Integrating ROS into Educational Robotics:
Bridging the Gap between Grade School and Grad School
Ross Mead, Stefan Zeltner, and Maja J Matarić
University of Southern California
rossmead@usc.edu, zeltner@usc.edu, mataric@usc.edu

Integrating ROS into Educational Robotics: Bridging the Gap between Grade School and Grad School

1 Introduction and Motivation

Robots and robot competitions have been used throughout the entire STEM (science, technology, engineering, and math) education pipeline [8]. However, a gap has been identified in the focus and reinforcement of educational robotics concepts between K-12 and higher education [3]; for example, K-12 robot competitions tend to focus more on open-loop, reactive, and/or rule-based solutions with limited sensing and processing, while university-level robot competitions tend of focus more on closed-loop, deliberative, and/or inference-based solutions with advanced sensing and processing. ROS (Robot Operating System) [7] is an open-source robotics software framework that has resulted in reusable software packages [9] developed and shared by a community of robotics researchers, professionals, and hobbyists [15]. Making such advanced tools and technologies accessible to K-12 students could help bridge the education gap between K-12 and university robotics and promote more artificially intelligent (AI) solutions.

In this work, we demonstrate the feasibility of installing and using the high-level ROS software framework [7] on a low-cost educational robotics microcontroller, the KIPR Link [4]. We first describe the ROS and its capabilities for robotics applications [7]. We then discuss the KIPR Link [4], and present a series of steps to install ROS on the microcontroller; we also present an example system in which a Link-controlled mobile robot is capable of mapping, localizing in, and navigating its environment. We conclude with a discussion of the implications of ROS in educational robotics, and steps to make the technology more accessible to K-12 students.

2 ROS: Robot Operating System

The ROS Wiki [10] describes ROS as follows:

> ROS provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license.

2.1 ROS Framework

A ROS system is a distributed network of so-called *nodes* [9], which are processes for a specific task (e.g., one node for controlling wheel velocities and one node for path-planning). The *ROS master* [9] is the central process of every ROS system and serves as a switchboard for all nodes, allowing them to establish communication with one another. Nodes communicate with each other using one of three methods: (1) messages, (2) services, and (3) actions.

Messages provide unidirectional (one-way) communication [9] (Fig. 1a). Each message is sent over a *topic* [9]; for example, a reading from a distance sensor, such as an infrared rangefinder or sonar, might send messages over a topic called, "`distance_reading`". ROS utilizes a *publish/subscribe paradigm* [9] to connect nodes through these messages; for example, one node might "publish" (send) distance sensor readings over the "`distance_reading`" topic, and another node might "subscribe" to (receive) messages on this same topic, thus, allowing these nodes to share information about distance sensor readings. Note that zero, one, or many nodes may publish on a topic, and zero, one, or many nodes may subscribe to a topic. Also note that these nodes are communicating asynchronously (i.e., at different rates).

Services provide bidirectional (two-way) communication [9] (Fig. 1b). A service can be thought of as a function that can be called on the same or different computers/processes; it can be called with specific input parameters (referred to as the "request") and will respond with the appropriate output value (referred to as the "response"). One or many services are implemented and executed within a single node, called the *server* of these services; other services may be implemented in other server nodes. Service requests come from one or many calling nodes, referred to as *clients* of the service. Thus, only a single server node will respond to any number of client requests for a particular service (contrast this with messages, which can be published from many nodes). Service calls can be used synchronously (blocking) or asynchronously (non-blocking).

Actions (*actionlib*[*]) provide multi-directional (in this case, three-way) communication (Fig. 1c). An action is nearly identical to a service, differing in three ways. First, actions typically take much longer to execute, whereas services are practically instantaneous; for example, "search for a ball" would be an action, while "add two numbers" would be a service. Second, actions using different but analogous names — an action *goal* is similar to a service request, and an action *result* is similar to a service response. Third, because actions are executed for an extended period of time, they also provide *feedback*, information about the status of the action that can be sent at any time; zero, one, or many feedback messages may be sent during the execution of an action.

[*] Henceforth, the asterisk (*) will denote a ROS *package* [14], for which there is a corresponding ROS Wiki [10] page with the URL http://www.ros.org/wiki/<package>. This format will be used throughout the paper.

