|  |
| --- |
| Department of Electrical Engineering and Computer Science  Case Western Reserve University |
| A Low-Cost Mobile Manipulator for Industrial and Research Applications |
| Submitted in partial fulfillment of the requirements for the degree of Master of Engineering |
|  |
| Edward Venator |
| [Pick the date] |

|  |
| --- |
| This thesis describes the creation of a mobile robot equipped with an industrial robotic manipulator. The resulting mobile manipulator incorporates a suite of commercially-available sensors and processing hardware to enable the robot to operate as an intelligent agent alongside humans. Ultimately, this robot will be employed in research on autonomous kitting in industrial environments. |

Introduction 5

Industrial Mobile Manipulation 5

ABBY - System Design 6

Invacare Ranger Wheelchair Base 6

ABB IRB-120 Robotic Arm 6

End Effector 7

Custom Frame Design 9

Power 10

Sensors 12

Odometry 12

Yaw Rate Sensor 12

Microsoft Kinect 13

Sick LMS-291 13

Computing Hardware 13

PC 13

National Instruments cRIO 14

ABB IRC5 15

ROS Framework 16

The Robot Model 17

Hardware Drivers 19

The Mobile Base 19

ROS Industrial 20

Gripper Driver 21

Experimental Software 22

Kinect Calibration 22

Mobile Base Planning 22

Localization 22

Mobile Base Trajectory Planning 29

IK solver 33

KDL Solver 33

IKFast 34

Arm Navigation 34

Collision Detection 35

Kinect Data Filtering 35

Tabletop Box Manipulation 36

The Manipulation Controller 36

Box Manipulation 37

QR Code Recognition and 3D Localization 38

Industrial Safety 39

Reflexive Speed Limiting 40

Reflexive Halt Methods for Mobile Bases 40

Reflexive Halting for Manipulators 41

Emergency Stop System 43

E-Stop Systems Used in This Lab 43

E-Stop Requirements for This Robot 44

Version 1 Prototype 45

Version 2 Design 46

Experimental Results 47

Applications and Future Work 48

Conclusion 48

# Introduction

# Industrial Mobile Manipulation

\* Literature Review

# ABBY - System Design

Abby’s design was dictated by several factors. The primary factor in the design was reduction of cost, which was achieved by using materials and components already available in Case Western Reserve University’s Mobile Robotics Lab.

[Annotated rendering]

## Invacare Ranger Wheelchair Base

The Invacare Ranger is a wheelchair chassis in Invacare’s Storm series. The wheelchair base has a differential drive system with two pneumatic drive wheels in the back and two solid caster wheels in the front. The drive wheels are each powered by a 24 volt DC motor geared for a maximum speed of 5 miles per hour (2.24 m/sec). Because of the configuration of the robot’s wheels, it can spin on its own axis and drive forward and backward. It cannot move sideways.

The wheelchair base is prone to wheel slip when commanded to accelerate or decelerate quickly. The acceleration limits of the robot were characterized by testing a series of constant linear acceleration commands. These tests were performed with the robot’s arm in the stowed position on a smooth tile floor. From these tests, the maximum achievable forward acceleration (with no slip) was determined to be RESULTS. The same test was performed using constant rotational accelerations. From these tests, the maximum rotational acceleration was determined to be [RESULTS].

A Sabertooth 2x50 dual brushed DC motor controller controls the speed of the motors in the mobile base. The Sabertooth 2x50 is an H-bridge PWM motor controller that supplies a variable DC voltage from -24 volts to +24 volts to each motor based on commands it receives over a serial data connection. The Sabertooth 2x50 is powered by a 24 volt DC rail that is energized and de-energized by the emergency stop circuit described in [SECTION].

## ABB IRB-120 Robotic Arm

The manipulator on the robot is an ABB IRB-120 industrial robotic arm. The IRB-120 is a six-axis robotic arm with a spherical wrist. It has a tool flange that allows for the mounting of end effectors as well as pneumatic and electrical connections near the tool flange to connect sensors and actuators to the arm. The IRB-120 is ABB’s smallest robotic arm, with a 580 mm reach and a payload capacity of 3 kg. The arm itself weighs 25 kg and must be mounted to the extreme front of the robot, which means its weight exerts a large moment on the robot. This was a serious consideration in the placement of the robot’s center of mass. It can be mounted at any angle, and on this robot is mounted 90° (with the base mounted to a vertical surface). The arm is mounted vertically on the front of the robot so that the majority of the arm’s work envelope is outside of the volume of the robot. This maximizes the functional work envelope of the arm and minimizes the possibility of the arm colliding with other parts of the robot.

The IRB-120’s joints are powered by non-back-drivable AC electric servos, with position feedback from resolvers. According to ABB, the IRB-120 is capable of position repeatability of 10 micrometers. The arm’s position is controlled by an ABB IRC5 Compact robot controller, which is in turn commanded by a ROS Industrial interface. The details of this control structure are described later in this document.

## End Effector

There are many types of grippers and graspers used with industrial robots. Some are purpose-built fixtures for holding specific parts. Some grippers use suction to be able to quickly pick up light objects such as electronic components. Still other graspers use dexterous fingers to be able to securely pick up and manipulate objects of different shapes and sizes. This robot uses one of the simplest gripper types, a two-position parallel plate gripper.

Since this robot has to be able to manipulate part boxes of many sizes, a more dexterous gripper would have been desirable, but one of the goals of the project was to create the robot as cheaply as possible. A dexterous grasper like the BarrettHand costs about $30k, which would nearly double the cost of this robot. The pneumatically-actuated parallel plate gripper has only two positions (open and closed), and is simply and cheaply constructed from aluminum and a single double-throw pneumatic piston. When open, the gap between the jaws is 65 millimeters, and when closed the gap is 46 millimeters.

The gripper is pneumatically actuated using stored air from accumulator tanks that are kept at 825 kPa by an onboard 12 volt DC compressor. The compressor is turned on and off by an Innovation First Spike relay, which is controlled by digital pressure switch calibrated to close at 690 kPa and open at 825 kPa. This control circuit can be seen in the power distribution diagram in FIGURE. This 825 kPa stored air is regulated down to 275 kPa working pressure and used to actuate the gripper. The pneumatic piston in the gripper is controlled by a pneumatic solenoid valve, a magnetically actuated valve with one pressure inlet and two pressure outlets. The inlet is connected to the output of the regulator and the outlets are connected to each of the inlets on the gripper’s pneumatic piston so that applying pressure through one outlet opens the gripper and applying pressure through the other outlet closes the gripper. The solenoid valve is designed to that when one outlet is connected to the pneumatic pressure inlet, the other is vented to the atmosphere. The valve is actuated by running current through one of two solenoid coils. This current is supplied by a custom circuit based on an Arduino microprocessor development kit, as shown in [FIGURE].

[GRIPPER CONTROL SCHEMATIC]

Although the gripper has only two positions, the pneumatic nature of the system makes the gripper jaws back-drivable, with a constant gripping force of [NEWTONS], as calculated in equation [EQUATION NUMBER] with the pneumatic system’s adjustable regulator set to (275 kPa). The regulator can be set to any pressure up to the system’s maximum pressure of 825 kPa. The regulator setting was chosen so that the gripping force would be great enough to ensure a strong grasp on manipulated objects without being so great as to damage them.

[PNEUMATIC FORCE EQUATION]

## Custom Frame Design

Coupling together the ABB IRB-120 robotic arm and the Invacare Ranger wheelchair base is the main frame of the robot. The structural elements of the frame are made from Bosch Rexroth aluminum profile struts. This Bosch rail was chosen because it was readily available in the lab, but it has several features that make it a good choice for a prototype robot. Bosch rail is an extruded aluminum product with T-slots running the length of the rail. Because T-slots do not require holes to be drilled in the rail for mounting, they allow flexibility in adjusting mounting positions on the prototype robot. Because Bosch rail is aluminum, it is easy to machine, but strong and relatively light.

The design of the frame itself was motivated by the need to hold the IRC5 Compact robot controller and the assorted power and control electronics of the robot. The IRC5 is large (480mm x 580mm x 258mm) and heavy (28.5kg), and it dominates the robot frame. Previous experience with Invacare Storm Series chassis showed that they were prone to tipping unless the center of mass was carefully placed, so robot frame was meticulously designed in 3D CAD software to place the center of mass as close to the center of the robot volume as possible. The mass of every component of the robot was entered into the CAD models, and components were placed so as to keep the center of mass low as well as relatively centered between the front and rear wheels. The final center of mass, as determined by the CAD model, is [CG COORDS].

