The Tekkotsu Robotics Development Environment

Ethan Tira-Thompson and David S. Touretzky

***Abstract*— Tekkotsu has grown from a specialized framework for development on the Sony Aibo to a general purpose robotics development environment with support for a variety of hardware, algorithms for autonomous operation, virtual simulation, and associated curriculum for undergraduate education. This paper describes the implementation of these features, provides examples of their use in research and education, and draws a comparison with other popular open-source robotics frameworks.**

I. Introduction

Writing non-trivial robot software demands proficiency in a variety of disciplines, such as computer vision, planning, control theory, machine learning, and human-robot interaction. Development is greatly aided by the availability of tools for data logging and simulation.

This high barrier to entry creates a demand for development toolkits which allow users to focus on specific areas of new development [1]. Ideally, these toolkits can also promote collaboration and reproducibility of research results. However, the wide range of hardware available for robot construction has also spawned a wide variety of toolkits, each originally specialized for a niche hardware configuration, which shapes the toolkit's overall architecture [2].

Tekkotsu originated on the Sony Aibo [3], a four-legged platform with moderate computational resources. Legged platforms emphasize the importance of real-time performance in order to produce smooth motion for walking. This is seen in Tekkotsu's architectural division of labor between a 'Motion' thread with real-time constraints and a 'Main' thread which performs longer-term decision making.

Fig. 1. The "flagship" Tekkotsu platforms, clockwise from upper left: Create/ASUS, Hand-Eye, Calliope, and Chiara hexapod

E. Tira-Thompson is a doctoral student in the Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh PA 15213 ejt@cmu.edu

D. S. Touretzky is a Research Professor of Computer Science, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh PA 15213 dst@cs.cmu.edu

This emphasis on efficient use of limited resources has served Tekkotsu well as it has expanded to a variety of low-cost systems suitable for university education, such as the iRobot Create and others of the Tekkotsu lab's own design (Hand-Eye, Calliope, and Chiara). A quick overview of these platforms shown in Fig. 1:

- *iRobot Create with netbook and mounting bracket:* $785
  Wheeled navigation, fixed netbook camera
- *Hand-Eye manipulator [4]:* $995
  3 DOF planar arm with pan/tilt camera, mounted to table and tethered to a desktop computer
- *Calliope [5]:* approximately $3000 (in development)
  iRobot Create base with non-planar manipulator and pan/tilt camera
- *Chiara hexapod [6]:* under $4000 (price fluctuates with component upgrades and availability)
  Legged system with planar arm, pan/tilt camera, on-board computation, battery power, and wireless communication

The preceding prices represent assembled hardware from RoPro Design, Inc. However, our hardware designs are open-source (GPL), allowing users to independently construct, extend, and customize our hardware as well as our software (LGPL).

Tekkotsu users have developed support for additional platforms. A group at the Technical University of Crete [7] has Tekkotsu working on the Nao humanoid from Aldebaran Robotics. And Road Narrows Robotics [8] has been using Tekkotsu to develop support for the Kondo KHR-2 mini-humanoid.

II. Design Goals/Constraints

A. Real-Time Performance

Tasks such as object tracking require low latency between sensor processing and motion generation. Similarly, legged locomotion requires smooth generation of effector trajectories. This type of real-time control becomes unreliable when network communication is involved. Further, avoidance of off-board computation simplifies user configuration and reduces points of failure. For these reasons, Tekkotsu focuses on maximizing its use of on-board computation. This focus allows use of shared memory communication between real-time and non-real-time threads, which avoids the potential for backlogs that can form in buffer-based communication.

Instead, behaviors in the Main thread directly call methods on shared 'motion commands', and these methods can immediately return status information or modify joint trajectories. This architecture is further described in section IV-B.

However, it should be noted that our current systems use a standard Ubuntu Linux installation, not a specialized 'hard' real-time variant, which does restrict our ability to make real-time performance guarantees on these systems. Although we have found this sufficient for our purposes, users with more rigorous requirements can employ specialized real-time kernels.

B. Ease of Use

Tekkotsu's core audience is academia [9]. Several dozen colleges and universities are using or have used Tekkotsu for either teaching or research. Via the Cognitive Robotics curriculum (Section V), students with varying levels of programming proficiency are introduced to the framework and use it to complete their assignments and laboratory work.

