Easily-programmable Corobots for Student Use

Zack Butler, Max Bogue, Ravdeep Johar and Balaji Iyer
Department of Computer Science
Rochester Institute of Technology

***Abstract*—As mobile robots become more prevalent in our society, there are numerous potential applications for them. However, current educational opportunities tend to focus on the robotics aspects rather than higher-level applications. In this project, we are developing a system that takes care of the robotics aspects and provides a fully-capable yet fully programmable platform to allow students with interests in areas outside robotics can learn the traditional computing concepts within the robot context. These interactive robots, or "corobots", will operate within the department alongside people while performing tasks as programmed by the students. Here we present the overall system architecture, in which we solve fairly traditional problems but in a robust way that will allow for general use, and a student API which can control the robots at a high level while integrated with arbitrary other code.**

I. Introduction

In recent years, mobile robots have become more common in the daily lives of everyday people, at least in certain niche areas. As they become more capable and less expensive, we can expect to see them used for a wider variety of personal applications. The term "corobot" is used to connote this idea of robots working alongside humans instead of in isolation. However, the possible applications of such corobots are still relatively unexplored. In the Computer Science department at RIT, we are interested in giving all of our students the chance to interact with and program mobile robots, with the goal that they can learn about robotics and discover their potential, even as they learn content from all areas of computer science. That is, we are developing a system that does not target introductory programming, which has been well explored using robots, or robotics education per se. Instead, the robots presented here offer an easily-programmable system to provide a physical context in which to explore any topic in computing.

In the overarching project, our goal is to develop a number of assignments for different courses in which students are exposed to the robots while they learn and program traditional computing topics. The goal of the robot system itself is then that computing students can easily write and execute programs such as the example shown in Fig. 1. While this particular instance is clearly simplistic, its successful execution requires a robot capable of robust navigation. Likewise, when programs such as this one are possible, robotic actions can be easily integrated into more complex programs that are trying to solve other tasks, such as networking, data collection and management, or other applications yet to be conceived.

In order to create a system that is both very capable in terms of navigation and interaction, but also inexpensive, we have made several customizing design choices. That is, by developing a system that is only expected to operate in one building (albeit one that is over 30×100 m in area) and through a limited API, we can improve the overall performance under these specific conditions. The customizations are described in detail throughout the paper. Also, in general we have chosen to build a robot that is intended to be fully integrated with the department. That is, we have engineered the environment to enable easy localization, but done this in such a way that humans can also participate easily with the robots. Likewise, by limiting the way that the robots can be programmed, we can simplify our software architecture and provide a level of safety while also lowering the learning curve that might otherwise be present when programming such a robot.

```
def main():
  with Robot("corobot2.rit.edu") as r:
    pos = r.get_pos().get()
    print(pos)
    p = r.nav_to("Vending").wait()
    r.request_confirm("Buy me a soda please",\
          timeout=50).wait()
    r.nav_to_xy(pos[0],pos[1]).wait()
```

Fig. 1. Simple example of Python code that can be used to control one of our corobots.

A. Related work

Recent advances have led to significantly less expensive mobile robots while hardware such as the Microsoft Kinect has brought greater capability into more users' hands. However, the robots that have been created have been primarily designed for people interested in robotics per se. One interesting exception is the Cobot developed at CMU [1], [2]. In this system, robots wander around the hallways and interact with people in their environment, much like our system will be able to do. However, the focus of that project is in fact the interaction, and the directives given to the robots are not in the form of programs but rather service requests. Our focus is on delivering a platform that can be programmed by novice programmers but that still possesses navigational functionality. We are also interested in human-robot interactivity and will pursue this avenue once our system is more mature.

