Unified Humanoid Robotics Software Platform

McGill, Stephen G. [#1], Brindza, Jordan [#3], Yi, Seung-Joon [#2], Lee, Daniel D. [#4]

[#] *Electrical and Systems Engineering Department, University of Pennsylvania*
200 South 33rd Street, Philadelphia, PA 19104, U.S.A.
[1] smcgill3@seas.upenn.edu
[2] brindza@seas.upenn.edu
[3] yiseung@seas.upenn.edu
[4] ddlee@seas.upenn.edu

***Abstract*— Development of humanoid robots is increasing rapidly, and humanoids are occupying a larger percentage of robotics research. They are seen in competitions like RoboCup, as consumer toys like the RoboSapien from WowWee, and in various research and DIY community projects. Equipping these humanoid hardware platforms with simple software systems that provide basic functionality is both cumbersome and time consuming. In this paper, we present a software framework for simple humanoid behavior that abstracts away the particulars of specific humanoid hardware.**

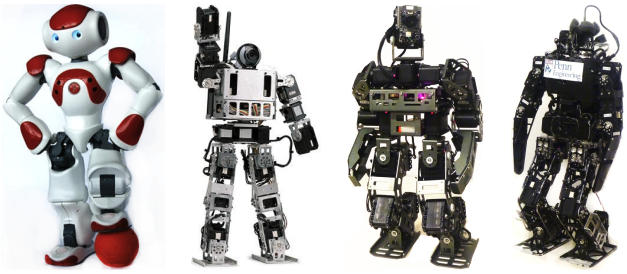

Fig. 1. Various supported humanoid robotic platforms, from left: Aldabaran Nao, MiniHUBO, DARwIn LC, DARwIn HP.

I. Introduction

From designing and gathering components to building and tuning algorithms, humanoid soccer design can be a cumbersome process. In our effort to reduce overhead in applying algorithms and testing across several humanoid platforms, we have developed a modularized software platform to interface the many components that are common to humanoids.

Our modularized platform separates low level components that vary from robot to robot from the high level logic that does not vary across robots. The low level components include processes to communicate with motors and sensors on the robot, including the camera. The high level components include the state machines that control how the humanoids move around and process sensor data. By separating into these levels, we achieve a more adaptable system that is easily ported to different humanoids.

This paper explains how each subsystem works, how they are tied together, and how new devices can be incorporated. Our inspiration for this system is the RoboCup soccer league, where we have competed with both the Aldabaran Nao humanoids and the DARwIn HP humanoid [1]. In dealing with these two humanoids, we have developed a shared code base that we wish to port to more robots.

We are targeting both the Nao and DARwIn HP platforms for this framework. Also, we are targeting the DARwIn LC, a low cost version of the DARwIn HP, and the MiniHUBO platform, a miniaturized version of the HUBO robot. Hooks to provide integration with the Webots simulator are provided, since testing changes on physical humanoids can often times result in damage hardware. The various physical platforms are shown in Figure 1.

II. Prior Work

In the Robocup soccer competition, some teams do release their code in an open source fashion [2]. However, these software releases are not amenable to porting from robot to robot. In the Standard Platform League, all teams share the same robotic platform, but their code cannot be used on other robotic platforms that vary from team to team in the other Leagues of RoboCup. Outside the Standard Platform League, teams configure custom hardware for their own robotic platforms, and their software, consequently, is not able to run on other robot outside of that team. Thus, while these RoboCup releases are open source, they are not portable and cannot serve as a a generic humanoid basis.

Aldabaran ships its Nao robots preinstalled with the Urbi software distribution. This software has been released under an open source license this summer. Urbi is a popular framework that is used across a wide variety of robots, and includes support for multiple humanoids. [3]

The well known ROS has long provided an open source approach to robotics. However, there are only two applications of ROS towards humanoid robotics, of which, only the Nao platform supports legged locomotion. The Nao specific ROS "provides joystick teleoperation, odometry, joint state, and a basic robot model for Nao."[4] With no locomotion support, its primarily use case is for teleoperation.

