Extensions of a RoboCup Soccer Software Framework

Stephen G. McGill, Seung-Joon Yi, Yida Zhang, and Daniel D. Lee

University of Pennsylvania
{smcgill3,yiseung,yida,ddlee}@seas.upenn.edu

Abstract. The RoboCup soccer leagues have greatly benefitted Team DARwIn and the UPennalizers. The stiff competition has hardened our code into a robust framework, and the community has allowed it to flourish as an open source project used by many teams. Working with the open source DARwIn-OP hardware allows even more clairvoyance into the inner workings on the low level code the builds to state machines. We show how our codebase performs in the Webots simulation and on the Open Source DARwIn-OP platform. From these beginnings, we wish to apply our codebase to new scenarios for humanoids including human robot interaction and manipulation tasks. Many of these scenarios are explored by other RoboCup leagues including @Home and Rescue, where we see a new avenue for our codebase. New human robot interaction features are described in our framework, and example performances are demonstrated. Finally, we describe added standards compliance and open source tool usage that will give our codebase more accessibility.

1 Introduction

As a brief background, Team DARwIn and the UPennalizers are teams from the Humanoid KidSize league and the Standard Platform League, respectively, for RoboCup soccer. These teams work on the same codebase for playing software, and have released open source versions of this code since 2011. The current releases can be obtained online[1]. Because these teams operate on two totally different humanoid robots in different leagues, the focus with each software release has always been on portability and compatibility with a variety of humanoid platforms.

As such, this code has been tested, and utilized in competition, on the ubiquitous Nao and DARwIn-OP platforms, and, additionally, on custom DARwIn-XOS and CHARLI platforms. This code has performed well, pushing the DARwIn and CHARLI teams to victory in each of the past two years, and running the UPennalizers standard platform team in the same period. Furthermore, many teams in the humanoid kid-size league have used our code in competition; we have received bug reports from these teams to improve its quality.

This software has been used in-house on humanoid robot experiments outside the realm of soccer, including teleoperative control and recently, the DARPA

[1] http://seas.upenn.edu/~robocup

S. Behnke et al. (Eds.): RoboCup 2013, LNAI 8371, pp. 608–615, 2014.

robotics challenge[4]. With these new directions, it became clear that we needed to extend the platform in a few crucial directions, many of which will directly impact the usability in RoboCup.

In summary, many teams have begun to examine or use our codebase as a starting point for their own RoboCup entries, and we have found it useful in working with external colleagues and visiting scholars on humanoid research. While RoboCup soccer does not focus on these extensions, the RoboCup@Home, RoboCupRescue, and the new RoboCup@Work leagues may find it applicable. With applications suitable for a wide audience, we document our recent developments for an open source humanoid robot framework.

2 Related Work

Open source robotic frameworks are not new; a good survey of original formulations of them can be found here [8]. There are many examples, from ROS to OROCOS, from the Microsoft Robotics suite to the B-Human RoboCup soccer release. Each framework, however, has its own focus based on its history. The ROS operating system [12] tries to be all things to all people, and has broad support in the robotics community. However, its history with humanoid robots in particular has not been in as much focus. With our experience stemming from RoboCup and its humanoids, we wish to prioritize humanoid control, sensing, and planning.

OROCOS, on the other side of the coin, focuses on its niche application of realtime control [2], and does this well. However, it cannot support a variety of device drivers that are required for building complex robots. Thus, OROCOS is not sufficient for running a soccer playing robot. Contrasting the niche focus, the Microsoft Robotics studio [6] provides a high level design suite for writing robotics programs. However, this system is restricted to use on the Windows operating system. We wish our framework to be supported on multiple platforms, with a focus on UNIX based systems.

Specific to RoboCup, the B-Human code release is tailored to the Nao robot, which many teams in the Standard Platform league choose to use. While the code has been proven very effective at winning championships, it will not port to other humanoids. Most recently, accompanying the release of the NimBro-OP robot was open source software. While this release is promising, it has just begun and is not tested on other platforms. Other frameworks originating from RoboCup competitions include the Fawkes system[10] which uses the Lua language like ours. However, there is no included walk engine - crucial for teams in RoboCup soccer and for humanoid research.

Given this representative sample of robot frameworks, what we try to accomplish with our framework is a focus on humanoids and a low dependency, small footprint, code base. Most importantly for RoboCup, we wish to keep compliant with the latest rules and regulations for all four humanoid robot devisions - Standard Platform league and each of the three humanoid leagues.