In contrast, a variety of educational robots have been used to teach introductory programming topics in a compelling way. Examples of this type of work include the work of Matarić [3], Lauwers et al [4] and the Institute for Personal Robots in Education [5]. However, the robots used in these projects

978-1-4799-2722-7/13/$31.00 ©2013 IEEE

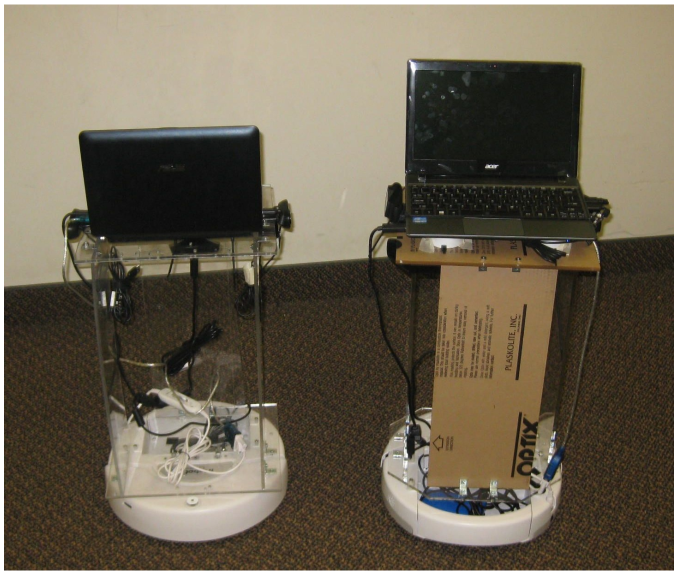

Fig. 2. The first two corobot prototypes, which differ only slightly in their construction. The first prototype (on the left) is facing toward the camera, while the other is facing away.

tend to be rather simple — in fact, Lego Mindstorms are often used, though the projects mentioned above involve more robust hardware — such that they can be easily purchased in large quantities or by individual students and easily maintained by novices. Thus, their capabilities are largely limited to local navigation and (often quite varied) sensing. These robots are certainly sufficient for the intended purposes, and robotics has been shown to be an exciting domain for students to learn how to program [4]. However, we are interested in a more capable system that lives within, and navigates around, the department instead of being a more "personal" robot, enabling the students to explore higher-level applications and computing topics.

B. System Framework

In order to enable the type of programmable robot that we have in mind, our overall system takes the form of several independent robots that primarily interact with a central server. The robots accept connections over the network, but the connections involve only the very high-level API that we have designed. User code can then connect to a robot either directly (for current testing) or via our server (for general student use) and request navigation, status updates, or perform other interactions within a general-purpose program of the user's design. In the remainder of the paper, we first present the robot hardware, followed by the software that sits on the robot to provide general navigation and interaction, and finally the user API and other code that runs off of the robot platform.

II. Hardware design

The goal for the hardware of the robot is to be relatively inexpensive (under US $1000), easily reproducible, and capable of robustly navigating our building while also interacting conveniently with people in the environment. Overall, our

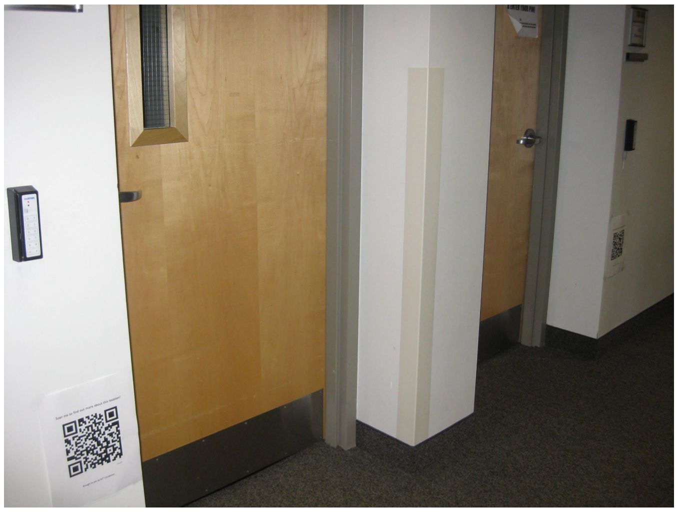

Fig. 3. Hallway in Golisano Hall of RIT showing QR codes mounted next to each doorway.

design is quite similar to the popular Turtlebot platform, though we have made a few different choices in our design. Like (some forms of) the Turtlebot, we currently use an iRobot Create as the base, primarily due to its cost and ubiquity. We also use a Kinect for obstacle detection and avoidance, with the thought that we can pursue more complex algorithms such as gesture recognition in future applications. Our first two robot prototypes, which differ only in the specific models of cameras and computer used as well as slightly in their construction, are shown in Fig. 2. This figure shows one difference between our design and the Turtlebot, in that the computer that controls the robot is left exposed and open at all times. In addition, the robot is somewhat taller than the Turtlebot, and future versions of our robot may be yet taller as long as stability can be maintained. This allows the robot to show a simple status monitor to, and request assistance from, any user passing by, and future versions of this monitor will allow more complex interactions such as providing tours of the building.