There are many other open source robotics stacks, but these do not apply specifically to the area of humanoid robotics, or are tied to a specific hardware set. The European Commission is funding an open source humanoid framework, where both the hardware and software is available under an open source

Proceedings of the 5th Workshop on Humanoid Soccer Robots @ Humanoids 2010,
Nashville (USA), 2010, December 7
ISBN 978-88-95872-09-4 pp. 7-11

license. However, this project is tied to a specific humanoid hardware configuration [5]. OpenRDK is an open source effort to provide "robotic applications rapid-development." It works with the Nao humanoid, but it is not mentioned whether it works on any other humanoid, or how to port their code to new robots [6]. Finally, the ORCA project has been used in many mobile robotics applications, but so far it has not been targeted at humanoids [7].

Our platform addresses some of the shortfalls of the previously mentioned platforms by being a small and efficient framework. All told, there are just over 15,000 lines of code to support 4 platforms, including a simulator. The platform is separable, such that other open source frameworks, like OpenCV, can be utilized in image processing, thus enhancing the provided and simple image processing code. Specific to Humanoid platforms, a Zero Moment Point (ZMP) solver is included that provides robust omnidirectional gaits. This ZMP solver is applicable across any humanoid platform, and, of the above mentioned platforms, none includes a ZMP solver.

III. Architecture Overview

To establish a flexible cross-robot platform, we define which components of the platform are the same across robots, and which components vary. The goal of this architecture is to have the same logic – the same behavior for interacting with the world – regardless of which humanoid hardware the platform interacts with. That is, we would like two different humanoid configurations to play soccer with the same set of decision rules, while having different sets of motor and sensor configurations (that still form a humanoid). The components that may differ across platforms are kinematic parameters, motor communication buses, cameras, and sensor sampling buses.

There are a certain number of "System Requirements" in order for the Humanoid Robotics Platform (HRP) to run effectively. At minimum, we require pose feedback from an IMU sensor and joint position readings from the motor. We require a humanoid configuration of motors, where the motors can accept angular position commands. A camera system is required, but can be disabled if not needed or wanted.

A. Technical Implementation

As discussed, the Humanoid Robotics Platform (HRP) platform runs on the Linux operating system. HRP is written in a combination of Lua and C/C++. The Lua portions operate the high level State Machine, Vision and Locomotion processes. The C/C++ routines implement the low level processing, handling interactions to motors and sensor systems along with the kinematics engine. We choose the Lua and C combination because we wanted a scripting library that could easily interact with C/C++ device driver routines [8].

IV. Motion Subsystem

We first tackle the issue of separating the platform specific, low level components from general, high level design goals for robot movement control. Low level components include

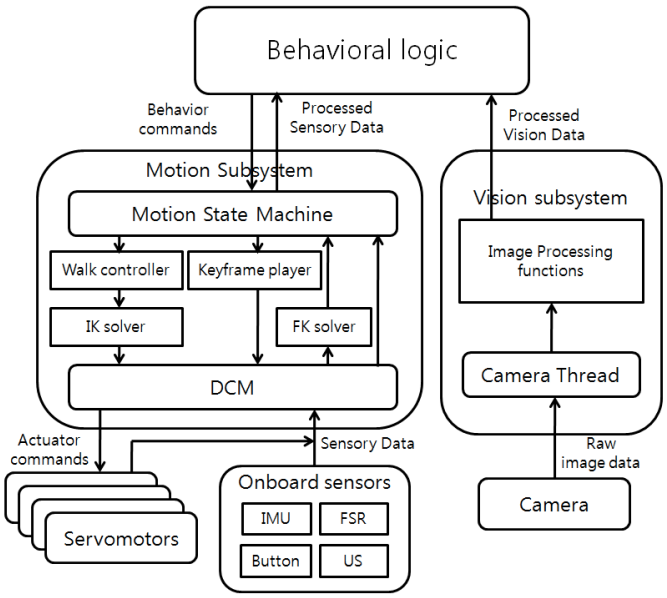

Fig. 2. Block Diagram of the Software Architecture.

the communication with actuators and sensors, such as commanding joint angles and receiving joint encoder measures. As each humanoids may have different hardware configuration, platform specific forward and inverse kinematics solvers are also delegated to this realm.

