
x displacement	0	0	0	0
y displacement	85	75	85	60
z displacement	0	30	-10	30
leg index	1	2	3	4

Fig. 14. 50 mm Footfall Summary.

switching employed here will have to be codified if it is to be learned automatically. Also, refinements to the field system used, including allowances for balance improvements, anticipatory fields, 6-DOF motion, and computational optimization might be necessary. Finally, the matter of extreme dynamic stepping must be addressed. Future studies will be required to solve these crucial problems.

Footfall Order and Step Displacements The largest unaddressed problem in the extreme step approach outlined here is that the footfall order and step displacements were pre-set by the operator. Any system that endeavors to provide locomotion over an arbitrary obstacle field cannot use this technique.

One possible way to solve this problem is to use the fields themselves to plan appropriate sequences of extension and recovery. By integrating the fields to create a scalar potential field (or by creating a new, simplified scalar field) important information can be gleaned about the "comfort" of certain poses. It may be possible to use a state machine to decide when to extend into "uncomfortable" positions and when to relax into "comfortable" ones (a naïve approach of only executing "comfortable" positions would greatly hinder the efficacy of the walk). Future footfalls could be planned in advance, and appropriate "set-up" steps could be sequenced to allow low potential and good balance.

This footfall code was not put in place due to its sheer computational complexity. Given that the step sequences themselves took upwards of 2 minutes to calculate, cycling through hundreds of potential future moves was not feasible. In addition, since all steps used in the study were handmade, the limited amount of terrain available for testing was likely to make any experimental system unreasonably condition-dependent. Finally, and perhaps most importantly, there was not enough good feedback to test out different footfall patterns. The DARPA

set-up will probably solve the last two problems. Significant re-tweaking might solve the first.

Primitive Switching The next significant problem with the previously outlined extreme step approach is the arbitrary creation of primitives, with their own field parameters and footfall sequences. The footfall sequence problem might be solved by the method described in the previous section, but the field parameter switching problem remains.

Enumerating primitives is a possible solution. Such primitives could be identified by the pitch angle and roll angle (see section 8.3) as well as the planned trajectory of the stepping foot. A large, but certainly non-infinite, number of primitives would result, and each could be tuned in turn. One difficulty with this method would be learning the border conditions between primitives.

A promising alternative would be to make the field parameters a function of pitch, roll, and intended step trajectory. A simple linear function might suffice. In this way, the problem of primitive creation and switching might be sidestepped. Also, the parameters that define the functions could themselves be learned with traditional gradient-based techniques. The number of dimensions to be learned, however, would at least double, and most likely triple, under this system.

Field Refinements The first clear area for field refinement is the balance field. The radial and angular foot fields enjoyed a "dead zone", which tended to enhance their performance. The balance field, on the other hand, had no such property. Indeed, the field, with its sole dependence on the distance between the torso centroid and planted-foot-triangle centroid, neglects what often are extreme aspect ratios in the triangle. Given additional time for implementation, the simple fix shown in Figure 15 might be applied. The lines are force directions, leading directly in from each face of the triangle. Within the triangle, there is a much smaller pull towards the center.

The inclusion of anticipatory fields is a second proposed improvement for future studies. Currently, all anticipatory action is hard-coded in: in between footfalls, the robot shifts its weight to the feet that will be planted during the next step. A set of anticipatory fields could accomplish this task automatically, making the entire process more coherent.

The third area for improvement is more obvious: the 4-DOF system should be extended to 6-DOF. This was not implemented in the current project, since it was decided that the symmetric nature of the step made the ability to roll and yaw unnecessary. For arbitrary obstacle fields, however, roll and yaw will require substantial refinement. To this end, a quaternion-based representation of the torso's orientation could be implemented, as described by Mirtich [12].

Finally, the fourth area for improvement is the method for determining field equilibrium. In the present state, the field is evaluated up to 100 times per frame to determine in which direction the torso must move. With proper tuning, it might be possible to eliminate this inefficiency all together, or at least limit the cycles to a more reasonable number. This would allow the robot to execute

its moves in real time, reacting to outside information. Computational overhead limited the current approach to being open-loop.