A. Cameras and barcodes

The most significant way in which our hardware differs from the Turtlebot is in the use of two side-facing cameras to detect QR codes placed throughout the building. A photo taken in one of the hallways in our building is shown in Fig. 3. The barcodes are primarily for assistance with localization, as done in similar fashion in several previous works. For example, the well-known SAGE project [6] used colored landmarks visible at all times to maintain the robot pose for a tour-guide robot within a museum, while the AprilTag system [7] performs full 6-DOF localization and has been shown to be very robust with respect to camera irregularities and lighting, partly due to its use of simple code images. For our project, we have specifically chosen QR codes so that they can be used by humans in the building as well. The work of Lin and Chen [8] also uses QR codes, but on the ceiling — these have the advantage of being more often visible, but mounting them on the walls gives us a system that is more interactive with passers-by. Also, because our robots' cameras are at a fixed height and horizontal, the accuracy of the robot localization is fairly robust to the mounting of the code on the walls.

In our setup, a QR code outside a faculty office (such as room 3651, the office of the first author) will correspond to a URL with a well-defined form (in this example, `http://www.cs.rit.edu/˜robotlab/Office3651`). When a robot detects and decodes the barcode, as described in detail below, it uses the last portion of the URL as a lookup into a table of barcode locations and can thus determine its location. However, when the code is detected by a QR code reader such as on a smartphone, the URL in question provides a redirection to the web site of the person in the office (`http://www.cs.rit.edu/˜zjb/`). Similar redirections are in place for research labs, while other barcodes that are in less distinct locations such as the building atrium currently have a default web page associated with them.

III. Robot Control Software

In the interest of space, we briefly explain those components which are fairly standard, while giving more detail on those that are more specific to our project. We have built our system using the Robot Operating System (ROS), enabling the use of some existing components, but have also developed many components on our own that are specific to the particular environment and robot we have. This simplifies the overall architecture so that it can be easily maintained and improved upon without being too large to handle — we are not concerned with having a highly flexible system since we are building a specific robot for a specific domain. The overall software architecture present on the robot is shown in Fig. 4. Several nodes were used directly from the ROS libraries, namely those that interact directly with the robot hardware and with the Kinect. Data sharing between nodes was done using ROS topics; most topics use predefined message types but for the `*pose` topics we created our own simplified planar pose message type. The discussion of our custom nodes follows.

A. Manager

The main job of the manager is to accept connections from user code and then to take the commands given in the user programs and turn them into appropriate values of ROS topics for the remainder of the nodes. It also subscribes to the relevant ROS topics so that the user code can be informed of the robot's progress (e.g. whether a goal has been reached, or whether the request to humans in the area has been acknowledged). This is overall fairly simple, but by making it a separate node we can easily add more functionality to the API by giving the manager access to other topics as necessary. The one challenge here is that the other end of the network connection is being maintained by arbitrary student code, which could easily include infinite loops and so on. Thus, we need to make sure the manager does not hang up waiting for the other end of the connection to terminate. The `asyncore` library in Python allows us to easily handle asynchronous network