One key to this usage is a variety of developer tools to aid in the configuration, execution, and debugging of users' behaviors. These tools range from runtime generation of C++ stack traces for various error conditions (without prescient use of a debugger), to GUI tools for configuring color segmentation [10] (EasyTrain), robot kinematics (DH Wizard [11]), behavior monitoring (Storyboard [12]) and interaction (ControllerGUI).

Tekkotsu also includes Mirage, a fully 3D physics-based simulation environment, which has proven to be enormously useful for developing and debugging behaviors. This is not only due to providing a controlled, repeatable testing environment, but also because it allows students to continue development outside of the lab, without physical access to the robots.

One of the most successful tools is the state machine compiler, which allows succinct description of complex robot behaviors. Notably, it does this by extending C++ syntax, not supplanting it, so users still have direct access to all framework interfaces as needed.

The simplified syntax used for specifying state machines also makes the framework accessible to novice programmers, so that high school students in two CMU summer programs, the Summer Academy for Mathematics and Science (SAMS), and Andrew's Leap, have demonstrated the ability to quickly produce interesting behaviors.

C. Modularity vs. Integration

Modularity can be a double-edged sword: it allows abstraction, which benefits conceptual simplicity, maintenance, and reusability, but it can also restrict interoperability to a lowest common denominator and incur overhead translating between internal representations and interface standards. Overuse can also increase complexity rather than lower it.

Tekkotsu's approach varies as it provides programming interfaces at different levels of abstraction. At the lowest levels, drivers and events promote modularity, allowing independent plug-and-play implementations. At the highest levels of abstraction, the "Crew" components [13] are highly integrated in order to allow efficient reasoning regarding use of the robot's resources, such as where to direct its sensors, coordination of navigation and localization, etc.

D. Open Source

Documented and accessible source code is crucial in a field where so many technologies are still undergoing active research and development. We remain committed to publicly releasing both our source code and our hardware designs in order to promote interoperability and extension to new technologies.

III. High Level Features

A. Crew

The Tekkotsu Crew [13] is the highest level interface for robot control. Users formulate requests for actions to be performed, and the relevant components work together to satisfy these requests.

Currently, the crew consists of four components: Lookout, Map Builder, Pilot, and Grasper.

The Lookout is responsible for the sensors mounted on the "head", typically a camera and rangefinders.

The Map Builder uses the Lookout to search for a target or build a map of the local environment, converting camera images to top-down views.

The Pilot plans collision-free paths through the environment, and uses the Map Builder and a particle filter to perform localization within the environment.

The Grasper [14] accepts requests to manipulate objects, also specified by map-based data.

B. Vision Processing

The Dual-Coding vision system [15] parses image data to extract symbolic structures such as lines and ellipses, but also to process pixel-based relationships such as edges and connected components. These representations can be converted bi-directionally, from pixels to algebraic representations and back again as needed to perform efficient operations and present final results back to the user. Dual-Coding operations are automatically tracked and presented in a GUI interface as a derivation tree which allows users to easily step through the processing that has been performed.

There is also color segmentation, SIFT [16], and AprilTag [17] support available.

C. Localization and Mapping (SLAM)

Tekkotsu includes a particle filter to perform mapping and/or localization from a variety of sources, such as a range finder scan or bearing-only landmarks. [18]

D. Kinematics

Tekkotsu's kinematic descriptions can include both collision models and graphical models to provide planning, physical simulation, and display of the robot in a 3D environment. The kinematic description supports tree structures, with specification of which inverse kinematic algorithm should be

used for solutions at each link. Tekkotsu includes a generic gradient descent IK solver as well as analytic solvers for common three-link configurations.

E. Speech Generation

It is often hard to see status information on a small moving screen attached to the robot, or to split attention between the robot itself and a fixed desktop display. On Linux, Tekkotsu uses the MARY [19] text-to-speech engine from the German Research Center for Artificial Intelligence (DFKI). On Mac OS X, Tekkotsu uses the built-in text-to-speech interface provided by the operating system.