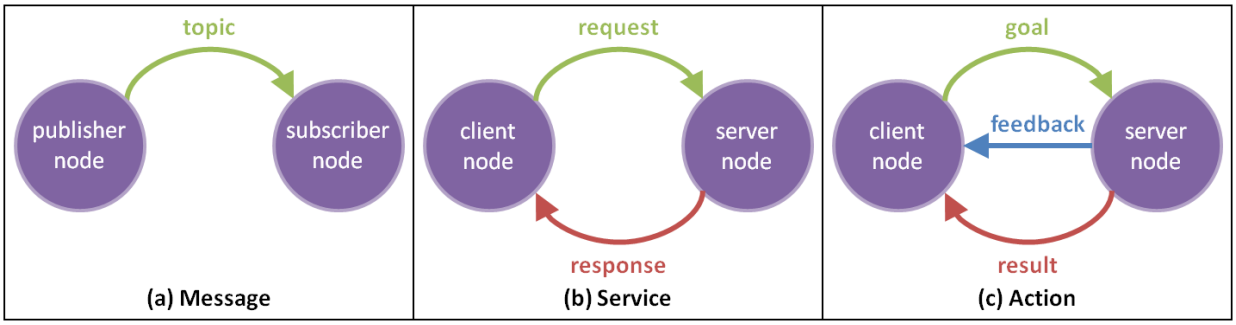

Figure 1: ROS inter-node communication methods: (a) messages, (b) services, and (c) actions.

2.2 ROS Software Packages and Development

Using the ROS framework, a software developer can create nodes that interact with each other (as well as nodes defined by other developers) using messages, services, and/or actions. A collection of ROS nodes and tools is often consolidated into a distributable ROS *package* [9]. ROS offers a wide variety of state-of-the-art high-level open-source software packages developed by leading robotics researchers around the world for use by the ROS community [15].

ROS is available for a number of robots (e.g., the PR2, iRobot Create, LEGO NXT) [15], operating systems (e.g., Linux, OSX, Windows) [11], and languages (e.g., C++, Python, Java) [12]. The ROS Wiki [10] offers detailed instructions for installing ROS on various platforms [11], and also provides a number of beginner, intermediate, and advanced tutorials that explore ROS in simulation [16]; thus, a physical robot is not needed to become familiar with ROS.

Some ROS packages might not function on some platforms due to resource limitations, such as processing power and storage capacity. Thus, installing ROS on a proprietary computer system, such as the KIPR Link [4], poses a difficult challenge — this is described in the next section.

3 ROS on the KIPR Link

The *KIPR Link* is a Linux-based robot controller that "aims to be simple enough for a middle schooler, powerful enough for a researcher, and versatile enough for the hobbyist" [4] (Fig. 2). The Link provides adequate processing power, and also offers more sensor and motor interfaces than other low-cost ROS-compatible controllers (e.g., the LEGO NXT or Android devices).

The KIPR Link comes with a single-core 800MHz ARM CPU and 128 MB of system memory [4]. Thus, it is infeasible for more processing- or memory-intensive ROS systems to run entirely on the controller. Fortunately, the flexible design of ROS allows nodes to be executed across multiple devices (e.g., laptop, desktop, or another Link) communicating over a wireless network. This allows the user to offload more complex nodes for external processing.

Technical Specifications
- Ångström Linux operating system
- Open-source robot control software
- Integrated color vision system
- Infrared receiver and emitter
- 800 MHz ARMv5te CPU
- 128 MB RAM
- 320 x 240 back-lit LCD touchscreen
- 802.11 b/g WiFi
- Internal speaker
- 1 physical button

Software-controllable Input/Output
- 4 1-Amp motor outputs with built-in PWM and PID control
- 4 standard servo outputs
- 8 analog inputs
- 8 digital input/output ports
- HDMI out
- 2 USB 2.0 ports (host)
- 1 micro USB port (slave)
- 1 TTL serial port
- 6 software buttons

Figure 2: The KIPR Link robot controller, and its technical and input/output specifications. [4]

The bundled Ångström Linux distribution and its package manager are not supported by the *rosdep*[*] command, which is a powerful tool used to install missing ROS package dependencies [9]. As a result, the user would have to manually resolve and install dependencies if additional packages are desired or required. To alleviate this burden, we have developed procedures to automate the ROS setup process, introduced below and comprehensively described in [13].

3.1 ROS Setup

The KIPR Link controller boots into an easy-to-use graphical user interface by default, but we need access to the underlying Linux system. There are two ways to gain access: (1) attach a USB keyboard and switch to an alternative login screen, or (2) use a Secure Shell (ssh). Both approaches are comprehensively described on our project page [13] and our technical report [17].

With this access, we can then install the ROS prerequisites, which are not included in the Ångström Linux distribution delivered with the Link. The controller does not have enough onboard and system memory for this, but the memory can be extended with a USB flash drive. An advantage of this approach is that the flash drive can contain the entire ROS environment and is, thus, kept separate from the Link firmware.