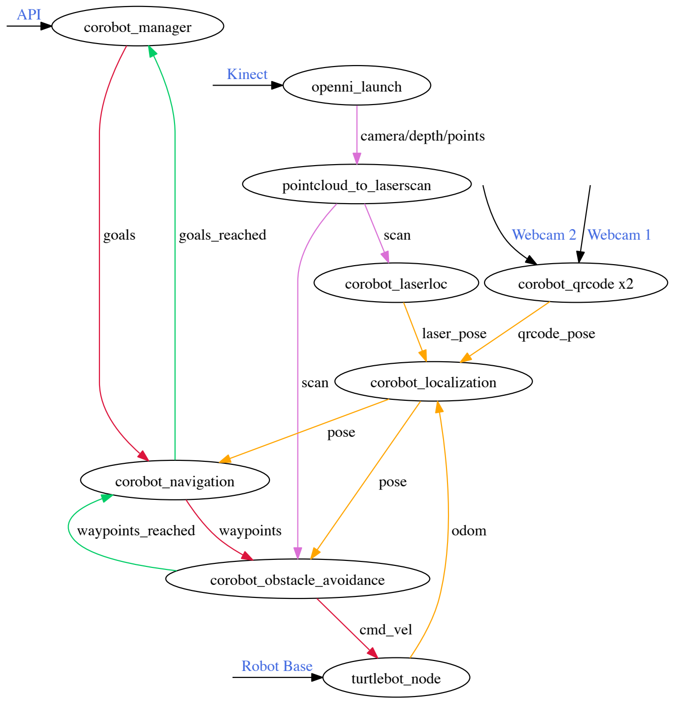

Fig. 4. Software architecture on the robot itself. Each oval is a ROS node — nodes that start with "corobot" were made for this project, whereas other nodes are provided by ROS. (A few ROS-provided nodes as well as the `corobot_map` and `corobot_monitor` omitted for clarity.) Lowercase-named edges are ROS topics. Edges are color coded for visual clarity only: red for commands, green for notifications, purple for range sensor data, orange for pose data. The API edge refers to commands coming across the network from the client.

communication with timeouts, and as such we have written this node in Python to take advantage of this library.

B. Navigation

For navigation through the building, we wish to provide the ability to plan and execute a path to any reachable location (i.e. office and classroom doorways, though perhaps not inside offices). Rather than perform an A* search over a dense map of the environment, we have used our prior knowledge to construct a waypoint graph. The waypoints are named, and their location and neighbor information is stored on board each robot. In this way, student programmers can easily specify destinations (such as in Fig. 1); each doorway has a waypoint with a meaningful name. In addition, we put waypoints down the center of each hallway so that the robot can always move in a straight line to at least one waypoint.

In the context of the robot software, we use a simple node called `corobot_map` to load the waypoint data from a file and provide a ROS service to provide the map to other nodes, thus avoiding having to reload the map multiple times. The nodes requiring map data (currently just the navigator, but future Kinect-based localization may use it as well) can query individual components or the entire map data structure.

When a navigation command is given to the robot, the search process looks for waypoints near the start and goal locations. In the case of navigation to a waypoint, clearly the goal is itself already on the roadmap, but the start will always be the robot's current position and thus not on the roadmap, and the goal can likewise be specified by arbitrary coordinates. To connect the start point to the graph, we do a line trace through the floor plan of the building to find all waypoints that are directly visible. The closest few of those are taken as the start "zone" (if none are within a maximum radius, just the closest is taken). A similar search is done for non-waypoint goal location. The A* search is then given a modified waypoint graph in which these extra edges are present. The result of this search, a sequence of waypoints, is published for the obstacle avoidance node to attempt to reach in turn.

C. Localization

Localization is a standard problem in robotics. Especially for a simple robot platform such as the Create, odometry can be very unreliable and the external sensors are of paramount importance. In the Turtlebot platform, EKF-based localization is provided that uses a small IMU and/or visual odometry to compensate for odometric error. In our robots, we have found (as have others) that the rotational odometry of the Create is especially poor when carrying a significant payload. However, here we can again take advantage of being able to constrain our problem space. Specifically, the obstacle avoidance code that actually drives the robot produces only a small set of rotational velocities, and so we calibrate the odometry specifically for those values within the localization node itself.

We can also include traditional Monte Carlo localization using the Kinect, however the floor plans that we have available do not include a significant amount of furniture around the building that is of similar height to the robot. As such the utility of such an approach is limited until (as we have plans to implement) the robots supplement the map with sensor data as they work. However, the combination of the QR codes and calibrated odometry is generally effective for our current use, and we are also developing a recovery protocol that will enable the robot to recover its position if it becomes lost by wandering until it locates a barcode.

