# Experimental Results

## The Validation Task

In order to validate the basic functionality of the robot, a simple task was devised to simulate part of the task of retrieving an assembly kit from factory inventory. The task is designed to test the localization and navigation systems, the arm navigation system, and the object recognition and obstacle avoidance using the Kinect. The task is as follows

1. Beginning at the operator’s station, drive to a predetermined location near a table a few meters away. This validates the performance of the drive base hardware and the localization and base navigation systems.
2. Once at the table, locate the small manipulable box on the table. This validates the Kinect and the tabletop object recognition system.
3. Perform the pick() operation on the box to lift it from the table. This operation is described in [SECTION] above. This step validates the use of the Kinect to provide target poses for the arm, the arm and gripper hardware, and the box manipulation software.
4. Perform the place() operation to place it in the bin. This operation is described in [SECTION] above. This step provides further validation of the arm, gripper, and manipulation software.
5. Drive back to the operator with the box in the bin. This provides further validation of the drive base hardware and software.

This task is meant to simulate the act of driving through a factory to a shelf (represented by the table), picking up an item for a kit, and delivering it to the assembly station. In an actual factory environment, steps 1 through 4 would be performed repeatedly until every item in the kit was retrieved.

### Validation Results

## Mechanical and Electrical Systems

### Wheelchair Drivebase

(precision)

### Chassis Design

The frame design proved to be somewhat problematic in a few ways. First, although Bosch rail made quick prototyping possible on the robot, it also made it difficult to reliably fix some components of the robot in place, namely the arm. The arm is held to the vertical rails of the robot using T-slot nuts, and over the months that the robot was operated, it slipped down a few centimeters, requiring the robot model be updated to reflect the new position. Second, the placement of the arm itself is less than ideal for the task the robot must perform. The center of the arm’s work envelope is directly in front of the center of the point where the base is mounted to the robot. Since the base is mounted at about 1 meter off the ground, that puts the center of the work envelope only slightly above table height. Being able to reach under the table is useless, so much of the arm’s work space is unused. The arm should be raised higher if it is to be used for tabletop object manipulation. Last, the onboard storage bin’s location was chosen based on the available space within the reach of the arm, which was limited. A better location or a better bin design should be considered. Raising the arm may open up more options as to the location of the bin.

### The ABB Arm System

This system performed well, but has a few drawbacks. First, the IRC5 Compact controller dominates the robot volume, limiting the space available for other robot hardware. A smaller controller would have made the design of this system easier and resulted in a more elegant design. Second, the IRC5 Compact runs on 220 volt AC power. Finding a suitable inverter to create this from the 24 volt batteries was one of the more difficult parts of the system design. Finally, the RAPID programming environment provided by ABB is antiquated, limited, and constricting. Although it performs excellently as a robot controller, it does not expose the lower level control of the arm, such as joint torques, which would be useful for performing more advanced control techniques than position control. Fortunately, this robot’s tasks are achievable using only position control, which the ABB system does very well.

### Gripper

The gripper was one of the biggest problems with this robot. It was chosen from a limited selection of available grippers, and is not appropriate to this robot’s task. The gripper is a very simple design, with only an open and a closed position and no grasping ability. This limited the manipulation capability of the robot to simple boxes that were within the size range of the gripper. Although the gripper’s overall size is about right for the boxes that the robot must manipulate, the difference between its open size (6 cm) and its closed size (7.8 cm) is only 1.8 cm. This makes picking up objects difficult because the gripper must be very precisely placed. Unfortunately, this is difficult or impossible using the Kinect. A gripper with a longer throw between its open and closed positions would have been able to pick up a wider range of objects sizes and also pick up objects that are not precisely centered in the jaws. Finally, the gripping force of the gripper was only somewhat adjustable. The solenoid valve used to open and close it only works at above 210 kPa of pressure, which set the lower bound for gripping force at [FORCE]. This means that the robot cannot pick up delicate objects without crushing them. The corrugated cardboard boxes used in the validation task were only barely stiff enough to withstand this force.