In addition to the Bosch rail structural elements, the frame includes four panels for mounting the robot’s electronic and pneumatic components. It was important to protect the onboard electronics from damage in the case of a collision, so the majority of the electronics are mounted to a polycarbonate panel underneath the IRC5, where they are completely enclosed inside the robot. The advantage of this choice is that these electronics are safe from collisions and the mass of the heavy power electronics is kept low to the ground. The disadvantage is that the electronics are difficult to service. In order to access these electronics for service, one must remove the robot’s batteries from the wheelchair base and access them from the underside of the robot. Space considerations also made it difficult to fit all of the electronics on this panel, so it is difficult to remove some components for service. Although this design is advantageous in terms of keeping the robot’s overall volume small and the robot’s center of mass low, it is not user-friendly in the event that the robot requires service. Fortunately, now that the robot is complete, it has proven very reliable and rarely requires service. The top panel of the robot, also made of polycarbonate, holds the pneumatic system and the PC. These were mounted on the top panel in anticipation that they would require more user access and to put the pneumatics close to the arm. Two front panels, made of aluminum sheet, hold the main power distribution rail and the power supply for the LIDAR. The power distribution rail is mounted on a front panel to make it easily accessible, and the LIDAR power supply is mounted on a front panel to place it close to the LIDAR, which is mounted to the front frame rail.

On the front of the frame is a vertical mast made of Bosch rail. This mast serves several purposes. First and foremost, it provides a mounting point for the IRB-120 robotic arm. The rails are spaced so that the arm’s four mounting holes line up with the two rails, and the arm can be fixed to any position along the height of the rail by tightening the T nuts that hold it in place. This allows the robot to be reconfigured for different tasks that may require the arm to be mounted at different heights. In addition to holding the arm, the mast provides a high vantage point for a Kinect camera and allows the WiFi router to be mounted far away from possible interference from other electronics.

## Power

[POWER DISTRIBUTION DIAGRAM]

All of the robot’s power is distributed using DIN rail power distribution blocks. These blocks are modular, insulated, and compact. The robot has two DC voltage buses (24 volt DC, and 13.8 volt DC) and a single ground block. Although previous robots in this lab had a 5 volt DC bus, it was minimally used, and for this robot all circuitry requiring a 5 volt supply is powered from the 13.8 volt bus using dedicated regulators. In addition to these main voltage buses, several parts of the robot have their own power regulators and supplies.

The robot’s main voltage rail is a 24 volt DC bus supplied by two 12 volt batteries in series. This 24 volt bus is required by the Invacare wheelchair base’s drive system, and the Invacare wheelchair base includes the batteries that supply the bus. The batteries are connected in series with a 120 amp main resettable circuit breaker, which also serves as the main power switch for the robot. In addition to the robot’s drivetrain, the robot’s PC, LIDAR, and the National Instruments cRIO are all powered directly from the 24 volt DC bus.

Because the cRIO is a critical component of the drivetrain and the inductive kick of the motors can cause significant noise on the 24 volt DC bus, a peak-detector circuit is used to protect the cRIO from voltage droop on the 24 volt rail.

In addition to the two DC buses on the robot, there is an AC inverter, which is used to power the ABB IRC5 Compact robot controller. The IRC5 is powered by single-phase 220 volt AC at 50 Hz. The inverter on the robot is capable of delivering up to 2 kW of power continuously and surges of up to 3kW, which is necessary to account for the high current draw when the controller first enables the motor drive. The inverter is powered from the 24 volt DC bus and is only used to power the IRC5 Compact and (through the IRC5) the IRB-120 robotic arm.

Much of the electronics on the robot require a lower voltage bus to operate, nominally 12 volts DC. These electronics are powered from a 13.8 volt DC bus. The 13.8 volt bus is powered by a (13.8 VREG DETAILS), which is powered from the main 24 volt bus. This bus powers the WiFi router, emergency stop circuitry, the cRIO interface board, the Kinect camera, and the pneumatic compressor.

Because the compressor draws a large amount of current, it caused the 13.8 volt regulator's output to droop to about 5 volts for approximately 450 ms (see [FIGURE]) when it switched on. This droop was sufficient to cause the onboard Ethernet router to reboot, interrupting communications between the computers onboard. In order to fix this problem, an LC filter was added to the 13.8 volt power rail. A 10mF ([CONFIRM]) capacitor acts as a charge reservoir for the electronics on the 13.8 volt power rail (including the router), and a 55 μH inductor acts as a current choke to limit the instantaneous current draw when the compressor turns on. Figure ([FIGURE]) shows that this filter kept the 13.8 volt rail from dropping below 10 volts, and it recovers to its nominal voltage in under 100 ms. This droop is not enough to cause the router to reboot.

[LC BEFORE AND AFTER TRACES]

[POWER MONITORING HARDWARE]

[BATTERY DISCHARGE INFO]

## Sensors

### Odometry

In order to sense the motor speed, there is a Grayhill [GRAYHILL MODEL NUMBER] encoder on each motor’s output shaft. The encoder outputs quadrature pulses the frequency proportional to the motor speed. These motor shaft encoders have very high resolution output (256 pulses per revolution), but cannot provide accurate wheel position information for odometry because of backlash in the gearboxes. For odometry, there is an encoder attached to each wheel by a toothed belt. The wheel encoders spin fifteen times more slowly than the motor encoders, but still provide a very high resolution output. [INFORMATION ABOUT RESOLUTION]. The output of the wheel encoders is differentiated to get the wheel velocities, which are then fed as control inputs into a Kalman filter that outputs a robot pose estimate consisting of X and Y coordinates and a heading.

### Yaw Rate Sensor

Odometry is prone to errors due to wheel slip, discretization, and linearization errors. Although it can be accurate over short distances, errors accumulate, particularly when the robot turns. In order to help increase the accuracy of the robot’s pose estimate, the robot has an Analog Devices MEMS gyroscopic yaw rate sensor. The yaw rate sensor is capable of measuring rotation rate about the robot’s yaw axis at up to 2.6 radians/second, with an onboard temperature sensor, which is used for automatic bias correction. Without automatic bias correction, the yaw rate sensor will have a non-zero output when the robot is stationary, and this “drift” will vary with temperature. The bias-corrected output of the yaw rate sensor is combined with the odometry in the pose Kalman filter to provide a more accurate estimate of yaw rate. The more accurate yaw rate in turn allows the Kalman filter to output a more accurate heading estimate than would be possible using the odometry or gyroscope alone.

### Microsoft Kinect

The Microsoft Kinect is an RGBD (Red, Green, Blue, Depth) camera marketed as a gaming controller. ABBY has a Kinect camera mounted high on the front mast with a custom-designed acrylic bracket that fixes is at a down-ward looking angle of 51 degrees from the horizontal. At this angle, the Kinect is looking down into the IRB-120's work envelope and capable of viewing the floor in front of the robot. The RGBD data from the Kinect are converted into three-dimensional point clouds, which are used to detect obstacles and manipulable objects.

### Sick LMS-291

An LMS-291 laser ranging sensor (LIDAR) from SICK AG is mounted horizontally to the front of the robot. The LIDAR provides planar range scans of a 180 degree arc in front of the robot. These range data are used for localization and obstacle detection.

## Computing Hardware

The robot has three main computing devices on board, connected by a local Ethernet network with an onboard WiFi access point so operators can wireless connect to the robot for maintenance and control.

### PC

The majority of the robot’s processing is performed on a Linux PC. This PC runs all of the perception and higher level planning algorithms, which do not require a real-time operating system. In addition, the PC is responsible for processing LIDAR and Kinect data directly from the sensors. These tasks are computationally intensive, particularly the perception task, which involves performing object recognition on point clouds from the Kinect.

The computer was designed so as to balance cost, physical size, and processing power. The computer’s motherboard is an ASUS micro-ATX motherboard, which was chosen over the smaller mini-ITX form factor because many mini-ITX boards were found to have poor thermal management during the construction of Otto. The case chosen was the smallest micro-ATX case available at the time from major computer vendors, measuring 13.00" x 3.80" x 15.40". The case came equipped with a compact AC power supply, but this was replaced with a 24 volt DC power supply so that the robot would not need an inverter to supply 115 volts AC to the PC power supply. The computer case and power supply combined cost $155, with the majority of the cost ($90) going toward the DC power supply.

The PC’s computing hardware is fairly moderate and represents a balance between cost and computing power. The processor is an Intel i5 2500k, a four-core processor utilizing Intel’s Sandy Bridge architecture clocked at 3.2 GHz. The PC also has 8 gigabytes of DDR3 RAM and a solid state hard drive. The motherboard, processor, and RAM were purchased specifically for this robot at a cost of $342. The solid state drive was recycled from a previous robot computer, but would have cost on the order of $50.. Combined, the total cost of the PC for the robot was $497 plus the cost of the hard drive. This cost is consistent with the goal of producing a low-cost mobile manipulation platform, and would continue to drop as computer processers become cheaper and more powerful.

