RoboPatriots: George Mason University 2009 RoboCup Team

Keith Sullivan, Christopher Vo,
Brian Hrolenok, and Sean Luke

Department of Computer Science, George Mason University
4400 University Drive MSN 4A5, Fairfax, VA 22030 USA
{ksulliv2, cvo1, bhroleno, sean}@cs.gmu.edu

1 Introduction

The RoboPatriots are a team of three humanoid robots sponsored by George Mason University. Each robot is built on a Kondo KHR-1HV humanoid base and a customized Gumstix Verdex microcontroller. The two attackers share the same behavior, and the goalie's is different. At present there is minimal communication between the robots, but this may change by the competition.

RoboCup 2009 provides an opportunity for the Autonomous Robotics Lab at George Mason University to apply its expertise in evolutionary algorithms to robotics. Our research goals in developing the RoboPatriots have focused on three elements: first, we are interested in the multiagent coordination aspects of the problem. Second, we are simplifying the design of robot motion behaviors by gathering human motion data, then converting this data to the humanoid servos. Third, we are optimizing these motion behaviors in a massively distributed evolutionary computation system attached to a custom humanoid robot simulator of our own devising.

2 Hardware

We are interested in embodied AI rather than hardware development. So, we have chosen commercially available equipment to construct our robot. In particular, we choose the Kondo KHR-1HV as our robot base. Figure 1 shows the hardware architecture and information flow between components. Each robot has 20 degrees of freedom (DOF) controlled via servo motors. There are 6 DOF per leg, 3 DOF per arm, and 2 DOF in the neck. The sixteen KRS-788HV digital servos used in the arms and legs produce 10 kg-cm of torque at a speed of 0.14 sec. / 60 degrees. The two KRS-4024 servos in the shoulders produce 10.5 kg-cm of torque at a speed of 0.17 sec. / 60 degrees. These servos are controlled via the RCB-3 controller board. In addition, two KRG-3 single axis gyros connect to the RCB-3.

The main processing is handled by a 600 MHz Verdex embedded computer attached to a Robostix microcontroller [1]. The Verdex communicates with the

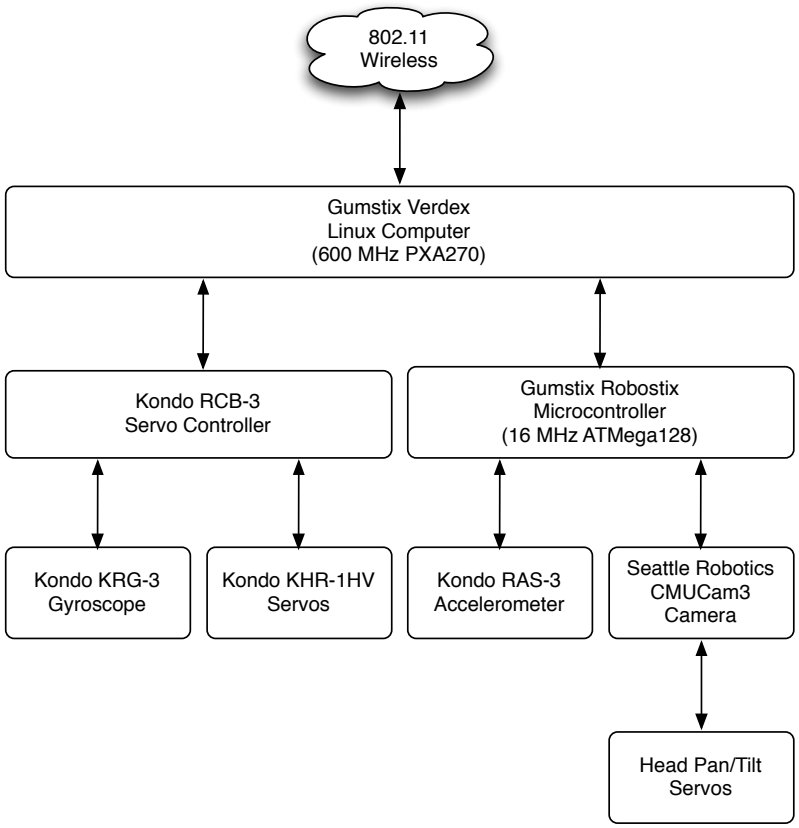

Fig. 1. Hardware architecture and information flow between components

Robostix via I^2C and with the RCB-3 via serial at 115200 bps. A custom inverter board facilitates communication.

We replaced the stock head with a pan-tilt mount constructed from two standard micro-servos and a CMUCam3 [2]. The pan-tilt servos are controlled from the CMUCam3. The CMUCam3 communicates with the Verdex through a serial bridge on the Robostix: data travels via I^2C from the Verdex to the Robostix, and from the Robostix to the CMUCam3 via serial at 115200 bps.