Please visit [13] for up-to-date step-by-step instructions for ROS installation onto a USB flash drive, as well as other useful resources, such as tutorials for ROS-Link development, a list of supported ROS packages, and instructions to build ROS packages from source.

3.2 Example ROS System

To demonstrate the feasibility and utility of a high-level robotics framework on a low-cost educational platform, we implemented an example system in which a Link-controlled robot used existing ROS packages [14] to generate a 2D map of its environment, determine its location within that environment, and navigate (via joystick or autonomous path-planning) while avoiding obstacles [13]. The *iRobot Create* [2] mobile robot platform was used for this application. We

mounted eight infrared rangefinder sensors [6] uniformly around the robot to provide distance information for mapping, localization, and safe navigation. The Create and sensors are connected to the Link. Movement commands can be provided via a PS3 controller or keyboard, which are connected to an external computer to communicate to the robot over the wireless network.

We used *libkovan* [5], the Link controller library provided by KIPR, to interface with the Create and sensors. We implemented the *ros4link*[*] package, which provides wrapper code around this library to allow users to interface with it using ROS [13]. For this application, we wrote a *robot_driver* node to subscribe to messages for Create movements ("`cmd_vel`") and publish messages for the current state of the Create — robot pose information (position and orientation) is published on "`odom`" and "`tf`" (see *tf*[*]) topics. The node also publishes all eight distance readings from the infrared rangefinders [6] on a "`scan`" topic, which is used by *gmapping*[*] for mapping, *amcl*[*] for localization, and *move_base*[*] for path-planning, navigation, and obstacle avoidance. The ROS *joy*[*] package is used to publish movement messages from the PS3 controller or keyboard; we wrote a *robot_mover* node that subscribes to these messages, allowing the user to manually navigate the robot while creating a map of the environment. The map of the environment and the robot location within it, as well as the planned path for navigation, can be visualized in real time in *rviz*[*]. The resulting network of ROS nodes is illustrated in Figure 3.

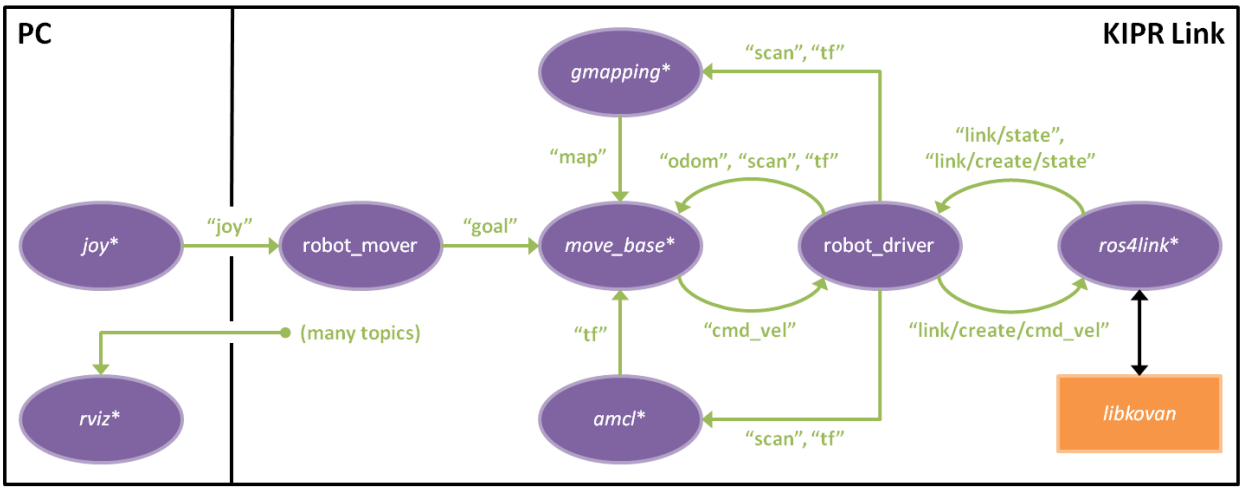

Figure 3: The distributed network of ROS nodes for our example system.
Note that some nodes, connections, and other details are omitted for clarity.

4 Conclusions and Ongoing Work

We have demonstrated the feasibility of using the ROS software framework on the KIPR Link. ROS offers a number of advanced tools for advanced robot operations, such as mapping, localization, path-planning, navigation, and obstacle avoidance. The KIPR Link offers adequate processing and a variety of sensor and motor interfaces to take advantage of the ROS framework and packages. Making these tools and technologies accessible to K-12 students could help bridge the education gap between K-12 and university robotics and promote more AI solutions [3].

