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.

1

2 Introduction

3 Mechanical Design of Abby

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.

3.1 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). (ACCEL AND SPEED DATA). 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.

3.2 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 (DIMENSIONS) and heavy, 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 chosen, 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 crawl under 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 the Kinect camera and allows the WiFi router to be mounted far away from possible interference from other electronics.

3.3 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 decision to mount the arm vertically on the front of the robot was so that the majority of the arm's work envelope would be 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.

For this project, the IRB-120 was fitted with a parallel plate gripper. Although a more dexterous gripper, such as a BarrettHand, would have been desirable, one of the goals of this project was to create this robot as cheaply as possible. This 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. It has deformable conveyor belt material for the gripping surface, which gives it a secure grip and some ability to pick up irregularly shaped objects. In addition to the deformability of the gripping surface, the pneumatic nature of the system makes the gripper jaws back-drivable, with a constant gripping force of (LB), as calculated in equation (EQUATION NUMBER) with the pneumatic system's adjustable regulator set to (REGULATOR SETTING). The regulator can be set to any pressure up to the system's maximum pressure of 120 PSI. 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.

4 Electronics and Sensor Packages

4.1 Power Systems

All of the robot's power is distributed using DIN rail power distribution blocks. These blocks are modular, insulated, and compact. The robot has three voltage buses (24 volt DC, 24 volt DC with emergency stop, 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. In addition to the robot's drivetrain, the robot's PC, LIDAR, and the National Instruments cRIO are both powered directly from the 24 volt DC bus.

There is a second 24 volt DC bus that is used to power the robot's drive train and a signal light to indicate that the bus is powered. This bus is disconnected from the main 24 volt bus by a solenoid relay when the emergency stop is activated. The safety features of the emergency stop are described in more detail below.

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.

4.2 Pneumatic System Control

The electrical components of the robot's pneumatic system are comprised of a 12 volt DC compressor, digital pressure switch, and a double throw 12 volt DC solenoid valve. The compressor turns on to pressurize the robot's pneumatic accumulator tanks, which provide the air pressure to the robot's pneumatic gripper. The compressor must be turned on to pressurize the tanks and turned off when the tanks reach their maximum pressure of 120 PSI. The digital pressure switch is attached to the robot's pneumatic system. It is normally closed, but opens when the pressure exceeds 120 PSI. The switch is not rated for the current required to power the compressor, so the switch is used to control an Innovation First Spike relay, which switches the compressor on and off.

The pneumatic solenoid valve is a magnetically actuated valve with one pressure inlet and two pressure outlets. The inlet is connected to the output of the pneumatic 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 the IRC5 Compact's 24 volt DC GPIO. Since the coils are meant to be operated at 12 volts DC, a (VALUE) resistor was put in series with the coil to divide the voltage down to 12 volts.

In addition to the three 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.

4.3 Drivetrain Control and Odometry

The drivetrain's two brushed DC motors are controlled by a Dimension Engineering Sabertooth 2x25 DC motor speed controller. The Sabertooth 2x25 takes in a 24 volt DC supply and outputs power to two brushed DC motors with voltage modulated by pulse width modulation at 32 kHz. The motor outputs are independently variable from -24 volts to + 24 volts, which correspond to full speed reverse and full speed forward respectively. The Sabertooth motor controller's output voltage is commanded by serial packets from the National Instruments cRIO.

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.

4.4 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.

4.5 Microsoft Kinect

4.6 Sick LMS-291

4.7 End Effector-Mounted Camera

4.8 Safety Systems

4.8.1 Emergency Stop

This robot has a new emergency stop circuit design that is meant to improve upon emergency stop systems previously used on robots in the mobile robotics lab. Historically, the lab used a wireless remote e-stop system from Remote Control Technology. However, this system had some shortcomings that made it dangerous. The most glaring shortcoming was that there was no heartbeat signal from the remote control. If the remote's batteries die or the remote goes out of range of the robot, there is no way to stop the robot and no way for the operator to know that the emergency stop is no longer reliable. Chad

Rockey made improvements upon this with the emergency stop he design for the autonomous wheelchair Otto. Otto's emergency stop circuit used a pair of xBee Pro radio modems to mirror the state of a digital input on the remote to a digital output on the robot. This digital output was monitored by Otto's computer systems, which would disable the autonomous functions of the robot if the digital signal dropped low. This system was superior to the Remote Control Technology solution because the robot's xBee Pro receiver was configured to set its digital output low in the event that it lost signal from the remote's xBee Pro transmitter.