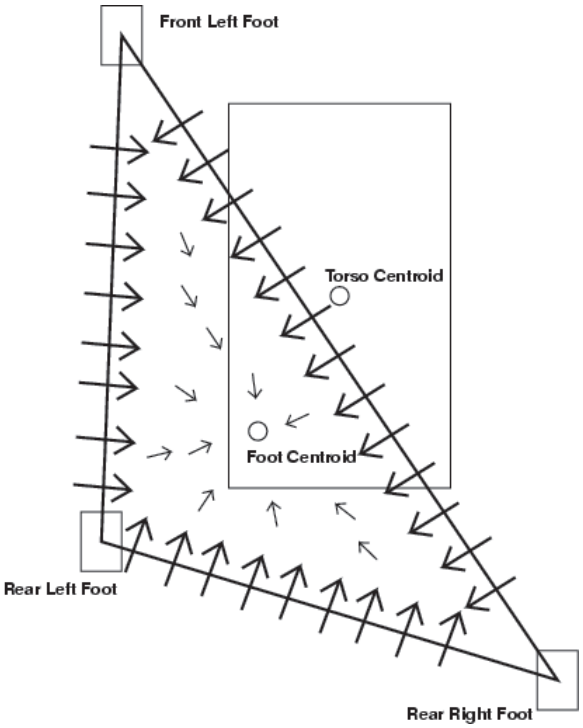


Fig. 15. Refined Balance Field.

Extreme Dynamic Locomotion The final problem to be addressed is most certainly the most interesting: the difficulty of developing dynamic steps for extreme conditions. Dynamic steps would not only provide speedier traversal of obstacles, they are also likely to offer improved performance, allowing the robot to cross obstacles which static steps alone cannot surmount.

The same field principles used in the execution of static steps could be used. But they would require significant and complex modifications. In particular, the balance fields and heretofore non-existent anticipatory fields would have to be extremely well tuned to deal with the unpredictable conditions encountered. The balance field would probably require some strategic imbalance as well. More likely, a completely different system will have to be developed to deal with so much additional complexity.

8 Summary

The University of Pennsylvania 2005 Robocup team resulted in several innovations regarding software architecture and algorithms. In particular, rapid prototyping and development was made possible by incorporating high-level languages such as Matlab and Perl in the software design. These tools made it possible to develop new algorithms for line detection, localization, and team coordination. This development also coincides nicely with the team's research goals of investigating learning algorithms for sensorimotor processing and distributed robotics. By releasing the source code under the GNU public license, it is hoped that in the future, other teams will be able to build upon the successes of this team.

References

1. OPEN-R SDK. `http://www.openr.org`.
2. MATLAB. `http://www.mathworks.com`.
3. Comprehensive Perl Archive Network. `http://www.cpan.org`.
4. Tim Jennes and Simon Cozens. *Extending and Embedding Perl*. Manning Publications, 2002.
5. Manuela Veloso, Scott Lense, Douglas Vail, Maayan Roth, Ashley Stroupe, and Sonia Chernova. CMPack-02: CMU's Legged Robot Soccer Team, 2003. `http://www.openr.org/robocup/code2002SDK/CMU/cmu_teamdesc.pdf`.
6. S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128:99–141, 2000.
7. 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/robocup/2002site`.
8. Ascension Technology Corporation. `http://www.ascension-tech.com`.
9. D. D. Lee. Technical Proposal: Learning Low-Dimensional Controllers for High-Speed Quadruped Locomotion. The University of Pennsylvania, Philadelphia, PA, 2005.
10. J. Bunting et al. Return of the NUBots, Newcastle Robotics Laboratory, The University of Newcastle, Australia, Oct. 2003.
11. J. Chen et al. Rise of the Aibos III – AIBO Revolutions, The University of New South Wales, Nov. 2003.
12. B. V. Mirtich. Impulse-based Dynamic Simulation of Rigid Body Systems. Dissertation for Ph.D. in Computer Science, The University of California at Berkeley, California, 1996.
13. T. Rofer et al. German Team Robocup 2004, Germany, 2004. `http://www.germanteam.org/GT2004.pdf`.