The CMUCam3 is an embedded vision system based on the NXP LPC2106 60 MHz processor and an Omnivision CMOS sensor. The RGB CMOS sensor has CIF resolution (352x288), and operates at 26 frames per second. The CMUCam3 has an open-source API allowing customized software development.

The robots each have a 12 V, 2100 mAh battery. A power regulator board ensures proper voltages to each component. Each robot is equipped with 802.11b wireless.

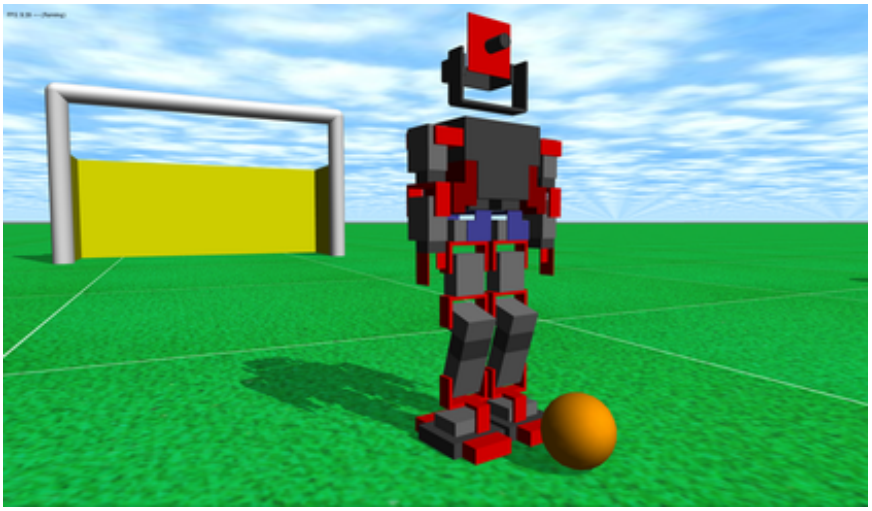

Fig. 2. The ODE simulation of a RoboPatriot

3 Software

Our robot software architecture has four levels, each handled by a different device.

- The RCB-3 Servo Controller handles gyro stabilization and predefined movements.
- The Robostix Microcontroller detects falls and instructs the RCB-3 to stand the robot back up, and instructs the camera to move the head into a safe position.
- The CMUCam3 handles blob tracking and camera servo movement.
- The Verdex performs high-level behaviors, including a simple architecture for localization, moving towards the ball, orientation for kicking, and kicking the ball towards the goal.

4 Human Capture Data

The high-dimensional search space for our evolutionary algorithm largely consists of worthless motions (e.g. unstable motions, self-collisions, etc). To bias exploration towards areas of the search space with desirable, human-like motions, we seed our evolutionary algorithm with retargeted human motion capture data.

We used the Ascension Flock of Birds [3] magnetic motion tracking system to collect motion data from a human subject (one of the authors). The Flock of Birds obtains orientation and rotation data from wireless magnetic sensors

placed on various points of the human body. Due to a limited number of tracking sensors, the data is first collected from 10 sensors attached to anatomical landmarks on the upper extremity of the subject and then using 10 sensors attached to landmarks on the lower extremity of the subject. For the upper extremity we placed one sensor on the summit of each shoulder blade, two on each elbow, and two on each wrist. For the lower body we placed one sensor on each hip, two on each knee, one on each heel, and one near the big toes.

We collected data from fifteen trials of each motion. We then retargeted the human motions to the robot using a simplified kinematic model that does not attempt correct self-collisions or instability. While this results in an imprecise translation between the human capture data and the robot, the evolutionary algorithm learns appropriate motions from the noisy input.

5 Simulation

Unlike evolutionary robotics, we execute our evolutionary algorithm in simulation to increase exploration speed, and to save wear-and-tear on the robot. We created a full 3D simulation using the Open Dynamics Engine simulator [4]. Figure 2 shows our robot and the corresponding simulation. While the simulation is physically plausible, it is not an exact replication of the real robot, and many parameters such as servo play are not modeled. Still, the simulation emulates the real robot: no changes are required to run motions on the robot or in the simulator. We discovered that the evolved motions are easily transferred to the real robot with minor corrections.

6 Evolution of Motion

Developing stable motions for a biped robot is a difficult problem. Typically motions are hand-crafted, requiring significant time and experience. The difficulty is usually related to the numerous degrees of freedom on a humanoid robot, and the careful attention that must be paid to prevent self-collisions and maintain static and dynamic stability. In addition, the hand-crafted motions are most likely suboptimal and specific to a single robot architecture. An alternative approach is use to machine learning to discover stable motions. Machine learning is attractive for several reasons: first, novel solutions may be discovered which are more efficient than human crafted motions. Also, the process can be automated, resulting in faster exploration of potential solutions. Finally, the discovered motions are optimal, or close to optimal.