Despite the improvements of Otto's emergency stop system over the previous solution, it was designed for a smart wheelchair, not a robot. An emergency stop for a robot should not rely on software and must be able to cut power to the drive base and other actuators. In order to solve these issues, a new emergency stop circuit was designed specifically for Abby. Like Otto's emergency stop, this system uses a pair of xBee Pro radio modems to mirror digital signals between the remote and the robot. However, this system adds several new features. First, this system has four different ways for the robot to be stopped in an emergency. The robot has an onboard emergency stop button, an emergency stop system in the ABB IRC5 controller, and a software-controlled emergency stop signal output by the cRIO. These three signals and the output of the wireless emergency stop must all be logic high for the robot's actuators to be enabled. Rather than relying on software to monitor these signals, they are combined with an AND gate IC in the emergency stop circuit to output a combined emergency stop signal, which will go to logic low if any one of the input stop signals is logic low. In addition, the state of each of the three onboard emergency stops (button, IRC5, cRIO) is sent to the remote control, where they are indicated by LEDs on the remote.

In addition, the emergency stop circuit has two outputs. The first output is used to enable and disable the robot's drive base. This output is a 24 volt output that supplies current to the coil of the emergency stop solenoid. When current is applied to the solenoid's coil, the voltage bus powering the motors is electrically connected to the main 24 volt bus by the solenoid relay. When any of the emergency stops are triggered, the emergency stop circuit stops delivering current to the emergency stop solenoid coil, which disconnects the two buses, removing power from the motors. The second output is used to enable and disable the ABB robotic arm. This output of the emergency stop circuit is a relay that is used to control the General Stop input of the IRC5. When no emergency stops are triggered, the relay is closed, connecting the IRC5's 24 volt rail to the General Stop input, which enables motion of the ABB IRB-120. When an emergency stop is triggered, the relay is opened and a pull-down resistor on the General Stop input brings the General Stop input to the IRC5 ground level. This disables motion of the ABB IRB-120. To prevent race conditions, the ABB IRC5's emergency stop output is not ANDed into the signal sent to the General Stop input. This signal is controlled with a relay to provide isolation between the IRC5, which is powered by an inverter, and the DC circuitry of the rest of the robot.

4.9 Computing

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.

4.9.1 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.

4.9.2 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 realtime 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.

4.9.3 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 Flex-

Pendant 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.

5 Software

5.1 ROS

The robot's software is all running 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. For example, ROS provides several navigation nodes, each implementing a different algorithm. Because the message interface is standardized across these nodes, they are "drop-in" replacements, which makes it easy to experiment with different algorithms without changing any code in the rest of the robot's software. ROS also has a vast library of existing nodes and algorithms, allowing researchers to leverage prior work without having to reimplement the algorithms.

5.1.1 ROS Topics

ROS nodes communicate to each other by sending messages to each other on topics. Messages have predefined types that declare what 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 character strings, and can be 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 there is direct communication from the publisher to the subscriber, and the ROS master node only facilitates this communication my maintaining a list published topics and negotiating the direct connections between nodes.[**?**]

5.1.2 ROS Services

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. Other ROS nodes can be service clients. A service client sends a request message to the service server; the request may be an empty request for a service, or it may contain parameters or data to be processed. The service server performs the

service requested and replies with the reply message; the reply message may be an empty confirmation message, or it may return data or a status message about the service.[?]