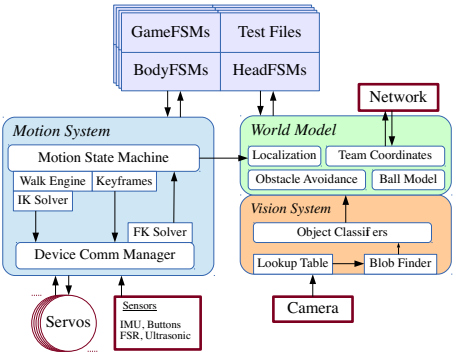

Fig. 1. The system architecture is divided into low level drivers (boxed in red), task specific motion and perception modules, and high level finite state machines (lavender boxes), communicating over shared memory and a message passing system.

3 Soccer Playing Framework

Our soccer framework has focused on providing a general purpose walk controller, a simple vision system, and a set of extensible state machines. The code is written in a combination of the Lua scripting language and the widely used C and C++ languages.

Shown in Figure 1 is a high level overview of how data flows in our system. There are three processes that execute the vision system the device communications manager (DCM) and the motion system state machines. Data is shared using a shared memory segment on the system, where the world model, vision system, motion system, and DCM have their own memory segments. These segments can be read by any other process. For instance, the localization system requires odometry information from the motion system, which is shared using this memory segment.

The benefit of this structure is that the DCM and Camera modules are able to abstract away the platform dependent drivers. We have a different DCM for the Nao, OP, simulated hardware, and any other robot we operate. For instance, the DARwIn-OP and Nao have Linux V4L2 cameras, while Webots has a memory segment. Each driver, however, interfaces the shared memory in the same way.

3.1 Locomotion

We provide a well-documented implementation of the popular Zero Moment Point (ZMP) walking engine. This implementation uses the analytic solution of the ZMP equation, assuming that the ZMP is piecewise linear. The main advantage of this approach is that it is very simple, and it does not require any preview interval. After the foot and center of mass (COM) trajectories are calculated, the inverse kinematics solver generates joint angle values for the leg actuators. It is important for open source software to have a simple and understandable

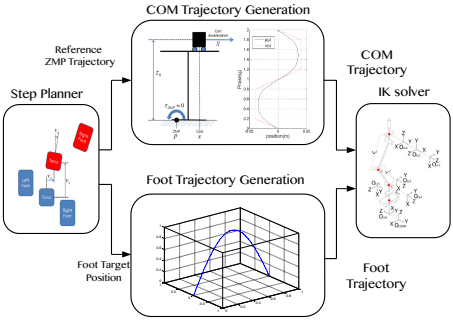

Fig. 2. The motion system provides a abstractions from desired footsteps through ZMP trajectories to inverse kinematics

base system. Shown in Figure 2 is a detailed look at how commanded walking velocities are transformed into joint commands.

3.2 Vision

In our provided vision system, we use a look up table to first categorize raw camera pixels into one of 8 color labels. The goal of the colortable generating utility is to provide a supervised learning approach to classify raw camera pixels into a set of 8 color class labels - possibly green for the field, orange for the ball, etc. Typically, we receive a YCbCr color formatted pixel array from the camera, and convert this to RGB for displaying to the user. Figure 3 shows the colortable generating tool for a real robot camera frame, and how the generated lookup table performs in simulation.

The user clicks on pixels that belong to each of these labels to generate both positive and negative examples of the color. We then apply a gaussian mixture model to generate color segments. For each mixture, we perform a threshold to find pixels with a high probability of belonging to that mixture?s color class. With a high enough score, we add that pixel to a lookup table mapping to the color class. This lookup table from the mixture model to color class saves on per pixel computation, since the probability does not need to be computed for each pixel on every incoming frame; downsampling further increases speed. This labeled image is fed into high level blob detection routines and object classifiers.

Given a labeled image it is downsampled, where labeled pixel are grouped into 4 pixel by 4 pixel bitwise OR-ed blocks for faster execution in later image analysis. This downsampled image is fed into high level blob detection routines and object classifiers. Results are show in Figure 3.

3.3 World Model

Our World model collects information about odometry from the motion system, teammate positions from network, and observed objects from the vision system.