### National Instruments cRIO

Some tasks pertaining to sensor interfacing and motor control require real-time processing, analog to digital conversion, and robust digital I/O. These tasks are beyond the reach of commercially-available PC hardware. The cRIO 9072 from National Instruments combines a 266 MHz PowerPC processor with a 1M gate Xilinx FPGA. The PowerPC processor is running the vxWorks real-time operating system and the Xilinix FPGA is connected to the PowerPC processor and to 8 reconfigurable IO slots. These reconfigurable IO slots accept a myriad of modules sold by National Instruments ranging from analog to digital converters to serial bus interfaces. Abby’s cRIO is equipped with three IO modules. A digital input/output module is used to read values from the wheel encoders and to output the enable signal to the emergency stop. A high speed digital input/output module is used to read values from the motor encoders and to send serial packets to the Sabertooth motor controller. An analog input module is used to monitor the voltage rails and read values from the yaw rate sensor.

The FPGA is used to perform minimal signal processing on the inputs and outputs, including counting encoder ticks and forming packets to command motor speeds. Besides this signal conditioning, the only processing performed on the FPGA is the PID controller that determines the motor speeds. Because PID control is dependent on very fast loop closure (10 ms) and is sensitive to the lag that can occur even in a real-time operating system, it is implemented on the FPGA. In addition to this minimal processing, the FPGA acts as a bridge between the IO connections and the cRIO’s PowerPC processor.

In addition to the FPGA, the robot uses the cRIO’s PowerPC processor for low-level processing related to the operation and control of the drive base. The robot’s physical state observer (PSO) takes in the current encoder counts and yaw rate sensor measurements from the FGPA and uses a Kalman filter to generate an estimate of the robot’s current position. The PSO used on this robot is described in detail in [PERKO]. In addition to this processing, the PowerPC operating system passes raw values from the FPGA to the robot’s PC over the robot’s local Ethernet network and receives speed commands from the robot’s PC that it then passes to the PID controller on the FPGA.

### ABB IRC5

The ABB IRB-120 robotic arm can only be controlled by ABB’s IRC5 Compact robot controller. This controller contains all of the processing hardware and power electronics to control the arm. It runs a custom real-time operating system that can only be programmed in ABB’s proprietary RAPID programming language. Although the IRC5 has built-in software to perform inverse kinematics and path planning, it is very finicky about avoiding singularities, and the preferred method of programming it is to “teach” it by manually moving the robot to points. Although this method is useful in industrial environments where the robot executes a predefined path, it is not possible with a dynamic planner. Because of these limitations of the RAPID programming language and operating system, we limited the software running on the controller to the bare minimum to interface with the IRB-120 arm. There are two TCP servers running on the controller. One publishes the current state of the arm, including joint states and stop conditions, and the other receives joint trajectories as a stream of joint angles. Each point in the trajectory contains six angles, which fully specifies the position of the robot. The only processing that the IRC5 performs is interpolation between the points in the trajectory, which is accomplished with the built-in functions of the RAPID programming language.

In addition to the real-time RAPID operating system, there is a second computer connected to the IRC5 Compact cabinet, the FlexPendant. The FlexPendant is a handheld touchscreen computer running a custom software package under Windows CE. On this robot, the FlexPendant is used only by operators as a monitor for the IRC5 status. It is possible to run the robot “headless” with the FlexPendant disconnected.

## ROS Framework

The robot’s software runs within Robot Operating System (ROS). ROS is a framework for research robotics development that encapsulates algorithms as nodes, which pass information to each other through sockets as messages. The use of modular nodes makes it easy to add functionality to the robot without adding complexity. Standardization of messages within ROS makes it easy to swap nodes for other nodes that perform similar functions. ROS also has a vast library of existing nodes and algorithms, allowing researchers to leverage prior work without having to reimplement algorithms.

ROS nodes communicate to each other by sending messages to each other on topics. Messages have predefined types that define the fields of the message. Many message types are already defined in the ROS core and in existing ROS packages, but developers can also define their own message types. Topics are identified by names, which are organized into hierarchical namespaces. ROS nodes can publish messages to one or more topics for other nodes to subscribe to. Many ROS nodes can publish to a single topic, provided that all the message types are consistent, and many ROS nodes can subscribe to a topic. ROS topic communication is distributed, meaning that nodes communicate directly from the publisher to the subscriber, and the ROS master node only facilitates this communication by maintaining a list published topics and negotiating the direct connections between nodes.[1]

In addition to one-way communication through ROS topics, ROS nodes can provide services to one another. A service is defined by a request message and a response message. A ROS node providing a service advertises it by its name in a hierarchical namespace, similar to the topic naming system. A service client sends a request message to the service server containing parameters or data to be processed. The service server performs the service requested and sends a reply message; the reply message containing processed data or a status message about the service.[1]

ROS also provides and action server interface. Like ROS services, ROS actions are based on a server-client model. Whereas services are synchronous—the client blocks until it receives a reply—actions are asynchronous, making them more appropriate for requests that make take a long time, such as moving an actuator or querying a sensor. Actions consist of three messages. The client sends a goal message to the action server. The server acknowledges the goal and begins processing it. Optionally, the server may publish feedback messages while it is processing the goal. When the server is finished processing the goal, it sends a result message, which notifies the client that it has finished processing the goal and returns the result of the process. [2]

### The Robot Model

Another feature of ROS is the definition of robot physical characteristics for visualization and simulation using Universal Robot Descriptor Files (URDF). URDFs incorporate kinematic information such as joint geometry and inertial properties, collision information defined by geometric primitives or meshes, and visualization rendering information, also defined by geometric primitives or meshes. URDFs are a dialect of XML, with tags defined for robot links and joints. Various ROS nodes use the data parsed from URDFs for tasks such as kinematics and frame transforms, collision detection, and visualization in the Rviz GUI application. ABBY is fully defined in a modular URDF file generated using the ROS xacro system of xml generation macros.

The robot frame is defined in the URDF as a series of links joined by fixed joints. The visual, collision, and inertial data for each of the links was imported from a 3d model of the robot created in Autodesk Inventor, a 3D CAD package. The only movable joints on the main robot frame are the wheels and casters, which can spin freely. When the robot is running, a static joint state publisher publishes a constant angle of 0 radians to all of these joints, but these joints could be used in the future for tasks such as modeling the current position of the casters if the necessary sensors (joint position encoders on the casters) are added.

The Kinect and the SICK LIDAR are defined as xacro macros that place their visualization and collisions meshes in the URDF and define all necessary sensor frames as mass-less links. This makes it easy to reuse the sensors in models of this and other robots by simply importing the xacro macros. The macros each define a sensor root link at an externally visible point on the sensor body. This is much easier than the previous method of manually publishing a transform from the robot root link to the sensor frame because the sensor frame is located inside the body of the sensor and consequently difficult to locate on the physical robot.

The IRB-120 arm is also defined as a xacro macro. Each of the 7 links is defined by a visualization/collision mesh created from solid models of the arm obtained from ABB. These links are joined by six joints, which are defined according to the joint dimensions and rotation limits provided in ABB documentation. This definition of the arm is used by the forward and inverse kinematics solvers to convert joint angles into Cartesian coordinates and vice versa. It was also used to generate the arm navigation package that performs trajectory planning for the arm.

[rviz screenshot with model]

In addition to providing the geometric definition of the robot, the robot model makes the Rviz robot visualization GUI much more usable. Because visualization meshes of the robot are defined, Rviz will render an accurate visualization of the robot in its current state. This is useful for verifying that the state of the robot in ROS matches the physical state of the robot. In an industrial environment, it would also allow a user to remotely monitor a robot without the need for a CCTV system external to the robot.

## Hardware Drivers

In order for ROS to read data from a sensor or send commands to an actuator, it must use a software driver implemented as a ROS node. The driver node for a sensor interfaces with the sensor hardware and publishes data as ROS messages to the appropriate ROS topic(s). The driver node for an actuator subscribes to actuator commands on the appropriate ROS topic and interfaces with the actuator hardware to execute the commands.

ABBY's Kinect camera and SICK LIDAR use preexisting open source drivers. The ROS driver node for the mobile base was developed previously by our lab for other robots using the same hardware, and required limited modification for this robot.[3] The driver for the ABB robotic arm was written for this project in collaboration with the Southwest Research Institute (SWRI) of San Antonio, Texas. Since the gripper is a custom device, it uses custom driver software.

### The Mobile Base

The mobile base is controlled by software running on the cRIO, as described above. The cRIO sends data to the PC containing information about the robot's pose, the state of the power supplies, and raw count data from the encoders. The PC sends angular and forward velocity commands to the cRIO and may send commands to the cRIO to activate or deactivate the emergency stop or to reboot the cRIO. These two tasks (sending and receiving data) are handled by two different ROS nodes. A third ROS node publishes pose information as a standard ROS message.