D. Barcode-based localization

As mentioned, we use QR codes located throughout the building to provide absolute localization whenever one code is in view of the robot's cameras. However, we do not simply use the barcodes to give the general location, but use the size and shape of the code within the image to pinpoint the robot's location. In order to do this, we need to know how the code will appear as the relative location of the camera changes. We have used the open-source ZBar library [9] to detect and decode the barcodes, and written code that uses the results of this image processing to locate the robot.

In the most basic case, we can determine the distance of the camera By assuming pinhole camera model, this is given as $z = f * d/w$ where f is the focal length of the camera, d is the size of the actual barcode and w the width of the barcode in the image. In our case, we fix d at 5 inches for

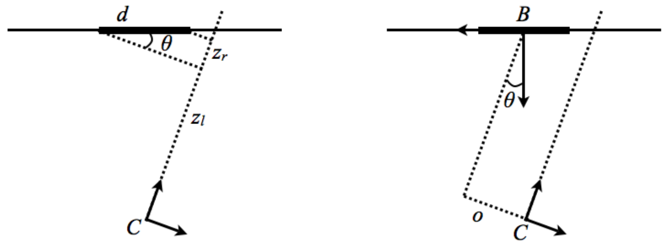

Fig. 5. Geometry of the camera relative to a barcode on the wall (two top views). The left diagram shows the distances to the two edges of the barcode used to compute the relative angle θ, while the right diagram shows the computations for the pose of the camera (C) relative to the coordinate frame located at the center of the barcode (B).

all barcodes (we use English units so that a 1000 pixel wide code can be printed at 200 dpi and be confident in the exact size of the code). With our current cameras, this allows us to reliably detect barcodes from approximately 20 centimeters up to 2 meters away from the robot. Rather than relying on the camera specifications to determine f, we simply capture images at known z values to determine f empirically.

However, the robot (and thus the camera) may of course be located at any planar pose relative to the barcode, and this information can be determined from the image as well. We do assume that the barcodes are placed vertically and the cameras are likewise well-placed on the robot, but again our system allows these assumptions to be easily enforced. The effect of the robot's pose is that the barcode will be off-center in the image and have a perspective projection. These factors can be used to compute the location of the camera with respect to the barcode (${}^{B}C$), as shown in Fig. 5. When the barcode is not perpendicular to the camera, the left and right sides of the barcode will be at different distances from the camera. With these distances (determined by the pinhole model as above), we first determine the rotation of the barcode with respect to the camera axis as

$$\theta = \arcsin((z_r - z_l)/d).$$

We can then compute the pose of the camera relative to the coordinate frame B located at the barcode as shown:

$$\begin{aligned} {}^{B}C_x &= z\sin\theta + o\cos\theta \\ {}^{B}C_y &= z\cos\theta - o\sin\theta \\ {}^{B}C_\theta &= \pi - \theta \end{aligned}$$

where o is the offset of the barcode within the image, converted into real-world dimensions.

Once the location of the camera relative to the barcode is known, the robot looks up the absolute location of the barcode in a lookup table that we have created. Again, we take advantage of the opportunity to engineer the environment to produce this table. We transform the local position of the camera (after adding the offset of the camera with respect to the center of the robot) to a global coordinate which is fed to the EKF for localization. Using inexpensive webcams, we are able to achieve localization accuracy and repeatability to approximately 2 cm and 0.02 radians (1σ) when the robot is 75 cm away from the barcode and 10 cm and 0.07 radians when the robot is 1.5 m away. Beyond this range the barcode becomes too small to detect reliably, but the nature of the localization is such that any detection will be helpful.

E. Obstacle avoidance

Once a path has been planned and the robot's pose determined, the path must be executed. For navigation we use a fairly standard artificial potential field approach. The waypoints in our system have been placed such that the robot should always be able to travel to its next waypoint in a straight line in the absence of unmapped obstacles, so the usual issues of static local minima are not particularly problematic. When people are present interfering with the robot's desired path, generally this type of reactive approach will work well, moving around the person or waiting until the path is clear. To detect obstacles, we use the Kinect as a planar range sensor (much like a standard laser rangefinder). We then use clustering techniques to turn the individual points into coherent obstacles that apply repulsive virtual forces on the robot while the robot's next waypoint provides an attractive virtual force.