5.1.3 ROS Actions

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. This makes them more appropriate for requests that make take a long time, such as moving an actuator or querying a sensor. Unlike services, 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. [?]

5.2 Hardware Drivers

In order for ROS to read data from a sensor or send commands to an actuator, it must have 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.[?] 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.

5.2.1 The Mobile Base

The mobile base is controlled by software running on the cRIO, as described above in Hardware. 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 deactive 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 converts pose information into a standard ROS message type.

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 velocity commands from the 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.

5.2.2 ROS Industrial

ROS Industrial is a project led by SWRI to develop a standard ROS framework for using ROS with industrial robots.[?] 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.

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. I experimentally determined that 250 points was sufficiently

long for trajectories for the IRB-120.

5.2.3 Gripper Driver

Since the gripper is a combination of custom hardware, there was no existing ROS driver to control it or monitor its state. The gripper driver is a ROS node that runs natively on the Arduino's AtMega 328 microcontroller using the ROS Serial framework. It sends and recieves ROS messages over the USB serial connection. A ROS node on the PC 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. Since ROS serial did not properly support ROS services in the Fuerte version of ROS, I expanded the ROS serial framework to enable it to handle services running on microcontrollers.

5.3 Localization

The task of localization is to determine the robot's position in its environment. One method of localization is to use odometry from the wheel encoders. The advantage of odometry is that it is (approximately) continuous and can be updated very rapidly. The odometry localization on ABBY updates at FREQUENCY Hz. However, odometry is prone to error due to wheel slip. This was particularly problematic on ABBY because so much of ABBY's weight is on the front (non-driven) wheels. When the drive wheels slip, it introduces error to the odometry. The error is particularly problematic if it affects the robot's estimated heading.

An absolute localization method is used to solve the problem of accumulated error in the odometry. Absolute localization is performed with respect to a map of the robot's environment. Map-referenced localization runs more slowly than odometry and is therefore discontinuous. Since discontinuous localization is unsuitable for motion control, both odometry and absolute localization are performed simultaneously, and two separate transforms are used the define the robot's position. The first transform is from an odometry frame to the robot base frame and is updated by the odometry-based state estimator. The second transform is from the map (world) frame to the odometry frame and is updated periodically by the absolute localization algorithm at a slower rate. This second transform cancels out the error that accumulates in the first transform between absolute localization updates.

ABBY uses Adaptive Monte Carle Localization (AMCL)[?] for absolute localization. AMCL uses a particle filter to compare scans from the LIDAR to an a priori map of the robot's environment. By evaluating many randomly-sampled robot pose candidates, the algorithm determines the robot's most likely pose. AMCL was already implemented as a ROS node that subscribes to laser scan data and a map server and publishes transforms. Once tuned for the robot's

sensor and odometry parameters, AMCL is able to correct a wheel slip error within a few meters of driving.

AMCL requires an a priori occupancy grid map[?], which was generated using the GMapping Simultaneous Localization and Mapping (SLAM) algorithm.[?] Although SLAM can be used in place of AMCL to perform world-referenced localization without an a priori map, it was inadequate for our robot due to the frequent occurance of wheel slip. In particular, GMapping failed when going through doorways into unknown areas. Without a known map region to compare laser scan data to, the new area could become skewed with respect to the known area. Once this skew error was introduced to the generated map, it was nearly impossible for GMapping to correct it. The maps generated by GMapping were manually corrected to remove skew error using image processing programs. Given this problem with SLAM, I chose to use SLAM only to generate maps (which I manually corrected) for later use with AMCL.

5.4 Path Planning

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.

Planning for the mobile base is performed by a local and a global planner, each implemented as a ROS node. NavFn[?] is a global planner node that operates on a grid-based global costmap populated by the a priori map and data from the LIDAR. Given a set of desired endpoint coordinates, NavFn finds a minimum-cost path using Djikstra's algorithm[?]. 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. 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,[?] 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 [_planner?] which does not attempt to navigate around obstacles.