We examined the use of evolutionary algorithms to discover stable motions. Evolutionary algorithms offer the advantage of parallel search and efficient exploration of the search space. We exploit this parallel search capability by using hundreds of machines to simultaneously explore the space. In addition, evolutionary algorithms can avoid being trapped in local optima.

To improve exploration, we employ opportunistic evolution [5]. Opportunistic evolution operates similar to traditional master-slave evaluation, but tries to

maximize network efficiency by amortizing the cost via increasing the number of evaluations per byte sent. The master bundles the population into chunks, and sends a chunk to each slave. The slave then evaluates the individuals, and, if there is remaining time, performs a mini-evolution loop on its chunk of individuals. In this way, we increase the number of evaluations, and try to exploit any good individuals.

6.1 Representation

We evolve a direct representation of the servo control program rather than an abstract or parameterized representation (e.g., central pattern generators). Our servo controller does not permit a more complex representation, and the direct representation allows the evolution of non-cyclic motions such as kicking. However, the number of possible direct representations is huge, and many of the possible servo configurations or transitions between configurations are impractical. To help our evolutionary algorithm, we seed the algorithm with human capture data.

Each individual is represented by a script. Each script contains $N > 0$ moves, where each move consists of a speed s and 24 servo positions, of which, 18 are evolved and six are held constant. We do not evolve control for the neck servos.

Our representation necessitated custom genetic operators. The mutation operator changes the individual in several ways. First, with probability p_{add} a random move is added at a random position to the script. Then, with probability p_{delete} a random move is deleted, ensuring that the script contains at least 3 moves. Finally, we loop through all the moves and add uniform random noise to the speed with probability p_{speed} and add uniform random noise to each servo parameter with probability p_{servo}.

Our crossover operator first performs a variant of traditional two-point crossover. To crossover two individuals, we select two random integers a and b, both less than the length of the shortest individual. The moves between a and b are then swapped. Next, with probability p_{arms} we swap just the arm servo commands across all moves between the two individuals. Similarly, we swap the legs with probability p_{legs}.

6.2 Parsimony Pressure

One issue with our representation is unconstrained growth of the individuals without significant performance improvements. Typically, this growth, called *bloat*, is controlled by special genetic operators that limit the maximum size of an individual. However, parsimony techniques, i.e., selection methods which consider the size of the individual, have recently become popular and have shown to be effective tools for controlling bloat. The simplest parsimony pressure technique is lexicographic tournament selection which works the same way as normal tournament selection, but if multiple individuals have the same fitness, then the one with the shortest length is chosen [6].

6.3 Experimental Results

Using our own ECJ evolutionary library [7], we performed experiments to evolve walking, kicking, turning and side step. Each motion required a different fitness function, due to the different goals of each motion. For all the motions, stability is the primary goal. For walking and side step, we want to maximize how far the robot moves in a straight line. In kicking, we want to maximize how far the ball travels in a straight line, and for turning we want the final orientation to be as accurate as possible. Due to these multiple objectives, we used the Strength Pareto Evolutionary Algorithm [8] to evolve a Pareto front of solutions; of the possible solutions, we then manual chose the most appropriate one.

References

1. (No Author): Gumstix, Inc. `www.gumstix.com` (2007)
2. Rowe, A., Goode, A.G., Goel, D., Nourbakhsh, I.: CMUCam3: an open programmable embedded vision sensor. See also `www.cmucam.org`. CMU-RI-TR-07-13, Robotics Institute, Carnegie Mellon University (2007)
3. (No Author): Ascension technology corporation. `http://www.ascension-tech.com` (2008)
4. Smith, R.: Open dynamics engine. `http://www.ode.org` (2009)
5. Sullivan, K., Luke, S., Larock, C., Cier, S., Armentrout, S.: Opportunistic evolution: Efficient evolutionary computation on large-scale computational grids. In: Proceedings of Genetic and Evolutionary Computation Conference Late Breaking Papers. (2008)
6. Luke, S., Panait, L.: Lexicographic parsimony pressure. In Langdon, W.B., ed.: Proceedings of the Genetic and Evolutionary Computation Conference. (2002) 829 – 836
7. Luke, S.: ECJ 18: A Java-based evolutionary computation research system. `http://cs.gmu.edu/~eclab/projects/ecj/` (2008)
8. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: improving the strength pareto evolutionary algorithm. Technical report, Swiss Federal Institute of Technology (2001)