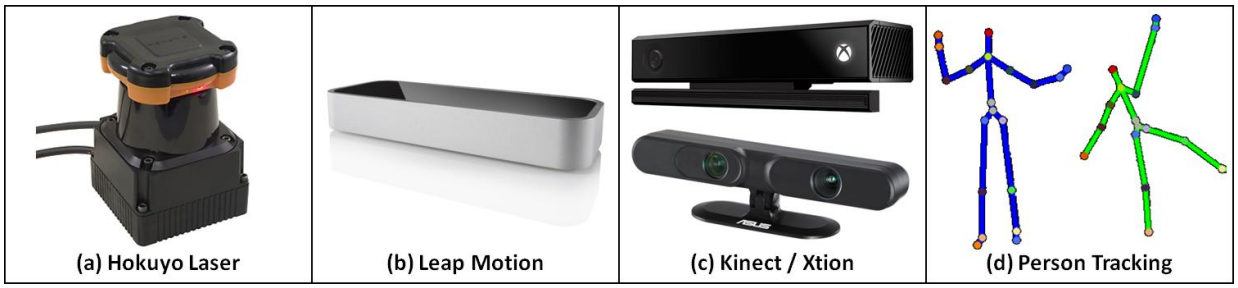

Figure 4: More advanced sensor capabilities: (a) Hokuyo laser rangefinder, (b) Leap Motion, (c) Microsoft Kinect and Asus Xtion, and (d) person tracking from the Kinect and Xtion sensors.

We are currently working on making our ROS install scripts easier to customize, setup, and extend. We are also providing Link interfaces for more advanced sensors, such as the Hokuyo laser rangefinder (Fig. 4a), the Leap Motion (Fig. 4b), and the Microsoft Kinect and Asus Xtion (Fig. 4c). These sensors further promote AI solutions, and create opportunities to investigate *social* robotics applications (e.g., human-robot interaction) in STEM curricula [1] (Fig. 4d).

5 Acknowledgments

This work is supported in part by NSF grants IIS-1208500, IIS-1117279, and DRL-1139415. We thank the KISS Institute for Practical Robotics (KIPR) for their dedicated support of this project.

References

[1] D. V. Lu and R. Mead. Introducing students grades 6-12 to expressive robotics. *Proceedings of the 7th International Conference on Human-Robot Interaction (HRI-12)*, 411-412, 2012.

[2] iRobot. iRobot Create. http://www.irobot.com/us/robots/Educators/Create. 2013.

[3] J. R. Croxell, R. Mead, and J. B. Weinberg. Designing robot competitions that promote AI solutions: lessons learned competing and designing. *Technical Report of the AAAI Spring Symposia (SS-07-09)*, 29-34, 2007.

[4] KIPR. KIPR Link. http://www.kipr.org/products/link. 2013.

[5] KIPR. libkovan – kipr GitHub. https://github.com/kipr/libkovan. 2013.

[6] KIPR. Rangefinder Sensor – Botball Store. http://botballstore.org/product/rangefinder-sensor. 2013.

[7] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, A. Y. Ng. ROS: an open-source Robot Operating System. *ICRA Workshop on Open Source Software*, 2009.

[8] R. A. Mead, S. L. Thomas, and J. B. Weinberg. From grade school to grad school: an integrated STEM pipeline model through robotics. *Robots in K-12 Education: A New Technology for Learning*, 302-325. 2012.

[9] ROS. Concepts – ROS Wiki. http://www.ros.org/wiki/ROS/Concepts, 2013.

[10] ROS. Documentation – ROS Wiki. http://www.ros.org/wiki, 2013.

[11] ROS. Installation – ROS Wiki. http://www.ros.org/wiki/ROS/Installation. 2013.

[12] ROS. Introduction – ROS Wiki. http://www.ros.org/wiki/ROS/Introduction. 2013.

[13] ROS. KIPR Link – ROS Wiki. http://www.ros.org/wiki/Robots/KIPR_Link, 2013.

[14] ROS. Packages – ROS Wiki. http://www.ros.org/browse, 2013.

[15] ROS. Robots – ROS Wiki. http://www.ros.org/wiki/Robots, 2013.

[16] ROS. Tutorials – ROS Wiki. http://www.ros.org/wiki/ROS/Tutorials. 2013.

[17] S. Zeltner, R. Mead, and M. J. Matarić. Installing ROS on the KIPR Link robot controller. *Technical Report of the USC Center for Robotics and Embedded Systems (CRES-13-001)*, 2013.