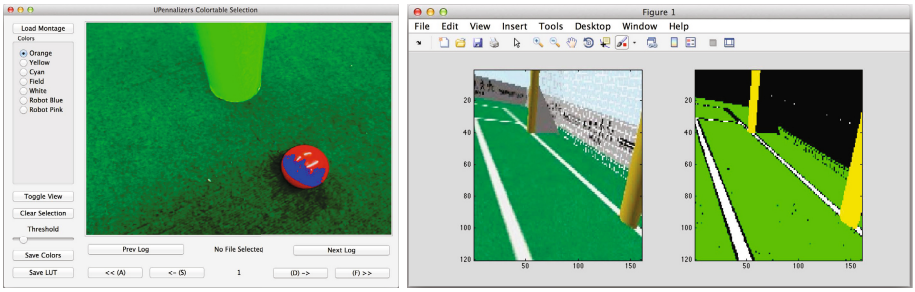

Fig. 3. Colortables can be made with a QT user interface for classifying ball colors (left). The generated lookup table can be monitored in MATLAB (right).

With this information the world model can assign roles for the robot, determine the robot's pose, and identify obstacles. Localization is performed using acoustic triangulation or particle filters based on landmark observations. Our obstacle avoidance methods is described in [13].

3.4 Interprocess Communication

While we use the Boost shared memory system to synchronize data across processes, we needed to add event based information sharing for more responsive processing loops. To ensure compatibility, we choose to implement a widely used standard that supports a broad array of features. Our framework has adopted the ZeroMQ system [5], which focuses on being lightweight with low latency. An additional benefit of the ZeroMQ system is that it is an open source, standard, definition with bindings in many languages.

Since we use Lua objects to represent information like poses and ball position, we must serialize these objects before sending over a ZeroMQ channel. We have two serialization libraries - our own custom library and a MessagePack library. With MessagePack, Lua objects are reconstituted easily in other languages, since tables, strings, and numbers, for instance, are well defined.

3.5 Debugging Tools

Figure 3 shows our monitoring program that receives frames from the Webots simulator or the real robot, and displays the raw camera frame and the labeled image side by side. Recently, we have added open source tensor manipulation libraries using the Torch7 project [3], which runs under Lua. We have extended the available Torch system as a packaged module. For graphical interfaces, we provide QT windows for displaying information. These moves allow team members, and other institutions, who do not have access to MATLAB to be able to use our system and contribute to it readily. An additional benefit of having two methods for visualization is that it pushes the codebase to be more modular. If a better interface becomes available in the future, we can port more easily our code.

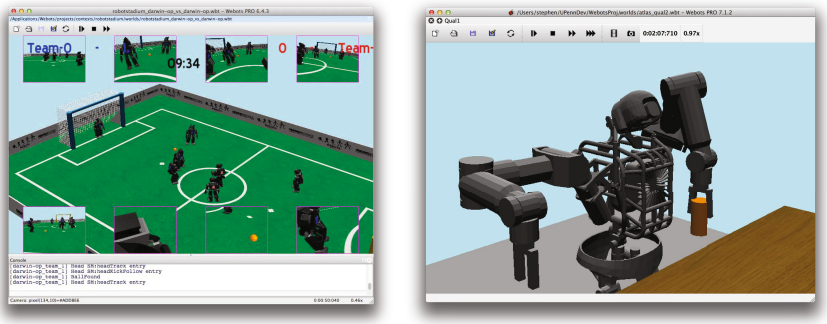

Fig. 4. Left: Teamplay is simulated using the Robotstadium competition, where we can rapidly prototype behaviors which perform similarly on real hardware. Right: The open source ATLAS model used in the Webots simulator to execute our teleoperated manipulation routine.

4 Evaluation

We utilize the freely available Webots [9] Robotstadium platform to evaluate our teamplay code. Using a simulator allows for rapid prototyping, with reasonable speeds, even for full physics simulation. Shown in Figure 4 is an example of two teams of four robots running our code. Each team (or even each player) can be slightly modified to run a different parameter set. We test all of our software on Mac and Linux operating systems, and provide initial support for the Gazebo simulation platform [7].

Most importantly, we evaluate our platform on the DARwIn-OP, the Aldebaran Nao, and custom humanoid hardware. Our software platform is extensively tested in competition each year during RoboCup.

We evaluate our platform on the DARwIn-OP, the Aldebaran Nao, and custom humanoid hardware. Our software platform is extensively tested in competition each year during RoboCup.