High level components include a zero moment point (ZMP) based bipedal walk controller and keyframe motion player. Although they are platform independent, the walk controller and keyframe motions can take platform specific parameters or keyframe data. The behavioral logic module commands the Motion subsystem through a Motion State machine that executes 'stand,' 'walk,' and other human like motions. We will discuss details of them in following subsections.

A. Data Communications Manager

The low level communication with the actuators and sensors is split into a completely separate process from the high level behavioral logic. This separation ensures that the low level communication is continuously operated at the highest rate possible, regardless of the rest of the system. Without such separation, a high processing load at other part of the system may disrupt the low level communication of the robot, possibly resulting in jerky or unstable movement of the robot.

This communication process is dubbed the Data Communications Manager (DCM). The DCM constantly writes all servomotors' desired angles and reads the servomotors' current joint encoder measurements; it also polls onboard sensors such as inertial sensors or buttons.

Since it handles low level communication to a specific set of motors and sensors, the DCM process is platform specific. In our implementation on the Aldabaran Nao, the DCM interacts with the NaoQi module because the NaoQi is central interface for all actuators and sensors [9]. For a more general DARwIn platform without such a module, the DCM uses an interface

Proceedings of the 5th Workshop on Humanoid Soccer Robots @ Humanoids 2010,
Nashville (USA), 2010, December 7
ISBN 978-88-95872-09-4 pp. 7-11

module written in C to directly communicate with Dynamixel servomotors and onboard sensors. We communicate using the Dynamixel protocol [10]. When using the Webots controller, we connect the Webots NaoQi controller [11].

To share motor and sensor data with the Behavioral logic and the rest of the Motion subsystem, we use the Linux shared memory framework. The DCM maintains two shared memory blocks: one for motor data, and one for sensory data. The motor block includes motor command variables such as target position, electrical compliance, maximum velocity, and other related variables. The sensory block shares readings such as joint encoder values, as well as accelerometer and gyroscope measurements. By using shared memory, we allow other processed, like MATLAB, to be able to provide debugging tools.

B. Kinematics and Keyframes

Our locomotion code uses forward and inverse kinematics solvers that platform specific to account for the differences in humanoid limb configurations. Due to the possible hardware constraint such as the hip joint of the Nao robot, we provide the opportunity to use platform specific kinematics solvers, as declared in the global configuration file. However, in most case, tuning the existing kinematics solvers with different parameter sets will work.

Using the platform specific kinematics solvers, we provide a system to retrieve motor mapping from anatomical descriptions of the humanoid to joint ids. This allows functions to be named aptly as "get_head_position," etc.

In addition to commanding limb end effectors to certain positions, we provide methods to playback prerecorded keyframe motions. The keyframe motion is typically used for kicking and standing up behaviors. Keyframe data is platform specific as well, which is automatically selected for each platform by the specification in the configuration file.

C. Locomotion

In our HRP, the zero moment point (ZMP) [12] based omnidirectional dynamic walking controller governs humanoid locomotion. A ZMP trajectory is generated in real time according to the commanded walk velocity from the Behavioral logic. The center of mass (COM) trajectory is also calculated in real time to meet the ZMP criteria.

Instead of a more general, optimization-based approach such as ZMP preview method [13], [14], we use 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 COM trajectory is calculated, the inverse kinematics solver generates joint angle values for the leg actuators.

Our walking controller also includes stabilization control using sensory feedback. Bipedal walking is very susceptible to external disturbance or surface irregularities, and humanoid robots can operate much better in real world with the help of explicit feedback stabilization.

We use two types of sensory feedback: proprioceptory and inertial. For proprioception, we use joint encoder angles from leg actuators and the relative angle between support leg frame and torso frame, calculated using the forward kinematics solver. For inertial feedback, we use the current angular velocity of the torso, measured using the torso-mounted gyro sensor. These feedback sources provide negative feedback on the desired torso angle in order to stabilize the robot under disturbances.

Our walk controller can be used for general humanoid robots and has a number of configurable parameters that influence the performance of the walking. While we believe the default walking parameters that we provide will work as a good starting point, we expect users to tune them when it comes to adapting the HRP to their humanoid.