F. Locomotion

Legged locomotion is a non-trivial problem with a variety of approaches. Early Aibo-based development used a gait-based engine developed by the CMU RoboSoccer team CMPack'02 [20], as well as an alternate engine from the University of Pennsylvania [21]. We have since written a new engine, also gait-based, for the Chiara hexapod.

The choice of motion engine (gait generator or wheel controller) is made at compile time based on the target robot. The engines use a common interface, so user behaviors are mostly portable between platforms.

IV. Architecture

A. Deliberative vs. Real-time Processing

As noted in the introduction, Tekkotsu uses threads to separate real-time tasks such as hardware interaction, sound playback, and motion generation, from deliberative tasks such as planning and image processing. Sensor updates are considered a real-time task, as motion generation may depend on this data, for example to maintain balance or other dynamics. However image processing is considered a deliberative task as it is typically time consuming and not able to conform to useful real-time constraints.

B. Flow of Information

The canonical sense-think-act loop is used to illustrate the processing of the framework.

1) Sense: Tekkotsu provides a Hardware Abstraction Layer (HAL) which is responsible for device configuration and coordinating the movement of data.

A device driver can advertise one or more data sources. Each of the data sources listed in the HAL configuration will receive callbacks to initiate and terminate sensor activity when appropriate. (Besides starting or stopping the executable, users can also pause execution or freeze individual data sources on demand.) Data sources are responsible for pushing data into Tekkotsu as it becomes available, commonly spinning off a thread to block on or poll a resource.

Sensors that provide individual values, such as non-scanning rangefinders, contact sensors, torque sensors, etc. are collated into a flat array of sensor data. Sensors that provide arrays of values, such as cameras, are published as individual images immediately as they become available.

2) Think: The publish/subscribe pattern provides a popular method to efficiently distribute information between modules. Not only does it eliminate the overhead of repeated polling, it also allows publishers to query whether they have any subscribers and thus avoid unnecessary processing.

Tekkotsu uses a centralized publish/subscribe model, implemented by a global Event Router from which all events can be posted and received.

Tekkotsu's centralized event processing is one of its unique characteristics, particularly relative to other robotics frameworks, and has interesting consequences:

- Providing a centralized list of events types makes the system more transparent and comprehensible.
- Publishers can be added, removed, or replaced without any changes to subscriber code, because publishers and subscribers have no direct contact with one another.
- Events are normally processed on the local host with no serialization overhead. However, serialization is supported in order to subscribe to events on remote hosts.
- Users can extend the event hierarchy by subclassing the standard event types, allowing generic event operations to proceed naturally, while savvy subscribers can access the extended information via a simple C++ cast.
- When event processing (e.g., a new camera image) triggers a child event (e.g., object detected in the image), the centralized event queue completes processing of the parent before recursing on the child. This ensures all subscribers always receive events in temporal First-In-First-Out order.
- Although not strictly a feature of centralized processing, Tekkotsu's Event Router guarantees to publishers that all processing for an event has completed before the `postEvent()` call returns. This allows subclassed events to safely include direct references to generators' internal data structures without resorting to reference counting or semaphores, reducing the overhead of data marshalling.

3) Act: The motion process entails repeatedly polling active motion commands for their control values, and then sending these to device drivers' motion hooks.

Motion commands can implement arbitrary control schemes, such as walking, balancing, or simply playing back scripted trajectories. These motion commands are usually primitive and task specific, relying on event-based behaviors to do long term planning. For example, a gaited "walk" motion command will produce a cycle of leg motions to locomote in a specified direction, but relies upon a higher-level behavior to direct the walk around obstacles. A footstep-based motion command might receive a series of footstep positions to iterate through, planned by an external behavior.

Since motion commands are independent, two motion commands might try to concurrently move the same actuator. Tekkotsu's motion manager coordinates actuator values using a system of priority levels assigned to each motion command as well as a actuator weights which are used to perform a weighted average of values at the same priority level.

This management system allows some motion commands such as an emergency stop to override other motions using the priority level, and also allows motions to fade-out when they complete, which prevents instantaneous actuator motion if the motion command was in conflict with another. (If two motion commands are in conflict, the actuator moves to the weighted average of their requested positions—if one of these commands were removed without fading, the actuator would snap suddenly to the remaining command's position.)