5 Beyond RoboCup

Outside of robot soccer, humanoids perform manipulation tasks and interact with humans. While manipulation tasks are gaining traction in soccer competitions, they are not a focus; human interaction is limited to off field training. Thus, these aspects have not become a focus for many teams.

However, outside RoboCup soccer, these tasks are immeasurably important. In RoboCup@Home, for instance, robots must manipulate objects on tables, as well as interpret commands from a human. In RoboCupRescue puts an emphasis on exploration using sensors not allowed in the soccer competitions. However, the recent DARPA Robotics competition has actually merged RoboCup soccer

and the Rescue league by encouraging humanoid robots to be used in disaster relief scenarios. Many people involved in our RoboCup effort are working on this DARPA project, and so our software must scale to achieve this broader impact. For instance, shown in Figure 4 is our Webots version of the open source ATLAS model picking up an object.

Our first foray into manipulation and human interface requires a set of device drivers to be present in our a code base. To control our robots by teleoperation, we have focused on providing drivers for some human interface devices. While UVC cameras have well defined drivers in out framework, we needed to add more devices We now interface a Spacemouse, originally developed for robots [1], for 6D control of the manipulator, a Kinect for skeleton tracking using the NiTE [11] libraries, and recently a LEAP sensor for gripper control.

5.1 Challenges and Opportunities

While providing support for a multitude of humanoids and devices enhances the usability of the codebase, it presents challenges for maintaining concise abstractions. For instance, allowing more operation modes that the DARPA challenge and other RoboCup leagues require that more motor interfaces must be specified, and that the higher level state machines need to amend certain API calls. By using using Lua in place of C/C++ for dynamically adding function calls, extra operating modes become less intrusive.

Additionally, sensors must be able to interpret a wider scope of environments than that of the well defined soccer field. While our framework does not solve these problem, it can provide a way to approach it. This particular aspect is where many new state machines, sensor drivers, and other aspects of the framework are evolving. We hope to release these new features as they become more mature.

Overall, we are are attempting to target humanoid researchers who have a background in RoboCup, or an understanding of the RoboCup competition. Because RoboCup provides a concrete evaluation of software, the particular state machines and abstractions become easier to understand precisely their is some intuition of the nature of soccer. In the near term, we hope that our general humanoid implementation can be applied to leagues outside of soccer.

References

1. Space mouse, the natural man-machine interface for 3d-cad comes from space. In: ECUA 1996 Conference for CATIA and CADAM users in Europe (1996)
2. Bruyninckx, H., Soetens, P., Koninckx, B.: The real-time motion control core of the Orocos project. In: IEEE International Conference on Robotics and Automation, pp. 2766–2771 (2003)
3. Collobert, R., Kavukcuoglu, K., Farabet, C.: Torch7: A matlab-like environment for machine learning. In: BigLearn, NIPS Workshop (2011)
4. Defense Advanced Research Projects Agency DARPA. DARPA robotics challenge (April 2012)

5. Hintjens, P.: ZeroMQ: The Guide (2010)
6. Jackson, J.: Microsoft robotics studio: A technical introduction. IEEE Robotics and Automation Magazine 14, 82–87 (2007)
7. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2004, vol. 3, pp. 2149–2154 (2004)
8. Kramer, J., Scheutz, M.: Development environments for autonomous mobile robots: A survey. Auton. Robots 22(2), 101–132 (2007)
9. Michel, O.: Webots: Professional mobile robot simulation. Journal of Advanced Robotics Systems 1(1), 39–42 (2004)
10. Niemüller, T., Ferrein, A., Lakemeyer, G.: A Lua-based Behavior Engine for Controlling the Humanoid Robot Nao. In: Baltes, J., Lagoudakis, M.G., Naruse, T., Ghidary, S.S. (eds.) RoboCup 2009. LNCS (LNAI), vol. 5949, pp. 240–251. Springer, Heidelberg (2010)
11. PrimeSense Inc. Prime Sensor NITE 2 Algorithms notes, 2012 (last viewed March 21, 2013)
12. Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: An open-source robot operating system. In: ICRA Workshop on Open Source Software (2009)
13. Vadakedathu, L., Sreekumar, A., Yi, S.-J., McGill, S., Zhang, Y., Lee, D.: Comparison of obstacle avoidance behaviors for a humanoid robot in real and simulated environments. In: The 7th Workshop on Humanoid Soccer Robots (2012)