### Battery Life

Three tests were performed to determine the life of the battery. In each test, the test process was run until the battery voltage reached 21.5 volts DC, which was considered the critical shutdown voltage. At this point, each cell in the lead acid battery has been depleted to about 1.8 volts, or 90% of its nominal voltage.

[IDLE VOLTAGE FIGURE]

In the first test, the robot’s systems are all turned on, but the actuators are disabled. In this idle test, the robot ran for 4 hours and 44 minutes before it hit the critical voltage.

[DT VOLTAGE FIGURE]

In the second test, the robot was strapped to a platform with rollers, and the drivetrain was exercised by commanding rotational velocities that followed a sawtooth profile, increasing from the maximum negative velocity to the maximum positive velocity, then resetting back to the maximum negative velocity. In this test, the robot ran for 3 hours and 13 minutes before it hit the critical voltage.

[ARM VOLTAGE VIGURE]

In the third test, the drivetrain was disabled but the arm was commanded to repeatedly execute trajectories between the stow position and the position to drop an object in the bin. At the end of each trajectory, the gripper was opened and closed. In this test, the robot ran for [RUNNING TIME].

From these tests, it appears likely that the robot could run in typical operation for about 3 [CHECK THIS] hours before its batteries are critically low. With adequate charging facilities and redundant robots or batteries, this should be adequate for a factory environment.

## The Kinect

### For Object Localization and Arm Planning

A major part of the validation task depended on the Kinect as a sensor for object segmentation and recognition. Unfortunately, the data from the Kinect proved to be unreliable.

### For Reading QR Codes

The Kinect’s RGB camera captures video at VGA resolution (640 by 480 pixels). Experiments with reading QR codes showed that a Kinect did not have sufficient resolution to read a tag, even when the tag filled the entire field of view of the Kinect. Unfortunately, this means that in order for the Kinect to be useful for recognition of tagged “smart” payloads, it must be paired with (and calibrated to) an external camera with a higher resolution sensor.

## Open Source Software

A major goal of this project was to use open source software, specifically software from the ROS community. The purpose of using open source software was to decrease development and testing time by leveraging the work of other researchers. In practice, this choice had mixed results.

The ROS core, consisting of the ROS graph node management and message passing systems, is stable and includes many tools that facilitate rapid robot development and management of complex robot software. However, much of the open source software developed for ROS is incomplete, limited, or inadequately tested. This section describes some of the problems encountered with open source software from the ROS community.

### API Stability

Much of the software created by the ROS community is still in progress, and so very few packages are guaranteed to be stable. Updates are frequent and usually provide improvements, but occasionally break existing functionality. This is especially true for distribution release updates, which frequently include API changes. ABBY is the first Case robot to use the Fuerte distribution of ROS, which was released in Spring 2012. At the time, it was the only ROS distribution to run on Ubuntu 12.04, which is the current Long Term Support release of Ubuntu. ROS Electric, the previous version of ROS, does not support Ubuntu 12.04. In order to port the existing CWRU ROS package to Fuerte, several changes had to be made. When the ROS Groovy distribution was released in Fall 2012, so many APIs were changed, particularly in the arm navigation stack, that the decision was made not to upgrade.

Another example of difficulties with unstable or unfinished ROS APIs is ROS Serial, a package that allows development of ROS nodes on microcontrollers. This package exposed an (undocumented) API for creating ROS services on microcontrollers, but the code was non-functional and clearly untested. As a result, ABBY uses a custom-written fork of ROS Serial that properly supports ROS services. The Groovy release of ROS serial includes and complete rewrite of the code in question, but this was not discovered until after the changes were made, and the rewrite was not backported to fuerte.

### Documentation

ROS documentation is a mix of a wiki system, a question and answer forum, and autogenerated documentation for the code. Unfortunately, because the documentation is in the form of a wiki, it is often incomplete and out of date. The autogenerated documentation is up to date, but not often very helpful because the code itself is not documented. As a result, much of the autogenerated documentation is little more than a list of available methods and a link to the source code. The lack of documentation means that reading and understanding the source code is a must before using most ROS packages.

