# Experimental Results

## The Validation Task

### Validation Results

## Mechanical and Electrical Systems

### Wheelchair Drivebase

(precision)

### Frame Design

(arm placement)

### Gripper

(size, dexterity, force)

### Battery Life

## 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.

### ROS Verdict

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.”