The receiving ROS node handles UDP packets from the cRIO. Encoder data is checked to ensure that all of the encoders are updating properly, and voltage data is checked to monitor the battery level and health of the power regulator. The results of these checks are fed into a ROS diagnostic updater, which can be used for operator feedback. Voltage information is also published to a custom ROS message so that other nodes on the robot can subscribe to the voltage data. Pose information is published as a custom ROS message type and sent to the odometry translator node. The odometry translator publishes the robot's pose using ROS-standard odometry messages, which are used in ROS's planning and localization packages.

The sending ROS node subscribes to ROS "twist" topics containing desired rotational and translational velocity from the mobile base trajectory planner and sends the commands to the cRIO as UDP packets. It also provides a ROS service to reboot the cRIO and ROS services to enable and disable the drive base motors with the emergency stop.

### ROS Industrial

ROS Industrial is a project led by SWRI to develop a standard ROS framework for using ROS with industrial robots.[4] ABBY uses the ROS Industrial framework of messages and driver nodes to control the IRB-120 using the IRC5 Compact. ABBY's ROS Industrial driver was written specifically for this project, but was later incorporated into the ROS Industrial codebase.

The robotic arm driver, like the mobile base driver, consists of two ROS nodes that communicate with a server running on the IRC5 robot controller. ROS trajectory messages describe the trajectory of a robotic arm as a series of points, with each point describing the position and velocity of all of the robot's joints. One of the ROS nodes subscribes to ROS trajectory messages, breaks them up into packets, and sends them to the IRC5 controller over TCP using a standard packet structure defined by SWRI. The other ROS node connects to the IRC5 controller over TCP and listens for state information from the controller, which is sent using another packet structure defined by SWRI. It publishes this state information, consisting of all of the robot's joint angles, as ROS joint state messages and ROS joint trajectory feedback messages. These messages are used by other ROS nodes to determine the position of the robot's arm and as feedback to the arm planning nodes. TCP was used because it is the only non-proprietary network protocol supported by the IRC5 Compact's RAPID system. [5]

The software on the IRC5 Compact is written in RAPID, ABB's proprietary programming language. The software running on the IRC5 Compact consists of a trajectory server, a state server, and a motion process. The state server periodically polls the positions of the joints in the arm and sends that information to the ROS system. The trajectory server receives trajectory packets from the ROS system and queues them for the motion process. When a complete trajectory is received, the motion process commands the arm to go to each point in the trajectory. By default, the IRC5 controller attempts to stop precisely at each point, resulting in jerky robot motion. This problem is solved by defining all of the intermediate points in the trajectory as being low precision waypoints, and only requesting a precision stop at the last point in the trajectory. Because RAPID only has fixed-length data structures, trajectories must have a fixed maximum length. Paths generated for the IRB-120 with the current planner software have been experimentally determined not to exceed 250 points, so the maximum trajectory length was set to 250.

### Gripper Driver

The gripper driver is a ROS node that runs natively on an Arduino's AtMega 328 microcontroller using the ROS Serial framework. It sends and receives ROS messages over the USB serial connection. A ROS node, included in the ROS Serial package, runs on the PC and acts as a transparent bridge between the ROS system and the ROS node on the microcontroller. The ROS node on the microcontroller publishes joint state messages describing the current position of the gripper plates and provides a ROS service to open and close the gripper. The Fuerte version of ROS Serial did not actually support ROS services.[6] In order to implement a service on the Arduino, changes were made to the ROS Serial bridge node and microcontroller code to enable support of services. [7]

# Experimental Software

## Kinect Position Calibration

Because the position of the Kinect camera's optical frames within its housing are not precisely known, it was necessary to perform a simple extrinsic calibration routine to determine the position and yaw angle of this frame.

To determine the height and yaw angle, the robot was placed so the Kinect had an unobstructed view of the floor in front of it. RANSAC was used to fit a plane to the floor, and the transform from the fitted plane to the Kinect was calculated. A similar method was used to determine the x position of the Kinect. The robot was placed to that the Kinect had an unobstructed view of a wall at a known distance in front of it, and RANSAC was used to fit a plane to the wall. The distance of the wall plane was then used to calculate the position of the Kinect. The y position of the camera was estimated by visually lining up the Kinect point cloud with a known reference, in this case the tip of the gripper jaw, which had a known position determined by forward kinematics.

With the position of the Kinect now known, it was necessary to precisely determine the pose of the arm relative to the Kinect. Due to the length of the arm, a small error in the orientation of the arm base pose could cause significant disparity between coordinates in the Kinect's frame of reference compared to the position of the robot's gripper calculated using forward kinematics. The pose of the arm base was originally taken from the CAD models used to design the robot, but small errors in construction make this somewhat inaccurate. The proper transform was calculated using a Mathematica program. Given a set of data points, each one containing the position of the tip of the gripper with respect to the Kinect, measured from a manually-selected point in the Kinect point cloud, and the pose of the gripper with respect to the base link of the arm, calculated by ROS's TF server using forward kinematics, the program calculates the transform (translation and rotation) between the Kinect and the arm base link that minimizes RMS error in the gripper tip position. This program was run using an input consisting of eighteen data points, and the resulting transform was used in the robot's URDF to define the position of the arm.

## Mobile Base Planning

Before the robot can pick up any parts, it must drive to their location. In order to do that, the robot must be able to determine its location and navigate through its environment.

### Localization

In robotics, localization is the task of determining where the robot is. Localization methods can be classified into one of two groups. Relative localization methods determine the robot’s location with respect to the robot’s previous location, and absolute localization methods determine the robot’s location with respect to an absolute location reference in the robot’s environment, sometimes called the “ground truth.” There are advantages and disadvantages to both classes of localization methods, and ABBY’s localization uses a combination of relative and absolute localization.

Relative localization methods have several advantages over absolute localization methods. Relative localization methods tend to be computationally simple. Odometry, for instance, can be accomplished on a two-wheeled differential drive robot in only 14 mathematical operations (7 addition, 4 multiplication, and 3 trigonometric). Because relative localization is so computationally simple, it allows for high frequency update rates and implementation on embedded processors or in logic circuitry. Relative localization requires no knowledge of the robot’s environment (such as a map), and it does not require the robot’s environment to be instrumented with sensors (to track the robot) or beacons (for the robot to track). Localization methods can also be accomplished with relatively cheap sensors such as inertial measurement units (IMUs), optical flow sensors, and (in the case of wheeled vehicles) shaft encoders.

Odometry Algorithm



















However, relative localization methods all share one major disadvantage. Because each localization update is performed with respect to the previous, error accumulates over time. The source of the error varies from method to method. In the case of wheeled odometry, errors are usually due to wheel slip and the non-linearity of the trigonometric function used to estimate the heading. In the case of IMUs, the error is usually a result of the slow drift of the accelerometers and gyroscopes due to temperature variation. A well-calibrated relative localization system with an accurate observer model will still accumulate error over time, and the estimated position of the robot will slowly diverge from the true position.

There are several different types of absolute localization methods that use different types of sensors. Some methods use external sensors, such as cameras radio frequency (RF) tracking systems to monitor the robot’s position. In order for a robot to use these methods, the operating environment must have already been instrumented with the necessary sensors. Other methods use sensors on the robot to track features of the robot’s environment and compare them to a known map of the robot’s environment. Trackable features include visible features such as lights, signs, or painted patterns; these features may already exist in the environment, such as ceiling lights in an office building, or may be added, such as position-coded labels on a warehouse floor. Trackable features could also be RF beacons, which may already exist (such as WiFi access points), or may be installed specifically for localization. The global positioning system (GPS) is an example of an RF localization system that uses time-of-flight from satellite radio beacons to triangulate the position of the receiver. Systems with ranging sensors can track features of the geometry of the environment itself. To perform an update, all of these localization methods compare sensor data to a map or model of the environment to estimate the robot’s position. This means that they are dependent on an accurate map or model, which may not be possible in an environment with changing features. These methods also depend on the environment having suitable features to localize against. It is difficult to use these methods in featureless environments, such as open fields, and environments with many repeated similar or identical features, such as long hallways.

Although most absolute localization methods require an a priori map (or model) of the environment, one class of absolute localization methods perform simultaneous localization and mapping (SLAM). SLAM algorithms begin with no map of the robot’s environment and incrementally build one as they explore. For example, a SLAM algorithm using a LIDAR can exploit partially-overlapping LIDAR scans to register new scans with respect to previous scans. The translation and rotation required to register the scan can be used to determine the robot’s position with respect to the previous position. Each new scan increases the known region of the environment, slowly building a map. Because SLAM algorithms are incremental, they are more prone to error than localizing against an a priori map or model. In the LIDAR SLAM example, each new LIDAR scan must have enough overlap with the existing map to perform registration. This assumption may be violated if the robot moves too quickly or the geometry of the robot’s environment limits the field of view of the sensor. Once errors are introduced to the robot’s map of the environment, they may be difficult or impossible to correct. Although SLAM methods will usually out-perform relative localization methods, they are less reliable than absolute localization against an a priori map or model. (CITATION)

ABBY’s sensor suite contains several sensors that can be used for localization. The encoders on the wheels are well-suited for odometry, and the gyroscopic yaw-rate sensor can be used for inertial heading measurement. The LIDAR and the Kinect depth camera can both be used for absolute localization using a number of methods, including both SLAM and a priori map localization algorithms. In addition, the Kinect camera could be used for localization based on patterns on the floor. Of these possible methods, odometry was chosen for relative localization, and Adaptive Monte Carlo Localization (AMCL) using LIDAR scans and an a priori 2D occupancy grid map was chosen for absolute localization.