We have made a few customizations to the general approach for better functionality within our application. First of all, since the Kinect has a small horizontal field of view, obstacles can easily "disappear" as the robot turns to avoid them. Thus, we maintain a set of obstacle objects that are used to generate forces — obstacles are added to the set when first encountered and removed when the robot is more than one meter away from them (one meter is also the current threshold beyond which an obstacle produces no force on the robot). We also have added state to the APF technique to determine if the current goal location is blocked: if the APF suggests a near-zero velocity for ten seconds, it reports that it has failed to reach a waypoint. If this was an intermediate waypoint, it will try to reach the next waypoint using the APF; otherwise, it will simply report failure back to the manager.

F. Monitor

Since the robots are designed to be an interactive part of the department, it is important that they display a public face during their travels. The `corobot_monitor` node takes the form of a Python GUI that runs at all times. ROS subscriptions are used to give the monitor access to relevant data, though we do not plan to show low-level data but rather higher level things like current position and goal. The monitor also communicates with the manager over ROS topics to request human interaction and reply as necessary. Future projects will add more features to this monitor such as video and the option of giving tours of the building when otherwise idle.

IV. Off-board software

The robots are designed to be autonomous entities within an overall team of corobots. However, their commands will come from user code that is developed and executed off board. Here we discuss the various important components that face the general users of the system.

```
def succ():
  print("Made it!")

def fail():
  print("Failed!")

def main():
  with Robot("corobot1.rit.edu") as r:
    f = r.nav_to("Office3651").then(succ, fail)
    # parallel computation can be done here
    # then wait for arrival/failure:
    f.wait()
```

Fig. 6. Example of callbacks using the Python API that allow asynchronous computation while the robot moves.

A. Client APIs

Since the overall goal of the project is to provide an easily-programmable team of robots for student use, the client APIs are built to enable users to incorporate robot commands with any other programming. The simple program of Fig. 1 shows some of the commands available in our Python API. We have also developed a Java API but here we describe the Python version as it has greater functionality at present.

By and large, the effect of the API is simply to open a network connection to a robot and convert function calls into messages sent (and received) over that connection. The interesting issue here is how to handle the time that the robot takes to execute certain actions. For example, the `nav_to` function takes a named waypoint and executes a plan through the building, an action which can easily take a couple of minutes. Should the student code block during that action? If the student code wants to execute other non-robotic actions during that time, then it should not block, but certain actions could be triggered by arrival at a destination as well. Rather than requiring knowledge of multi-threaded programs, we have provided an API that enables both types of operation seamlessly. In the Python API, each call into the robot class returns a `Future` object. Ignoring this object (that is, simply making a call like `robot.nav_to("Vending")`) will produce non-blocking behavior — execution will continue on the next line while the robot begins moving. On the other hand, if blocking behavior is desired, this can be accomplished simply by calling the `wait()` method on the returned Future object (i.e. `robot.nav_to("Vending").wait()`). Finally, the Future class also allows callbacks to be passed in using its `then()` method so that certain code can be executed upon completion. A small example of this is shown in Fig. 6.

The current API is quite limited so that we can better understand how the robot will behave under a limited set of circumstances. Users can navigate the robot to named locations or arbitrary (x,y) locations and request confirmation and robot status as in Fig. 1. We will extend the API gradually to enable more complex user interaction as specific projects demand.

B. Web-based access

For internal use and testing, the robots are accessed as shown in the various code examples by specifying an IP address to connect to. However, once the robots are used by the general student population, we need to provide a more seamless way of interaction. This is especially the case when we will have a team of robots and the user code will not necessarily use the same robot from one occasion to the next. We have developed a simple web server to mediate the interaction of the user and the corobots. It uses an HTTP-Request protocol to receive status updates from all active robots and display and maintain information about the overall system. Web sockets are used so that the user's connection to the web server is maintained over time and updates from the robots are seen live without having to refresh the page.

From the user's point of view, the workflow is such that they will log in and upload their code to the server. They can then request deployment of the code relative to a robot. The server then starts their code in its own process by giving it the IP address of a currently idle robot or putting the deployment request in a queue until a robot becomes available. This allows us to have control over where the code runs and force all interaction with the robots to go through the API discussed above. We are currently undertaking improvements to the web server that will increase user-friendliness in preparation for full deployment to students. In particular, the usability of the web site in terms of ease of uploading and deploying code, the logging of the results of user code, and more interactive display of robot status are all under development.