### Reusability of PR2 Software

The Willow Garage PR2 is a mobile manipulator with two arms and an array of sensors that is commonly used in robotics research. Because it is a common platform, and because it is the flagship product of Willow Garage, the maintainer of ROS, there is a lot of software available for the PR2. It was hoped that much of this software could be reused or adapted for ABBY, as it would open up a large library of abilities for ABBY. However, much of the software written for the PR2 is not written in such a way to be easily ported to other robots, and the documentation for PR2 software is often limited to instructions for running it on a PR2 or simulated PR2, meaning that even understanding what the software is doing and how to interface with it requires exploring the source code. As a result, the only PR2-specific software running on ABBY are the tabletop object segmentation node and the object bounding box server, which are a very small part of a tabletop object manipulation pipeline written for the PR2. The rest of ABBY’s software is either from more general ROS stacks or custom-written.

### Safety and Reliability

There are many standards and standard practices for writing software for industrial machines. These standards were created to ensure that the software runs safely and reliably, is robust, and fails gracefully and safely. ROS does not conform to these standards.

One example of ROS instability is the driver for the Kinect, which for several months had a bug that caused it to crash seemingly at random. Once the driver had crashed, it could only be restarted by killing the nonresponsive process and restarting it, sometimes several times. While this was simply a nuisance on a research robot, it would make the Kinect unusable with ROS in a production environment. The bug was eventually fixed, but the fact that it existed, unfixed, for so long is an example of how a lot of ROS code, even something as popular as the Kinect driver, is not yet ready for general use.

Other program instability was apparently in more concerning areas. The arm navigation stack includes several trajectory filters. Occasionally, one of the trajectory filters will fail. In this case, the safe behavior would be for it to emit an error and stop processing the trajectory or skip the filter. Instead, a malformed trajectory is passed to the next step in planning, which causes the entire arm controller to crash. This is not only wrong but potentially dangerous. Software running on a robot or industrial machine should never crash under any circumstances.

In fact, most ROS software will crash easily due to memory allocation errors when receiving a memory allocation error. This behavior was observed in the inverse kinematic solver and in other ROS software. There is no ROS standard for checking whether memory has been allocated before accessing it, even for data structures from “untrusted” sources, such as messages received from other nodes. In addition to allowing for sloppy code to crash parts of the robot software, which is dangerous in and of itself, this exposes a large security hole in ROS. It is possible to bring down or otherwise compromise a robot simply by sending it a malformed message. Since message senders are not validated and messages are not encrypted in any way, an attacker could do this from anywhere on the robot’s subnet, provided the robot’s ROS master IP and port were known.

### Conclusions on ROS and Open Source Software

Although solving problems with open source software ended up taking up the majority of the time on this project, the ROS core enabled rapid development of the basic systems of the robot. In addition, ROS tools such as Rviz, rxconsole, and roslaunch enabled easy management and troubleshooting of the system. Independent development of similar GUI tools would have been a massive undertaking, beyond the scope of this project. Given the experiences described in this section, the best approach to open source software and ROS is carefully choose which software packages to use and which to develop from scratch. If a mature, well-tested package exists for a specific task, it should be used. The ROS core, for instance, is well-tested and reliable, and it fulfills a specific purpose, i.e. management of a multiprocess robotic software system. On the other hand, the practice of adapting existing software for unintended purposes is not recommended, i.e. reusing code written for other robots or tasks, such as PR2 code. This also extends to software that has not been thoroughly tested. The ROS arm navigation stack is widely used, and there are many tutorials for using it with custom robots, but is listed on the ROS wiki as unstable. Although this did not cause concern, several bugs were encountered with the arm navigation code, as described above. Most of ROS is similarly a work in progress, and should be treated as such, even the components that have been “released.”