D. Motion State Machine

The motion state machine controls the overall behavior of the motion subsystem. It provides abstract commands that the behavior logic module employs, such as 'standup', 'walk', and 'kick'. The motion state machine dispatches these high level motion commands to start the appropriate behavior. Also, by checking the IMU sensory data, the Motion state machine can detect if the robot is falling. It then reacts accordingly to minimize the falling damage, and initiates a stand up motion according to the fallen posture of the robot.

V. Vision Subsystem

Similar to the Motion subsystem, we separate the data collection process from the behavioral logic. We have written drivers for the UVC class of video camera on Linux, and separated frame grabbing into a distinct thread of execution. Unlike the motion, we expect more customizations to be applied on the image processing end of the Vision system.

We have made a driver for UVC cameras, with the Video4Linux2 API [15]. If a robot's camera is not UVC compatible, a custom camera driver can be used by providing a Lua interface that can provide images, height, width, and other parameters described in the HRP API. This capturing process runs in a thread that continually samples images from the camera, and exposes the image in memory to the Lua interface. Camera parameters, such as white balance and exposure time, can be configured in the Config file.

In addition to the UVC camera driver, there is a "dummy" camera driver that is provided. This driver yields random data for each call to grab an image from the camera. This is important for debugging, and for adding new humanoid platforms. By using the dummy driver, the DCM can be developed unimpeded by the lack of a camera.

Images can be grabbed at any time from the camera, but we have consolidated all image processing into a single Vision process which contains the entire pipeline of image processing. This Vision process, like the other middle level functions, runs as an update process in the main loop. On each update, the pipeline is run again.

Proceedings of the 5th Workshop on Humanoid Soccer Robots @ Humanoids 2010,
Nashville (USA), 2010, December 7
ISBN 978-88-95872-09-4 pp. 7-11

A. Processing

In our provided image processing, we use a look up table to partition objects in the environment, based on color. To this end, the look up table matches a perceived color with its associated object - the redder the pixel, the more likely it is a ball. Before areas are fully labeled, we block the image into 4 pixel by 4 pixel bitwise OR-ed blocks for increasing processing speed. A potential ball is further pruned based on characteristics like size and shape [16]. Currently, only ball detection is included in the release, as the software is not built for a specific task in mind. Results of this pipeline are shown in figure 3.

We determined to be vision platform agnostic, so as to provide a small and nimble library. If users would like to add support for libraries like OpenCV, it is easily workable by adding new function calls in the Vision update routine.

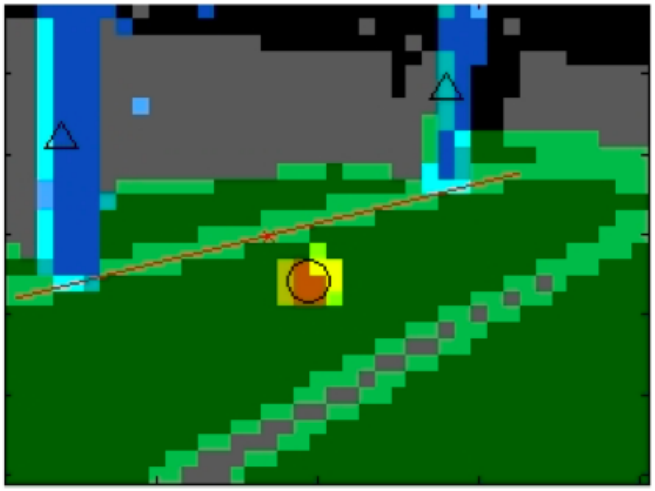

Fig. 3. Results of Image Processing, where the ball is circled [16].

B. Customizing Image Processing

We provide a set of standard image processing functions that perform a range of tasks such as image segmentation and lowering the resolution. All Vision code is written in C/C++, and is custom built, using no outside vision libraries and are available to all robots.

A user can effectively enable or disable certain image processing functions based on their available computation power or other needs. In order to apply their own image methods, they need only to provide Lua wrappers to their code and call their methods from with the Vision processing segment of the update routine. In this way, other image processing libraries can be used in place or, or in addition to, the provided set of functions.

