# 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

1. 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.
2. 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

* 1. 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.
  2. 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.
  3. 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.
  4. GRIPPER CONTROL SCHEMATIC
  5. 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.
  6. 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.

* 1. 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

### Sick LMS-291

## 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 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.

### 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.\cite{ros-pub-sub}

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.\cite{ros-srv}

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. \cite{ros-actionlib}

## 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.\cite{igvc-2010-harlie} 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 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.

### ROS Industrial

ROS Industrial is a project led by SWRI to develop a standard ROS framework for using ROS with industrial robots.\cite{ros-industrial} 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.

### 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.