C. Robot Simulator

In order to speed up development time, we have also created a simple simulator that can interact with the API in much the same way that the robots do. While we are inspired in this idea by ROS's Gazebo and other platforms that allow interchangeable hardware and simulation, we do not aspire to a high level of fidelity. In fact, for the goals of the overall project we want to encourage people to work with hardware, so we have explicitly designed the simulator to be at best kinematically accurate. We do not simulate sensing, localization or obstacle avoidance, rather simply a robot that obeys the given commands by moving to the requested destinations and responding with the same asynchronous messages as the real hardware. Since the users of the API do not currently have access to sensor data, this does not present a significant limitation, and when it comes to real-world behavior, we have chosen to require real-world operation.

V. Discussion / Future Work

As of the time of writing, our robots are able to successfully navigate most of the third floor of Golisano Hall via the student API using code substantially similar to that shown in Fig. 1. The barcode localization and the calibrated odometry are sufficient to maintain pose information as long as the barcodes are sufficiently dense around the building (since we have put barcodes next to all doors to provide interactivity for humans, this is generally not an issue, though we have placed some additional barcodes around open spaces and a fairly empty hallway as well). Our current focus is to make the overall system more robust and user-friendly in preparation for student use in the fall of 2013. In particular, we look to improve the quality of real-time and post-hoc feedback through the use of better monitoring, a cleaner web site, and more streamlined logging facilities for user code. As we develop our code, we will continue to make all of the code, design documents and supporting information available online through Github (`http://www.github.com/corobotics`).

More importantly, as the main focus of our project, we are developing assignments for different classes that will use our robots so that students can develop programs that utilize the robots outside the specific context of a robotics course. For example, we plan to introduce an assignment in a second-semester course where students plan a path for the robot (effectively taking the place of the navigation by using the waypoint graph) and see the robot execute it in real life. An assignment in a database course will require the students to have robots collect information from around the building and build a distributed database of that information. Finally, we expect that the system will provide a base for students to pursue independent studies, MS projects and so on. A current independent study is focused on an Android app that uses the barcodes and server connections to display robot positions and interact with nearby robots, and future projects can develop a wide variety of new and exciting robotics applications.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Award No. IIS-1208566. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Karl Berger worked on early versions of the robot code. Kumar Chandan and Pavan Nutalpallati assisted with robot testing.

References

[1] S. Rosenthal, J. Biswas, and M. Veloso, "An effective personal mobile robot agent through symbiotic human-robot interaction," in *Proc. of Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.

[2] B. Coltin, M. Veloso, and R. Ventura, "Dynamic user task scheduling for mobile robots," in *Proceedings of the AAAI Workshop on Automated Action Planning for Autonomous Mobile Robots*, 2011.

[3] M. J. Matarić, N. Koenig, and D. Feil-Seifer, "Materials for enabling hands-on robotics and STEM education," in *AAAI Spring Symposium on Robots and Robot Venues: Resources for AI Education*, Palo Alto, CA, March 2007.

[4] T. Lauwers and I. Nourbakhsh, "Designing the finch: Creating a robot aligned to computer science concepts," in *Proceedings of AAAI*, 2010, pp. 1902–7.

[5] "Institute for Personal Robots in Education (IPRE)," http://wiki.roboteducation.org.

[6] I. Nourbakhsh, C. Kunz, and T. Willeke, "The mobot museum robot installations: A five year experiment," in *Proc. of the International Conference on Intelligent Robots and Systems*, 2003, pp. 3636–41.

[7] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *Proc. of IEEE Internation Conf. on Robotics and Automation (ICRA)*, May 2011, pp. 3400 –3407.

[8] G. Lin and X. Chen, "A robot indoor position and orientation method based on 2d barcode landmark," *Journal of Computers*, vol. 6, no. 6, pp. 1191–7, 2011.

[9] J. Brown, "Zbar bar code reader," http://zbar.sourceforge.net/.