ABBY uses odometry from the encoders on the wheels for relative localization. The physical state observer runs under the real time operating system on the cRIO, and publishes pose estimates with uncertainty (represented as covariance) to the ROS system at FREQUENCY Hz. Relatively high frequency updates to the robot’s pose are required as an input to the local planner, which generates velocity pairs at a rate of FREQUENCY Hz. The odometry system does accumulate error over time. This error was characterized by operating the robot using only the relative localization system and manually driving it along simple geometric paths. When the robot is driven along a ten meter straight line with no observed wheel slip, the odometry error is RESULTS. When the robot is driven in a circle with radius 1m for five laps with no observed wheel slip, the odometry error is RESULTS. As expected, change in heading causes a greater error in the odometric localization. The greater error due to change in heading is a feature of the differential drive system on ABBY. For a differential drive system to turn, one or both wheels must slip, which introduces error into the odometry. In addition, rapid acceleration or deceleration of the robot causes the wheels to slip as the force exerted by the wheels exceeds the static friction between the wheels and the floor. This, in turn, introduces a sudden, relatively large error to the odometry. To limit problems due to wheel slip, the acceleration of the robot was software-limited to below the wheel-slip threshold of (WHEEL SLIP ACCELERATIONS).

It may be possible to improve the accuracy of the relative localization system by fusing in other sensors. The yaw rate sensor on the robot was not used because undiagnosed electrical problems rendered it inaccurate. Other researchers at Case [8], have fused rotational velocity data from a gyroscopic yaw rate sensor with odometry using an Extended Kalman Filter (EKF) . Because the yaw rate sensor is inertial, it is not affected by wheel slip. However, it does drift slowly over time. An EKF with variable measurement covariance, as described in [8] can improve the relative pose estimate, but the author of that research concluded that the improvement was minimal.

AMCL is an absolute localization method that model's the robot's pose as a probability distribution[9]. The robot's pose is considered probabilistic to represent the uncertainty of the sensor measurements used to determine the pose. ABBY uses AMCL to match LIDAR scans to an *a priori* map. On each update, the previous pose distribution is taken as the Bayesian prior and a new measurement is incorporated to calculate a posterior pose estimate. AMCL represents the (continuous) probability distribution of the pose in discrete space with a particle filter using KLD sampling, which adapts the number of sampled points in the distribution based on the covariance of the distribution. As the robot becomes more sure of its location, fewer particles are needed to accurately represent the distribution. In addition, random particles are added to help break the filter out of a false convergence. By starting with particles uniformly distributed through the map, AMCL can theoretically solve the “wake-up robot problem,” in which the robot that is initialized with no estimated pose. However, testing with ABBY showed that AMCL could not reliably solve this problem, and would often converge on a false pose estimate. Instead, ABBY is initialized to a pose at or near the true pose, with sufficiently large covariance that the true pose is within the likely region of the estimated pose. As the robot runs, the pose estimate will converge on the true pose. Each pose update from AMCL takes approximately TIME milliseconds, which means that it can run no faster than FREQUENCY Hz.

Of course, AMCL is only possible with an a priori map. ABBY's maps were generated by the robot itself, using the same LIDAR used for AMCL and the gmapping SLAM package. Gmapping uses a Rao-Blackwellized particle filter to generate maps as it localizes. Gmapping was considered as a possibility for the absolute localization scheme on ABBY. The main advantage of gmapping over AMCL is that it does not require an a priori map, making it easer to install the robot in a novel environment. However, gmapping on ABBY was unable to reliably traverse doorways without accumulating error. This was sufficient reason not to use it for localization. In addition, the maps generated with gmapping were manually edited to remove skews from going through doors.

Combining odometry with AMCL yields results that are better than either one alone. Because AMCL takes so long to compute, it cannot be used to approximate continuous localization. This makes it unsuitable for local planning, which updates at FREQUENCY. Whereas odometry is more suitable for the local planner, the error it accumulates as the robot runs eventually makes it unsuitable for global planning. In order to combine these two methods, two transforms are stored in the robot’s TF tree. One transform is between the robot’s base\_link (a coordinate frame with its origin on the floor between the robot’s wheels) to its parent, the odometric frame odom. This transform is updated by the odometric state estimator on the cRIO at FREQUENCY, and provides an approximately continuous position estimate. Local planning is performed in the odom frame. The top level transform is from the map frame (the absolute coordinate system) to the odom frame and the map frame. This transform is updated by AMCL, which runs an update every time the robot moves more than 0.05 meters in translation or 0.1 radians in rotation. On each AMCL update, the transform from the map to odom frame is change such that it cancels out any error in the tranform from the odom frame to the robot's base\_link. Over a long period of operation the odom transform may accumulate significant error, but the transform from the map frame to the base\_link remains accurate because of the absolute localization updates.

(FIGURE the tf tree from /map to /base\_link with robot model)

Some ways to improve the localization on this robot were beyond the scope of this thesis due to hardware or time limitations. An IMU could have made significant improvements to the relative localization by helping to counteract errors due to wheel slip. Electrical problems with the yaw rate sensor made it unusable for this project, and the robot was not outfitted with accelerometers. However, the recent affordability of six degree of freedom single-chip IMUs would make this an excellent avenue of research to improve the localization system. One obstacle to pursuing this route is that it would require rewriting and retuning the localization EKF on the cRIO, which is beyond the scope of this thesis.

Similarly, optical flow sensors looking at the ground might be used to mitigate wheel slip error. Optical flow sensors bounce light off of a surface and measure how quickly that surface is moving relative to the sensor. A pair of optical flow sensors, mounted near the drive wheels, could supplement the odometry. Because they are unaffected by wheel slip, they might prove more accurate than the existing odometry system. [READ SOME STUFF ABOUT OPTICAL FLOW HERE AND EXPLAIN WHY YOU DIDN’T USE THEM]

Another way to potentially improve the robot’s localization would be to use the Kinect. The Kinect’s limited range and field of view (figure) makes it relatively useless for localization using building geometry such as AMCL or gmapping. However, because the Kinect is facing the floor, it could be used for absolute localization based on patterns on the floor. If the floor is marked with landmarks in the form of painted symbols or codes, the Kinect could be used to detect a landmark and localize the robot with respect to the known location of the landmark in the map. This technique has been used in other industrial mobile robots [10]. A system of visual landmarks on the floor might be useful as a starting seed pose for AMCL. Currently, when the robot is started, the starting pose must be manually entered using Rviz or a launch script. With visual landmarks on the floor, the robot could be started at any arbitrary point in the environment, provided a landmark was visible to the Kinect, and the robot could generate a starting pose estimate from the landmark.

(FIGURE floor map with landmark symbols)

### Mobile Base Trajectory Planning

One of the major tasks for a mobile robot is navigation through its environment. In order for the robot to get parts from inventory, it must first travel through the inventory shelves to the location of the parts. In an industrial application, the location may be retrieved from an inventory database, or it may be specified by a human operator, but the robot's task is the same. From its current location, the robot must plan a path to another location in its environment. The path must avoid obstacles, and it should be as direct and efficient as possible. The robot must then generate a trajectory to follow the path and travel to the goal location. The trajectory cannot violate the dynamic constraints of the robot. Given a trajectory, the robot must execute it by controlling the actuators as accurately as possible to adhere to the desired path and trajectory.

ABBY's differential drive system allows it to move forward and backward and rotate in place, but not move laterally. This makes navigation and control somewhat more difficult than for a holonomic drivebase such as the caster drive system on the Willow Garage PR2.

At the simplest and lowest level of the navigation task is speed control. Speed control on ABBY is implemented as a pair of PID controllers, one for each wheel. The PID controllers are implemented on the cRIO's FPGA for speed and robustness, with loop closure rates of 100 Hz. Each PID controller's setpoint is specified in meters/second and its output is an 8-bit signed integer. A simple geometric algorithm (equation ref), implemented on the cRIO's PowerPC processor, is used to convert twist-style commands (forward and rotational speed) into speed commands for each wheel. These signed integers represent the desired voltage to be output by the Sabertooth motor controller, with -127 being full reverse and 127 being full forward. Since the Sabertooth motor controller can vary its voltage output from -24 volts to 24 volts, the 7 bits of speed resolution in each direction correspond to a voltage output resolution of about 189mV. The PID controllers on ABBY were originally tuned for another similar robot based on the same drivetrain known as ALEN [11]. ALEN was significantly lighter than ABBY and also had a different weight distribution. As a result, it is likely that ABBY's PID controllers are not optimally tuned. This sub-optimal tuning makes it impossible for ABBY to execute low-speed commands because the controllers do not command a high enough voltage to the motors to overcome static friction in the drivetrain. As a result, ABBY's minimum achievable forward/reverse speed is 0.1 m/s and minimum rotational speed is 0.35 rad/sec. This compares unfavorably to the minimum speeds that Eric Perko was able to achieve on HARLIE, which were 0.1 m/second and 0.1 radians/second respectively.