Once the motion manager produces a final array of values for each actuator, which is passed to all device drivers' motion hooks. These drivers are responsible for transmitting the actuator values to their respective hardware.

C. Control and User Interface

To interact with and control the robot, Tekkotsu provides a set of GUI tools written in Java. These can be run from a remote machine and communicate with the robot over a network connection.

The main interface is dubbed the ControllerGUI, which is used to launch behaviors and bring up additional interfaces, such as a remote control for locomotion or controlling the robot's "head" (as defined by the robot's configuration). There are also data views for investigating the Dual Coding data structures encoding the robot's parse of the visual world or streaming video.

Independent tools provide additional user interfaces, such as the EasyTrain program to assist in segmenting color space for fast vision processing [10], or the Storyboard log viewer to investigate state machine execution [12].

One major tool is Mirage, shown with some additional interfaces in Fig. 2. Mirage provides 3D visualization (via Ogre3D [22]) with dynamic physics simulation (via Bullet Physics [23]). This allows many robot behaviors to be developed and debugged in a simulated environment, which has proven extremely useful.

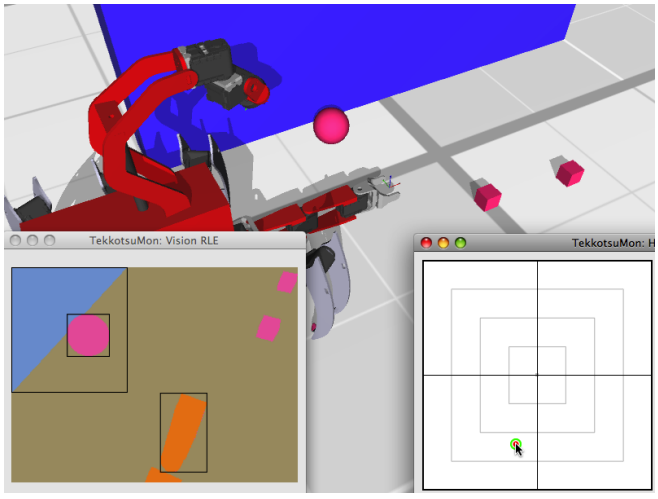

Fig. 2. A screen capture of the Mirage simulation environment (background) with additional windows. The lower left window illustrates the result of color-segmented image processing with object bounding boxes and the right window provides a remote control for the head. `http://www.youtube.com/watch?v=IZc2EftHVNQ`

Mirage supports multiple concurrent robots interacting in a common virtual world. Each robot makes a network connection, and then sends its kinematic configuration followed by a stream of desired actuator positions. It can make additional network connections to receive a stream of image and sensor data from the simulated robot.

Several advanced features are available as well. For example, behaviors running in Tekkotsu can send a message to the Mirage driver that a point on the robot should be constrained to its current position, which will be translated into a physics constraint in Mirage. This is used by the walking motion commands to allow "perfect" legged locomotion, as the feet are prevented from slipping or tipping over. However, the Mirage driver can also be configured to ignore these requests, enabling purely physics-based locomotion. Each of these modes is useful in different development contexts: sometimes a user may wish to move ideally to test a navigation behavior, other times they may wish to have more realistic locomotion errors to test the locomotion itself. Wheeled navigation is also supported and can similarly select between friction-based or "perfect" motion.

Mirage uses a simple XML-based communication protocol, which allows a variety of different tools to use it for visualization of robot state, regardless of whether this state derives from a physical robot or a simulation. This communication is independent of the Tekkotsu framework, allowing Mirage to be used in other environments. For example, the DH Wizard tool [11] provides interactive editing of kinematic configuration using the Denavit-Hartenberg conventions, relying on Mirage for visualization of the robot described by the kinematic tree.

D. Robot Definition and Behavior Portability

A unique feature of Tekkotsu is its handling of robot configuration. Each robot is defined primarily by a header file defining a series of preprocessor flags and a C++ namespace. The preprocessor flags advertise robot features, such as the existence of a camera, buttons, legs, etc. The namespace contains statistics for these features, such as the number of arms and legs, the number of joints in each limb, the number of sensors, etc. The namespace also defines string names for each of the outputs and sensors, and provides symbolic offsets to each entry.