VI. Behavioral Logic

The behavioral logic is a Lua codebase that is shared across disparate humanoids and controls the high level behavior of the robot. This behavior includes tracking a ball with the camera, walking towards that ball, kicking the ball when the ball is in range, etc. It initializes the DCM process first so that the robots' sensors and actuators are available for communication, and then it starts the main loop which regularly updates the behavior state machines. Each behavior state machine then communicates with the low level subsystems to get access to hardware functions such as head movement or walking.

A. Behavior State Machine

As we have mentioned above, the high level control of the robot is done by using a number of state machines in the behavior logic. In the HRP we provide two state machines, the head state machine and the body state machine.

The head state machine controls the movement of the head, so that it can look around until it finds the ball and track the ball when the ball is within sight. When the ball is not visible, it starts looking around again. The body state machine controls the movement of the body by controlling the walk direction and velocity or initiating a keyframe motion. It makes the robot wander around when the ball is not visible, and move to the ball and kick it if the robot sees a ball.

Users can easily modify those state machines or add new ones to meet their ends, and we think the provided behavior state machines will be a good starting point.

VII. Adding a New Humanoid

The crucial feature of this software release is being "platform agnostic." We have described how driver software is separated from high level functionality. To add new humanoid platform that can run our soccer playing behavioral logic, one needs only to implement the Lua interfaces for their camera, sensor, and motor systems. From there, it is time to tune the kinematics, camera, and walk parameters. These parameters are stored in a single configuration file, and therefore, easy to navigate.

A. Adding a DCM

The first step in adding a new robot is to add the DCM process that is run just before the state machine is executed. As described, this code runs as a standalone process that reads and writes motor and sensor data in shared memory files. If using a Dynamixel based robot, this step can be effectively skipped, by copying the code available for the DARwIn platform.

The first step in the DCM is to make a C routine that allows your Linux computer to move and measure motors and read sensor values. Next, the goal is to be able to interact your code with Linux's shared memory system. We provide a Lua utility for reading and writing shared memory, but it is up to the user to comply with HRP's shared memory data structures.

By complying with the data structure of the shared memory, you can test that your DCM operates correctly by writing command values to the shared memory from a Lua command prompt while your DCM is running. Your motors will move accordingly.

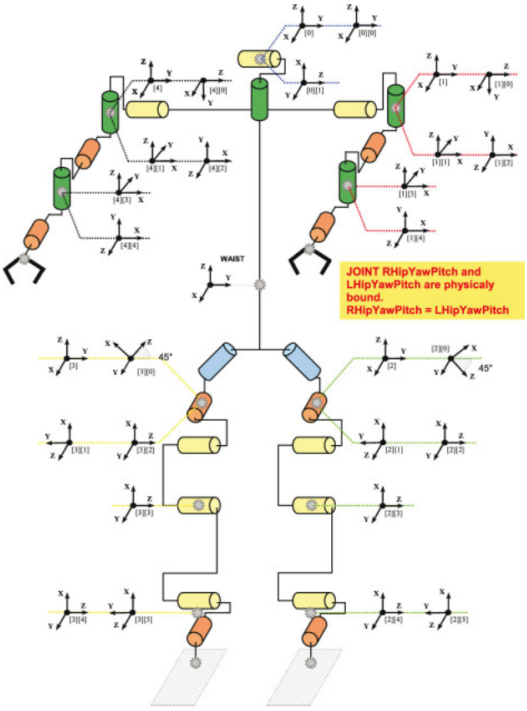

Fig. 4. Kinematics of the Nao [17].

B. Kinematic Body Parameters

The next step is to set up the correct kinematic and body parameters for your robot shown in figure 4. The body parameters are stored in the "Body" folder of your platform's library, under the body.lua file.

C. Adding a Camera

While working on implementing the DCM and kinematics of the robot, you can use a dummy camera driver while testing your code. As described, the dummy camera will yield random data, but is a good drop-in replacement if your own camera driver is not completed while working on other portions of the code.

Since the system runs on Linux, we require that a Linux compatible camera is used. You need to determine the focal length and other camera parameters. For testing the effectiveness of various parameters, we recommend using guvcview (for UVC cameras).