The higher-level components of navigation are path and trajectory planning. Path planning is the task of determining a path from the robot's current location to a desired pose. Trajectory planning takes the path and determines a series of velocity commands to move the robot through the path without violating the acceleration and velocity constraints of the robot. On ABBY, these tasks are performed by a global and a local planner, respectively.

NavFn[12] , the global planner node, operates on a grid-based global costmap populated by the *a priori* map and data from the LIDAR. Given a desired pose, NavFn finds a minimum-cost path using Djikstra's algorithm[13]. This path is defined as a series of intermediate "breadcrumbs," robot poses along the path. NavFn can successfully plan paths for ABBY in relatively open environments, but because it assumes a circular robot base, it will sometimes plan impossible paths in crowded environments.

DETAILS AND CHARACTERIZATION OF NAVFN

The local planner generates trajectories to follow the path produced by the global planner; it operates on a local costmap populated by data from the LIDAR. The robot performs local planning using a dynamic window approach,[9] which forward-simulates translational and rotational velocities and evaluates the resulting trajectories for proximity to obstacles, proximity to the goal, and adherence to the global path. These scores and weighted and summed to determine the trajectory's score. The highest scoring velocity command is sent to the mobile base driver. On ABBY, dynamic window planning sometimes results in unintuitive behavior as the robot approaches the goal. Namely, the robot will sometimes rotate the wrong way, forcing it to turn all the way around to reach the proper heading.

There are some alternatives to NavFn and the base local planner packages used on ABBY. The ROS navigation stack includes a global planner called Carrot Planner,[14] which does not attempt to navigate around obstacles. Instead, it moves as close as possible to the goal along a straight line until it encounters an obstacle, then stops. This planner is only useful in very open environments where a straight-line path to the goal is likely to exist or obstacles are likely to move out of the way for the robot. As such, the planner is not suitable for most industrial environments, which are full of permanently fixed machines, assembly lines, and shelves. In his masters thesis, Perko[8] addressed many of the problems with NavFn and base local planner and devised new algorithms for precision navigation of a mobile robot or wheelchair in an indoor environment. However, Perko's ROS implementations of his algorithms do not conform to the same API as existing ROS navigation nodes, nor do they provide the same functionality. Whereas the existing ROS navigation stack takes an arbitrary Pose (x, y, theta 2D coordinate) as a goal, Perko's path planner requires that all goals be predefined points in an a priori map. Furthermore, the path planner requires that path segments between the goals be predefined. In order to use this global planner in an industrial environment, every possible desired position in the inventory would have to be predefined, as well as a graph of paths between positions. This set-up task would be monumental in a large factory, so Perko's global planner was not used for this project. Perko's local planner uses a combination of local path linearization and a third-order steering algorithm to generate velocity commands. Unlike base local planner, which takes in arbitrary paths composed of a series of poses, Perko's local planner requires that paths be defined as a series of line segments and constant-curvature arcs. This prevents it from being interoperable with NavFn.

## IK solver

An important part of robotic arm planning is an inverse kinematic solver. Given a pose in the robotic arm's work envelope, an inverse kinematic solver determines a set of joint angles that would place the end effector at that pose. Because not all poses have possible solutions and some poses are degenerate cases, analytical inverse kinematic solvers are mathematically complex.

### KDL Solver

ROS includes a kinematic solver in the kinematics\_constraint\_aware package that wraps the inverse kinematic solver of the Orocos project's Kinematics and Dynamics Library (KDL)[15]. The KDL solver is a numerical solver that uses Newton-Raphson iterations. The KDL solver takes joint angle limits into account in evaluation of its solutions, only returning a solution with valid joint angles. However, the KDL solver will often fail for achievable poses and, because it uses an iterative numerical method, runs slowly, on the order of tens of milliseconds.

To counteract the problem of the KDL solver failing, the solver was wrapped in a method to retry the search on failure. When the solver fails to find an inverse kinematic solution for a desired pose, the solver is reseeded with randomly selected joint values and called again. To test the solver, poses were generated using forward kinematics of simulated joint angles. Since the poses were generated by forward kinematics, they are guaranteed to be achievable. Many of the test poses were not solvable by the unmodified kinematic solver, which only made one attempt at a solution. When the number of attempts was increased to 100, the success rate approached 100% for solvable poses. Since the seed is random, the number of requests needed to solve for a difficult pose is probabilistic, allowing for some possibility of inverse kinematic failure. The resulting inverse kinematic solver is suitably reliable, though slow, taking as long as [TIME] seconds before determining that a pose is unsolvable. This is acceptable for the robot at this time, but a faster inverse kinematic solver will be necessary in an industrial installation.

[Characterization table or graph]

### IKFast

The OpenRAVE project includes the IKFast kinematic solver tool,[16] which analyzes a kinematic chain and generates C++ code for an inverse kinematic solver for that chain. The solver it generates is a closed-form analytic solver, meaning it runs very quickly (<1 ms) and can handle the vast majority of achievable poses, including degenerate cases. The greatest limitation of IKFast is that it does not take into account joint limits, but assumes that all joints can rotate freely. This means that IKFast will often generate solutions that are unachievable for a rotation-limited robotic arm.

The ROS wrapper for IKFast rejects solutions that violate joint constraints, but since IKFast will only return the first eight IK solutions it finds, it is likely that none of these solutions will satisfy joint constraints, particularly for degenerate cases. In these situations, the ROS inverse kinematic service fails, despite the existence of a valid solution. It may be possible to rewrite the IKFast compiler or modify the generated code to work with joint limits, but at this time, IKFast was determined to be unsuitable for arms of ABBY's geometry due to the high incidence of solver failure.

## Arm Navigation

ABBY's arm navigation package is closely based on the standard ROS arm navigation package, which is generated from a URDF by a tool called the Planning Description Configuration Wizard. This wizard allows the user to define a manipulator kinematic chain from a URDF file and then generates an arm navigation application, including the necessary launch files for an inverse kinematics plugin, a planner, and a collision environment server. This "default" arm navigation application was augmented with filtering nodes for the Kinect data going into the collision environment and the modified KDL inverse kinematics plugin described above.

The trajectory planner for the arm is the default planner available with the ROS arm navigation package. It uses a sample-based planning algorithm from the Open Motion Planning Library called Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking (SBL). [17] SBL plans collision-free paths for the arm and publishes these paths as ROS trajectory messages, which the ROS Industrial arm driver executes. Because SBL performs all of its planning in joint-space, the arm navigation application uses the inverse kinematics plugin described above to convert goal poses into joint space before planning paths to them.

### Collision Detection

The robot maintains a collision map, a 3D occupancy grid represented by an Octomap oct-tree. The collision map is populated by data from filtered Kinect point clouds. In addition to this "raw" map, a collision environment is maintained, which contains information from this map and keeps track of detected objects such as the manipulable objects detected by the tabletop box recognition system described below. The objects are given IDs and stored in the map as meshes or geometric solids, enabling the arm navigation code to make decisions about whether collisions are allowable.

The arm navigation code navigates around obstacles in the collision environment, including the robot itself, preventing the robot from damaging itself and the objects around it. Collisions can be selectively allowed between robot links and objects in the collision environment. For instance, it is impossible to pick up an object without the jaws of the gripper colliding with it. When creating a request to make the final approach to pick up an object, collision detection between the gripper jaws and the object is disabled, allowing the robot to pick the object up while still preventing collisions between the robot and other obstacles.

### Kinect Data Filtering

The Kinect's field of view includes the robot's arm, which the collision detection process would classify as obstacles. This causes the robot to freeze because it is in collision with a perceived obstacle, even though the obstacle is the robot itself. In order to prevent this, the Kinect point cloud is pre-filtered to remove points that are within the robot volume, which is simulated by padding the 3D model of the robot generated from the URDF. The resulting point cloud, which has all points within the robot removed, is used for all collision monitoring and object detection.

## Tabletop Box Manipulation

[object manipulation block diagram]

Once ABBY arrives at the desired location in the inventory, it must identify objects and pick them up from the inventory shelves. This is an area for future researchers to improve upon, but a basic object perception and manipulation system was implemented as a proof of concept. The system uses several object detection nodes developed by Willow Garage for the PR2 robot, as well as two nodes written specifically for this project.