The duality between string names and compiler symbols allows users to decide how they want to trade off between dynamic run-time lookup versus compile-time verification of features. For example, if a developer wishes to read the current value of the infra-red distance rangefinder, he or she could write:

```
state->sensors[IRDistOffset]
```

This assumes the target model provides an `IRDistOffset` symbol, relying on naming conventions to provide cross-platform compatibility. In practice, we often define aliases in the robot header files to foster compatibility between models. For example, the Chiara robot has three rangefinders. The `CenterIRDistOffset` symbol is

duplicated as `IRDistOffset` so that code which only requires a single rangefinder will default to using the center.

If the target robot does not have an IR rangefinder, the developer will receive a compile-time error that the robot behavior cannot be built for the target robot model. If the developer wishes to write their behavior more portably, perhaps to fall back to another algorithm if a rangefinder is unavailable, either a `#ifdef TGT_HAS_IR_DISTANCE` preprocessor directive could be used (to check whether the macro was set in the target model's header file), or the global `capabilities` map could be queried at runtime to see if the string name has been defined.

An additional use of the capabilities mapping is to query the features available on robot models other than the current host. This can be used to coordinate planning between robots in a heterogenous team. In some cases where a robot is reconfigurable, the capabilities map also provides a mechanism to announce the current run-time functionality of the robot, whereas the compile-time checking is limited to the union of all features that *might* be available.

Another important source of robot configuration comes from a model-specific kinematics file, which defines all the joint names and the coordinate transformations between joint reference frames. This XML file can either be edited by hand, or updated using the DH Wizard GUI tool. On startup, Tekkotsu loads the kin file for the target robot model. Mirage also obtains its kinematic information from this file.

The third and final piece of robot configuration comes from the HAL (Hardware Abstraction Layer) file. This file is read when the Tekkotsu executable launches, and is used to instantiate and configure the appropriate drivers. It is often useful to load alternative HAL configurations—for example, a generic 'Mirage' HAL configuration can be used to tell Tekkotsu to connect to the simulator instead of attempting to connect to physical hardware. Similarly, a different HAL configuration file can be used to replay logged data from disk.

E. Extensions to Off-Board Computing

Although Tekkotsu usually runs entirely on-board, there are some extensions to streamline off-board communication.

One of these is the ability to subscribe to events on a remote host. The request is relayed to the remote host where a proxy listener is created, and then matching events are serialized and transmitted to the local host. The subscriber on the local host then receives these events like any other, and can check the event's host field to determine its origin.

Another way to achieve off-board control is to run Tekkotsu on the robot and open a network connection to the interface provided for GUI-based teleoperation tools. This mechanism was used by the Pyro toolkit [24] to allow Python programs to teleoperate the Aibo. More generally, any Tekkotsu application can open a network connection to do custom communication with off-board or distributed resources.

The CommPort configuration of the HAL can also be used to send and receive data over a variety of communication links. This transplants the entire 'brain' of a robot to an external platform, which may be required if the on-board computation is only sufficient to run a thin client.

V. Cognitive Robotics Curriculum

Although the curriculum described in [25] was developed with the Aibo platform, we have since transitioned to use the Create/ASUS as the flagship robot, with the more sophisticated Calliope platform on the horizon.

The Cognitive Robotics course allows a high-level exploration of the breadth of algorithms and techniques used in the robotics domain. Requiring the students to re-implement all of these technologies in a traditional "bottom-up" approach would require several courses, and indeed is already available in traditional computer vision, machine learning, and other classes. The Cognitive Robotics course instead allows students to concentrate on exploring how to combine these techniques to complete assignments on the robots, with broader discussion of the strengths and weaknesses of the various tools at hand.

The course also provides some new material, examining how cognitive science theories of human information processing can provide inspiration to approach similar problems in robotics, such as is demonstrated in the Dual-Coding vision system.

For more information regarding Tekkotsu in education, visit wiki.tekkotsu.org.

VI. Comparison to Previous Work

Among other open-source robotics frameworks, the Robot Operating System (ROS) [2] stands out for its rapidly growing acceptance in the research community.