VIII. Conclusions and Future Work

We are in the stages of adding a test suite for our platform. In this way, we hope that extending features of the HRP will be easier. We will also add better debugging support for visualizing the current image processing and other data.

The code has been open source released, and is located at: `http://fling.seas.upenn.edu/~robocup/wiki/index.php`.

References

[1] (2010) Darwin iv - robocup. [Online]. Available: http://romela.org/robocup/

[2] T. Röferofer, T. Laue, J. Muller, A. Burchardt, E. Damrose, A. Fabisch, F. Feldpausch, K. Gillmann, C. Graf, T. J. de Haas, A. H. rtl, D. Honsel, P. Kastner, T. Kastner, B. Markowsky, M. Mester, J. Peter, O. J. L. Riemann, M. Ring, W. Sauerland, A. e Schreck, I. Sieverdingbeck, F. Wenk, and J.-H. Worch, "Bhuman team report and code release 2010," Deutsches Forschungszentrum fur Kunstliche Intelligenz and Universitat Bremen, Tech. Rep., 2009. [Online]. Available: http://www.b-human.de/file%5Fdownload/33/bhuman10%5Fcoderelease.pdf

[3] Robotics Trends Staff, "The urbi robotic software platform goes open source," *Robotics Trends*, May 2010. [Online]. Available: http://www.roboticstrends.com/design%5Fdevelopment/article/the%5Furbi%5Frobotic%5Fsoftware%5Fplatform%5Fgoes%5Fopen%5Fsource

[4] (2010). [Online]. Available: http://www.ros.org/wiki/nao

[5] (2010) Robotcub - an open framework for research in embodied cognition. [Online]. Available: http://www.robotcub.org/

[6] (2010) Openrdk website. [Online]. Available: http://openrdk.sourceforge.net/

[7] (2010) Orca robotics. [Online]. Available: http://orca-robotics.sourceforge.net/

[8] R. Ierusalimschy. (2004) Overview of the c api. HTML Documentation. [Online]. Available: http://www.lua.org/pil/24.html

[9] (2010) The Aldabaran RoboCup website. [Online]. Available: http://robocup.aldebaran-robotics.com/

[10] ROBOTIS. (2010) Overview of communication. HTML Specification Files. [Online]. Available: http://support.robotis.com/en/product/dynamixel/dxl%5Fcommunication.htm

[11] [Online]. Available: http://www.cyberbotics.com/nao/

[12] M. Vukobratovic and B. Borovac, "Zero-moment point - thirty five years of its life." *I. J. Humanoid Robotics*, pp. 157–173, 2004.

[13] J. Strom, G. Slavov, and E. Chown, "Omnidirectional walking using zmp and preview control for the nao humanoid robot," in *RoboCup 2009: Robot Soccer World Cup XIII*, ser. Lecture Notes in Computer Science, J. Baltes, M. Lagoudakis, T. Naruse, and S. Ghidary, Eds. Springer Berlin / Heidelberg, 2010, vol. 5949, pp. 378–389.

[14] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, and K. H. K. Yokoi, "Biped walking pattern generation by using preview control of zero-moment point," in *in Proceedings of the IEEE International Conference on Robotics and Automation*, 2003, pp. 1620–1626.

[15] B. Dirks, M. H. Schimek, H. Verkuil, and M. Rubli. (2008) Video for linux two api specification. Single HTML Specification File. [Online]. Available: http://v4l2spec.bytesex.org/spec-single/v4l2.html

[16] J. Brindza, A. Lee, A. Majumdar, B. Scharfman, A. Schneider, R. Shor, , and D. Lee, "Upennalizers robocup standard platform league team report 2009," University of Pennsylvania, Tech. Rep., 2009. [Online]. Available: http://www.seas.upenn.edu/%7Erobocup/files/UPennalizers%5Fteam%5Freport%5Fspl%5F2009.pdf

[17] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. L. 0002, B. Marnier, J. Serre, and B. Maisonnier, "The nao humanoid: a combination of performance and affordability," *CoRR*, vol. abs/0807.3223, 2008.