### The Manipulation Controller

The Object Manipulation Controller serves as the central control node, translating and routing messages between the perception and manipulation nodes. It provides a callable method pick\_objects, which performs the task described in [ALGORITHM]

detected\_objects = tabletop\_detection(kinect\_data)

for each object in detected\_objects:

if graspable(object):

pick(object)

place(object, bin)

stow\_arm()

The tabletop\_detection() step is performed by the tabletop object segmentation package, developed by Willow Garage [18] for the PR2 robot. This software identifies a tabletop surface in a point cloud using RANSAC [19] and adds it to the collision environment as an object. Objects on top of the table (detected\_objects) are segmented into separate point clouds. These point clouds are inserted into the collision environment as Graspable Objects. These Graspable Objects are used later by the manipulation package during the pick() step to identify and locate the objects in the robot's environment. The pick() and place() steps are performed by the box manipulator package, which exposes them as action services using a standard pick-and-place API defined in ROS. This standard API allows the box manipulation package to be easily replaced by a new manipulation package, or the manipulation controller to be replaced by a more sophisticated package. In the final step, stow\_arm(), the arm is moved to a predefined stowed position, which minimizes the robot's footprint while it is driving.

### Box Manipulation

The box manipulator package is designed to be able to lift small boxes from a shelf or table and place them. It provides two services—one to pick up a graspable object, and one to place the currently held graspable object at a set of coordinates.

When the manipulation controller calls the pick service on a Graspable Object in detected\_objects, it uses another node created by Willow Garage for the PR2 to fit a bounding box to the Graspable Object's point cloud. Since the objects being manipulated are themselves boxes, the bounding box is a fairly accurate representation of the object. This bounding box is then used to generate an approach path composed of two poses. The first is a pregrasp pose close to the box and with the gripper jaws parallel to the sides of the box. This pose is sent to the arm navigation package, which generates a trajectory and moves the arm to the pose. The second pose is the grasp pose, with the box between the gripper jaws. This pose is also sent to the arm navigation package, and once the robot is in the grasp pose, the gripper is closed around the object.

Once the gripper is closed around the object, the object manipulation controller calls the place() service with a pose in the robot's onboard storage bin as a target. This service sends the pose to the arm navigation package, which generates a trajectory and moves the arm to the bin. The gripper is then opened.

## QR Code Recognition and 3D Localization

Mobile manipulators in industrial settings can benefit from information about the objects they manipulate. Most of the explorations of object manipulation have been directed toward general purpose object recognition and manipulation, but the problem of determining how to pick up and object can be greatly simplified by simply putting manipulation information on the object itself. For more complex information, the object could link back to an entry in a database. Properties salient to mobile industrial manipulation, such as grasp affordances, mass, volumetric data, and visual cues for registration and localization, can all be encoded into visual or RFID tags on manipulable objects. To this end, an exploration was made into using the Kinect in conjunction with a higher-resolution camera to read QR code tags on boxes and use the QR code to perform 3D localization of the box.

[QR code image]

QR codes can hold up to 3 kilobytes of data, or 174 bytes with 7% error correction. They include 3 "finder square" fiducials, which can be used to acquire and correct the skew and size of the code. QR codes are already used throughout the automotive industry for part labeling.[20]

Information about how to grip an object can be stored in a small amount of data in a QR code on an object. This information, which I call an “affordance cue,” contains, in a compact form, information that aids a robot in deciding how to manipulate an object:

* If an object has handles, the affordance cue contains the information necessary to define the handles and locate them relative to the tag.
* If an object can be lifted only from the bottom, as with a fork-lift, the affordance cue contains information necessary to locate the bottom of the object relative to the tag.
* If the object can be grabbed from the sides by a gripper, the affordance cue contains gripping force constraints and the information necessary to locate the sides relative to the tag.

[box with points image | Red dots are the QR code finder pattern. Blue is the projected handle center. Green are the handle corners.]

Using a Kinect sensor and MatLab, a rudimentary proof-of-concept was tested for boxes with handles on top. Because the Kinect image sensor does not have high enough resolution to read QR codes, it must be supplemented with a second, higher resolution camera extrinsically calibrated to the Kinect. This part of the detection was simulated, and the QR code data manually entered into the program. The coordinates of the three finder points in the Kinect RGBD image are then used to get the positions of the three finder points in 3D space. Again, because a high resolution camera was not available, this step was simulated by manually picking points. Using the 3D coordinates of the 3 finder points, it is possible to localize the QR code in 3D space, including both position and orientation. Using the location of the tag and the tag’s information about the handle's location relative to the tag, the handle can be located, allowing a robot to grasp it without knowing any other information about the box.

Because the Kinect RGB camera does not have sufficiently high resolution for the task, these experiments were not performed on the robot. However, this line of research is one of the planned future purposes of the robot, once a high resolution camera can be acquired, mounted, and calibrated to the Kinect.

# Industrial Safety

Mobile robots, particularly experimental platforms, require safety systems to disable them. These systems fall into two major categories. Reflexive halt and speed and separation monitoring systems keep robots from colliding with humans operators and workers interacting with the robot. Emergency stop systems are used to disable the robot in case it does something unsafe or unexpected. Both types of systems are necessary in an industrial robot, especially one that does not operate in a cage or with guards, neither of which are practical for a mobile robot.

## Reflexive Speed Limiting

A key safety feature of many robots is reflexive speed limiting or halting. Reflexive halting uses sensors to detect obstacles in the robot's path and prevent motion that would result in a collision. Reflexive speed limiting is particularly important for robots that operate in shared environments with people, because people can suddenly move into a previously clear robot path, endangering themselves and the robot. Whereas a planner will create safe paths for a robot around static obstacles, paths may be invalidated by moving obstacles such as people, and the planner may react too slowly to prevent a collision. Reflexive halting operates more quickly and on a lower level than planning and trajectory generation, and can override the velocity commands from the trajectory generator.

At this time, ABBY does not implement a reflexive speed limit for the base or the manipulator. Because both operate at low speeds, the robot does not pose a safety threat to the operator and can be easily stopped with the emergency stop system described below. Before the robot can be tested or deployed in an industrial environment, and to allow for faster movement, reflexive speed limits must be imposed. This section describes proposed methods for reflexive speed limiting.

### Reflexive Halt Methods for Mobile Bases

Mobile robots often implement a reflexive halt as shown in ALGORITHM. For example, if a measurement source such as a LIDAR reports that there is an obstacle in front of the robot at close range, the robot's velocity would be limited to turning and reversing. This approach is effective, but can cause difficulties in navigating tight areas, where there may be enough space for a robot to navigate through obstacles if it does so carefully.

Given:  
 Sensor measurements, M  
Do:  
 for each measurement m in M:  
 if(dangerous(m)):  
 prevent motion in direction of m

Another approach to reflexive halting was described by Chad Rockey in his masters thesis.[21] This approach, called Reflexive Avoidance Plus, was developed for a smart wheelchair, which must operate in crowded areas around people. Reflexive Avoidance Plus uses velocity limiting rather than preventing motion altogether. When a sensor detects an obstacle in the robot's path, it limits the maximum velocity in that direction using a scaling function based on the distance of the obstacle from the robot. This approach prevents the robot from colliding with an obstacle (the maximum velocity is zero below a certain threshold distance), but allows low speed progress toward an obstacle. Because the robot approaches the obstacle at lower speed, it can safely get closer to obstacles.

Since the Reflexive Avoidance Plus method described by Rockey was implemented on a robotic platform of similar size and speed to this robot, it would be a good candidate for this robot. However, there is some work still to be done to adapt the code, which was designed specifically for the wheelchair, to ABBY.

### Reflexive Halting for Manipulators

In addition to the mobile base, ABBY's robotic arm poses a risk for collisions. In industrial situations, manipulators are kept inside safety cages to prevent people from interfering with them or getting injured. A safety cage is not a possibility for a mobile manipulator, so another solution is necessary. A reflexive halt for the manipulator allows it to operate safely in the presence of people and obstacles without a safety cage.

Like the mobile base planner, the planner for the arm generates collision-free paths. However, the planner for the arm does not replan at all once it commits to a trajectory. The trajectory is generated and sent to the IRC5 for execution, and then ROS waits for the trajectory to be executed. If something enters the path of the trajectory, the robot does not alter the current trajectory. This makes the robot unsafe to operate around humans above a certain joint speed.

Rethink Robotics, with their robot Baxter, [22] solved the problem of operating an industrial robot without a safety cage with a mechanical and software solution that relied on force feedback and serial-elastic actuators. Because all of Baxter's joints are elastic and its arms are so light, it can safely collide with people and obstacles. It also uses force feedback in its joints to detect these collisions and become passive, allowing people to push it around. Because ABBY's robotic arm does not have serial elastic actuators or force feedback in the joints, this solution is not possible.