The Tekkotsu environment is similar to ROS in scope, with Tekkotsu's Hardware Abstraction Layer mirroring ROS's use of Player [26] for device control, and Mirage providing simulation similar to that of Gazebo. Tekkotsu event generators are analogous to "topics" in ROS, and various high level Tekkotsu interfaces such as the Crew would be called "services" in ROS.

However, these platforms differ significantly in the philosophical approach to abstraction. Player provides a character device model in the style of UNIX, perpetuated by ROS for serialized communication between processes. In contrast, Tekkotsu provides a structured data model in a single address space, minimizing serialization and IPC latency. Although Tekkotsu does provide mechanisms for off-board communication, this approach is the exception rather than the rule.

The platforms also differ in their approach to robot portability. As described in IV-D, various robot-specific capabilities can be declared at compile time. This provides automatic configuration of many features, for example whether the "walk" motion command should use a wheeled or legged implementation, or whether it should be left undefined to flag attempts to move on immobile platforms. In contrast, the distributed nature of ROS nodes implies they have no single global concept of the robot target, each requiring independent configuration, and lacking automatic verification of the

capabilities of a particular robot configuration. (However, the `roswtf` tool provides a degree of sanity checking regarding node communications.)

In an educational setting, the C++ class-based abstractions used by Tekkotsu are in line with the modern computer science curriculum, demonstrating practical use of inheritance, polymorphism, templating, functors, and namespaces. Using a single language also reduces conceptual barriers for students to move between modules, allowing them to easily "peek under the hood."

However, although ROS and Tekkotsu differ in many ways, they are not mutually exclusive. Just as Player serves a device driver role in a ROS network, Tekkotsu could do the same. Further, Tekkotsu's Crew and state machine mechanisms may be of interest for top-level control of the ROS network, using ROS's distributed mechanisms for either intermediary processing or additional device drivers. This is particularly attractive because the computation required for the highest levels of abstraction is usually relatively light, and keeping this onboard the robot affords contextually savvy error handling if remote resources become unavailable. (e.g., falling back on local computation, navigating towards better signal strength, alerting a human operator, etc.)

VII. Conclusion

Tekkotsu provides tools and algorithms to jumpstart robotics development on a variety of platforms. Although it features built-in support for specific robot models serving an educational curriculum, it also provides open source, documented interfaces for extension to new platforms and devices. Other popular environments such as the Robot Operating System provide similar capabilities, but differ in architectural focus and supported robot configurations. In particular, Tekkotsu's design emphasizes real-time embedded control befitting its early use of legged systems, which comes at the expense of easily distributed off-board processing. This may be addressed by complementary usage of distributed systems such as ROS.

VIII. Acknowledgments

This research was funded by National Science Foundation award DUE-0717705.

References

[1] J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, p. 132, 2007.

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[3] E. Tira-Thompson, "Tekkotsu: A rapid development framework for robotics," Master's thesis, Carnegie Mellon University, Pittsburgh, PA, May 2004. [Online]. Available: http://www.tekkotsu.org/media/thesis_ejt.pdf

[4] G. V. Nickens, E. J. Tira-Thompson, T. Humphries, and D. S. Touretzky, "An inexpensive hand-eye system for undergraduate robotics instruction," *SIGCSE Bull.*, vol. 41, no. 1, pp. 423–427, 2009.

[5] D. S. Touretzky, O. Watson, C. S. Allen, and R. Russell, "Calliope: Mobile manipulation from commodity components," in *Proceedings of the AAAI-10 Robot Exhibition*, Atlanta, GA, July 2010.

[6] (2010) The chiara robot. [Online]. Available: http://chiara-robot.org

[7] (2009) Tekkotsu over NaoQi. [Online]. Available: http://www.intelligence.tuc.gr/%7Erobots/ARCHIVE/2009w/projects/Orfanoudakis/

[8] J. B. Weinberg, W. Yu, K. Wheeler-Smith, R. Knight, R. Mead, I. Berstein, J. Croxell, and D. Webster, "Making intelligent walking robots accessible to educators: A brain and sensor pack for legged mobile robots," The 2008 Association for the Advancement of Artificial Intelligence (AAAI-08) Workshop on AI Education, Chicago, IL, Tech. Rep. WS-08-02, July 2008.