Using the Kinect sensor, it would be possible to implement one of several possible reflexive collision avoidance methods on ABBY. One method would be to halt arm motion when an obstacle enters the arm's work envelope and suspend arm motion until the obstacle leaves the work envelope. Although this is arguably the safest solution, it can cause the robot to become stuck in the stopped state. If an inanimate obstacle is brought into the robot's work envelope and left there, the robot will never re-enable the arm.

To resolve this problem, the reflexive halt behavior can be augmented as shown in ALGORITHM. In this algorithm, the currently planned path is repeatedly checked for dangerously close object (such as people) until execution is completed. If an object enters the dangerous area, the robot stops execution of the trajectory and waits. If the obstacle leaves the area before a timeout is reached, the robot resumes execution of the trajectory. If the obstacle does not move, the robot will attempt to retry planning to move around the obstacle to accomplish its goal. This reflexive halt algorithm has two advantages over the naïve algorithm described in the previous paragraph. First, it does not stop for obstacles that enter the work envelope but do not interfere with the planned motion. This allows humans to work alongside the robot and interact with it by giving it objects or taking objects from it. Second, it will plan around stationary objects that enter the work envelope, allowing a person to leave an object in the work envelope without stalling the robot.

Given:  
 Current Plan P, Measurements M  
Do:  
 for each state s in P, measurement m in M:  
 if(dangerous(m, s)):  
 halt  
 wait for obstacle to move or replan

The National Institute for Standards and Tests (NIST) has developed an algorithm to determine a safe separation distance S for a human to approach a robot. The equation [EQUATION], where KH is the speed of a moving human, KR is the speed of the robot, TR is the reaction time of the human, TB is the braking time of the robot, B is the robot braking distance, and C is a distance to account for measurement uncertainty. [23]

NIST researchers used this equation, LIDAR scanners, and a Kalman filter to track humans moving through the robot's work envelope and determine whether a human had entered a danger zone around the robot (if the distance from the robot to the human is less than S). This method completes algorithms [ALGORITHM] by filling in the dangerous() function. In order to implement this algorithm on ABBY, parameters B and C would first have to be determined for this platform. Then, the algorithm would have to be written as a ROS node and tested on this robot.

## Emergency Stop System

### E-Stop Systems Used in This Lab

The Case Mobile Robotics Group has used a few emergency stop systems in its robots. All of the HARLIE-class robots developed for the Intelligent Ground Vehicle Competition used a commercially-available wireless relay system from Remote Control Technologies. This system, shown in FIGURE, consisted of the remote control relay in series with an onboard disable switch and a second relay switched by an active-high enable signal from the cRIO, which is software-controllable. These three switches (one manual and two relay) control the current through the coil of a solenoid, which in turn switches the power to the motor controller on and off. This system has one critical flaw, which is that there is no “heartbeat” from the wireless remote to the remote control relay. This means that if the battery in the wireless remote dies or the radio communication is lost between the wireless remote and the robot, there is no way to remotely stop the robot, nor is there any indicator to the operator that the robot cannot be wirelessly stopped.

OTTO the smart wheelchair used a custom remote mode switching system designed to interface with the Arduino-based control system used to control the wheelchair drivetrain. Using a pair of XBee 2 Pro wireless network modules, a GPIO signal is transmitted from a remote control unit to the input of an Arduino on the wheelchair, disabling the autonomous functions of the wheelchair when a button is pushed on the remote. Because the Xbee wireless modules' GPIO mirroring has a programmable timeout and default output state, this system automatically disables autonomous functions if communication is lost between the remote and the robot. However, this system does rely on an Arduino microcontroller and was not designed as an emergency stop system, but as a switch between autonomous operation and normal (joystick) operation of a wheelchair.

### E-Stop Requirements for This Robot

This robot has several requirements that motivated the development of a new emergency stop system combining the merits of the HARLIE-class stop system and the OTTO remote switching system. First, the emergency stop system needs to be able to switch the high current, 24 volt rail providing power to the motor controllers. Second, the system needs to be able to activate the 24 volt emergency stop input on the IRC5 robot controller, which must be electrically isolated from the rest of the robot's DC electronics. Third, the system must monitor four sources, stopping the actuators if any of them are disabled:

1. 5 volt active-high enable signal from the cRIO, which is controlled by the ROS software
2. 24 volt active-high emergency stop signal from the IRC5, which is controlled by the emergency stop switches on the IRC5 and FlexPendant and by RAPID software.
3. Twist-lock stop switch mounted on the robot (The robot is disabled if the switch is opened or disconnected.)
4. Wireless remote control with a heartbeat signal of at least 1Hz

Fourth, the system should be implemented entirely in hardware for safety reasons. Software faults in a safety system are unacceptable and adequate testing of a software system would be too time-consuming for this project. Fifth, the remote control unit should have some feedback as to the state of the four emergency stop sources described above.

### Version 1 Prototype

Given the requirements, an emergency stop system was designed and fabricated using printed circuit boards. The schematic of the system is shown in FIGURE. The system consists of two circuits, a remote control and an emergency stop circuit on the robot.

SCHEMATICS GO HERE

The remote circuit uses an XBee radio module's GPIO mirroring function is used to transmit the state of the emergency stop button to the emergency stop circuit on the robot in the same manner it was used on OTTO. This system also uses the GPIO mirroring function to send the states of the other emergency stop sources to the remote, where they are displayed on LEDs. Because a twist-lock style emergency stop button was not available, an S-R latch was used to latch the state of a normally-open momentary pushbutton, requiring that the remote be powered off and back on again to reset the wireless emergency stop.

The emergency stop circuit on the robot has inputs for the onboard emergency stop button, the cRIO's enable signal, and the emergency stop output of the IRC5. The input from the IRC5 goes into an optoisolator IC because the IO on the IRC5 is floating relative to the rest of the robot's DC systems. A 7400 series AND IC is used to generate logic signals to enable the drive base and the IRC5's emergency stop input. The drive base logic signal controls a Darlington transistor, which in turn switches the coil of a solenoid that controls the drive base in the same manner as on HARLIE-class robots. The IRC5 output logic signal switches the 24v General Stop input of the IRC5 using an optoisolator IC.

These circuits were prototyped and installed on the robot. The input from the IRC5's emergency stop was defeated by installing jumper J1 because the output had not been configured in RAPID software. Additionally, testing showed that the 4N35 optoisolator used to switch the IRC5's emergency stop could not switch enough current to enable the emergency stop circuit, causing the IRC5 to go into General Stop mode seemingly at random. This function was defeated by disconnecting the IRC5   
Stop output and shorting the General Stop input of the IRC5. These two changes completely decouple this emergency stop circuit from the IRC5, meaning it no longer meets requirements 2 and 3b described above. Furthermore, the wireless link between the XBee modules proved unreliable, causing the system to momentarily switch into emergency stop mode seemingly at random. Extensive bench testing of the system suggests that this problem is caused by an insufficiently reliable power supply to one or both of the XBee modules. In order to make the system usable, the wireless emergency stop was replaced with a twist-lock style emergency stop button on a ten foot wired tether. This modification means that the system no longer meets requirements 3d and 5 described above. The system does reliably control the power to the drive base, providing a level of safety for the robot, but revisions are required to make the system function as specified.

### Version 2 Design

A revised version of this emergency stop circuit was designed and portions of it prototyped, but it has not been tested. This version should fix the problems discovered in the first version of the emergency stop.

SCHEMATICS HERE

To integrate the system with the IRC5, the optoisolator on the output of the emergency stop circuit was replaced with a relay module, which will more reliably switch the General Stop input of the IRC5. To complete integration with the IRC5, the RAPID software must be modified to output the current General Stop state to a GPIO, which must be connected to the IRC5 input of the emergency stop circuit.

To solve the wireless communication issues, the power supply in the remote was replaced with a 3.3 volt boost supply, which should be much more reliable, and bypass capacitors were added to the power rails of the XBee modules on both the remote and the emergency stop circuit. Testing has shown that the XBee modules are reliable when a sufficiently clean and reliable DC supply is available to power them.

In addition to solving the problems described above, some small changes were made to improve the circuit. To reduce the power consumption of the emergency stop circuit and reduce the heat produced by the onboard power regulator, the Darlington transistor used to switch the coil of the drivebase enable solenoid was replaced with a MOSFET circuit that performs the same function. To make the system easier to use and more reliable, the momentary switch and latch used on the previous version was replaced with a twist-lock style emergency stop switch.

Although this system has not been constructed, its constituent parts have been tested individually. The MOSFET switching circuit has been confirmed to work with a resistive load equivalent to the coil resistance of the solenoid used to switch the drive base power rail. The power supply circuit in the remote has been tested and provides a reliable 3.3 volt power supply from a pair of AA batteries. The use of a relay instead of a transistor to control the General Stop input is in line with recommendations from ABB's documentation, and the relay used meets the requirements. If the necessary components can be acquired for this emergency stop circuit, it should be able to meet all of the requirements described above.

# Experimental Results

# Applications and Future Work

# Conclusion