[9] D. S. Touretzky, "Preparing computer science students for the robotics revolution," *Commun. ACM*, vol. 53, no. 8, pp. 27–29, 2010.

[10] J. Bruce, T. Balch, and M. Veloso, "Fast and inexpensive color image segmentation for interactive robots," in *Proceedings of IROS-2000*, Japan, October 2000.

[11] N. Buroojy, O. Greeley, and L. C. Thorpe, "DH Wizard," 2010. [Online]. Available: http://www.cs.cmu.edu/afs/cs/academic/class/15494-s10/final-projects/2010/dhwizard/index.html

[12] A. Sangpetch, "Visualizing robot behavior with self-generated storyboards," Master's thesis, Carnegie Mellon University, Pittsburgh, PA, May 2005. [Online]. Available: http://www.tekkotsu.org//media/thesis_asangpetch.pdf

[13] D. S. Touretzky and E. Tira-Thompson, "The Tekkotsu "Crew": Teaching robot programming at a higher level," in *AAAI Symposium on Educational Advances in Artificial Intelligence*, Atlanta, GA, July 2010, pp. 13–14.

[14] J. A. Coens, "Taking Tekkotsu out of the plane," Master's thesis, Carnegie Mellon University, Pittsburgh, PA, Aug. 2010. [Online]. Available: http://www.cs.cmu.edu/ jcoens/MastersThesis.pdf

[15] D. Touretzky, N. Halelamien, E. Tira-Thompson, J. Wales, and K. Usui, "Dual-coding representations for robot vision programming in Tekkotsu," *Autonomous Robots*, vol. 22, pp. 425–435, 2007, 10.1007/s10514-007-9024-0. [Online]. Available: http://dx.doi.org/10.1007/s10514-007-9024-0

[16] D. G. Lowe, "Object recognition from local scale-invariant features," in *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*. Washington, DC, USA: IEEE Computer Society, 1999, p. 1150.

[17] E. Olson, "AprilTag: A robust and flexible multi-purpose fiducial system," University of Michigan APRIL Laboratory, Tech. Rep., May 2010.

[18] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *In Proceedings of the AAAI National Conference on Artificial Intelligence*. AAAI, 2002, pp. 593–598.

[19] M. Schröder and J. Trouvain, "The German text-to-speech synthesis system MARY: A tool for research, development and teaching," in *International Journal of Speech Technology*, 2001, pp. 365–377.

[20] J. Bruce, S. Lenser, and M. Veloso, "Fast parametric transitions for smooth quadrupedal motion," in *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*, A. Birk, S. Coradeschi, and S. Tadokoro, Eds. Berlin: Springer Verlag, 2002.

[21] D. Cohen, Y. H. Ooi, P. Vernaza, and D. D. Lee, "The University of Pennsylvania Robocup 2004 legged soccer team," Digital Equipment Corporation, University of Pennsylvania, Philadelphia, PA 19104, Tech. Rep., 1987. [Online]. Available: http://isl.ecst.csuchico.edu/DOCS/Robots/AIBO/DOCS/Robocup/2004/UPennalizers/UPenn04.pdf

[22] (2010) OGRE: Open sources 3d graphics engine. [Online]. Available: http://www.ogre3d.org/

[23] (2010) Game physics simulation. [Online]. Available: http://bulletphysics.org/

[24] D. S. Blank, D. Kumar, L. Meeden, and H. A. Yanco, "The Pyro toolkit for AI and robotics," *AI Magazine*, vol. 27, no. 1, pp. 39–50, 2006.

[25] A. B. Williams, D. S. Touretzky, E. J. Tira-Thompson, L. Manning, C. Boonthum, and C. S. Allen, "Introducing an experimental cognitive robotics curriculum at historically black colleges and universities," in *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on computer science education*. New York, NY, USA: ACM, 2008, pp. 498–502.

[26] R. T. Vaughan and B. P. Gerkey, "Reusable robot code and the Player/Stage project," in *Software Engineering for Experimental Robotics*. Springer-Verlag, 2006, pp. 267–289.