Building a Mini-Sumo Robot

Overview

Building a mini-sumo robot using the IntelliBrain™ robotics controller is a great way for you to learn about robotics and Java™ software development. The RoboJDE™ Java-enabled robotics software development environment (included with IntelliBrain) allows you to download the example mini-sumo robot application discussed here into your IntelliBrain controller. With RoboJDE you can modify the example mini-sumo application or write your own mini-sumo program to improve the competitiveness of the robot.

Mini-sumo is a competition between robots based on Japanese wrestling – "sumo" is the Japanese word for wrestling. Similar to traditional sumo matches, two opponents (robots) face each other in a ring named a dohyo – see Figure 1. The object is to stay in the ring while pushing the opposing robot out of the ring. The robot that stays in the ring the longest wins the match.

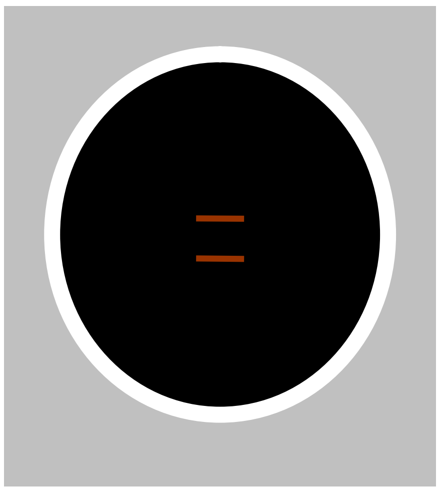

Figure 1 - Mini-Sumo Dohyo

Mini-sumo robots are small autonomous mobile robots designed specifically for sumo style competition. The mini-sumo robot competition rules restrict the robot length and width to 10 cm x 10 cm, but do not restrict its height. In addition, the robot cannot weigh more than 500 grams. Unlike larger battle robots, mini-sumo robots are not allowed to damage the opponent robot, they are only allowed to push it off the dohyo.

All the hardware components you will need to build the robot discussed here are readily available. A mini-sumo robot can be assembled in a few hours or less. Once you have assembled your robot, you can load the mini-sumo example program included with the RoboJDE software and have your robot ready for its first competition in just minutes! Once you have your robot up and running, the well documented example source code, the extensive robotics class library, and the easy to use integrated development environment will make improving your robot's intelligence and competitiveness a snap!

If you are new to mini-sumo competition, *Robot Sumo: The Official Guide* by Pete Miles (McGraw-Hill/Osborne 2002) is an excellent book about building robots, the rules for competitive robot sumo events and how to build your own dohyo.

Theory of Operation

To be effective, a mini-sumo robot must be able to do the following:

- stay on the dohyo
- hunt for the opponent
- target (aim at) the opponent
- attack the opponent

The following sections describe how the mini-sumo robot accomplishes these things.

Sensing the Dohyo Edge

The mini-sumo robot relies on two Fairchild QRB1134 infrared photo-reflectors to detect the white edge of the dohyo. These sensors are mounted at the left and right front corners of the robot, facing the surface of the dohyo. The sensors generate a low voltage output (~.2 volts) over the highly reflective white edge and a higher voltage output (~4.8 volts) when over the less reflective black center of the dohyo. By sampling the output voltage of the sensors frequently – using the analog-to-digital converter built into the microcontroller – the software can determine if the robot is at the edge of the dohyo.

Sensing the Opponent

Two Sharp GP2D12 infrared range sensors are used to sense and target the opponent. These sensors are mounted on the left and right front of the robot, facing forward. Similar to the line sensors, they output an analog voltage. The voltage is higher the closer the sensor is to an object. The sensor outputs approximately 2.5 volts when an object is 10 centimeters away and 0.4 volts when an object is 80 centimeters away. The sensor is not effective outside of this range. The sensor is more precise at the shorter distance end of the effective range.

Moving and Steering the Robot

Two servo motors attached to the two wheels provide the mechanism to move the robot. Steering is accomplished by rotating the wheels at different speeds. When one wheel turns slower than the other, the robot will move in an arc curving to the side of the slower turning wheel. By turning the wheels in the opposite direction of each other, the robot will rotate in place.

Servo motors have built-in control circuitry designed to rotate the output shaft to a specific position and hold that position. The servos used in the mini-sumo robot are designed for use in model airplanes to control the position of the airplane's ailerons and rudder. These servo motors are light, relatively powerful, and inexpensive. However, to be useful for the mini-sumo robot the servos must be modified to allow for continuous rotation of the wheels and variable speed control. Fortunately, this is fairly easy to do, or you can purchase the servos pre-modified.

The servo modification involves removing mechanical stops that limit the range of rotation and disconnecting the position sensing potentiometer from the rest of the mechanism. Once this is done, the potentiometer is calibrated so the servo's control circuit always thinks the output shaft is at the center of its range.

Unmodified servos are controlled by periodically pulsing the servo's signal input, as shown in Figure 2. A 1 millisecond pulse commands the servo to position the output shaft to one end of its range (0%). A 2 millisecond pulse commands the servo to position the output shaft at the other end of its range (100%). By varying the pulse width, the shaft can be positioned between these two extremes.

Modified servos use the same control signaling but the modification results in the ability to control the speed of the shaft rather than its position. This is exactly what is needed to steer the robot. Disconnecting the position sensing potentiometer from the rest of the mechanics and calibrating it to the 50% (centered) position causes the control electronics in the servo to think the shaft is always centered. If the control signal commands the servo to a position other than the center, the control electronics will apply power to rotate the shaft, attempting to move it to the commanded position, but the feedback from the potentiometer leads the control circuit to think the shaft hasn't moved, so it will continue to power the motor and spin the output shaft. The control circuit will apply more power the greater it thinks the error is between the desired position and the actual position. Therefore, the power applied to the motor can be varied by varying the pulse width of the control signal, giving the ability to drive the robot's wheels at different power levels, which results in different speeds if the robot's motion is not obstructed.

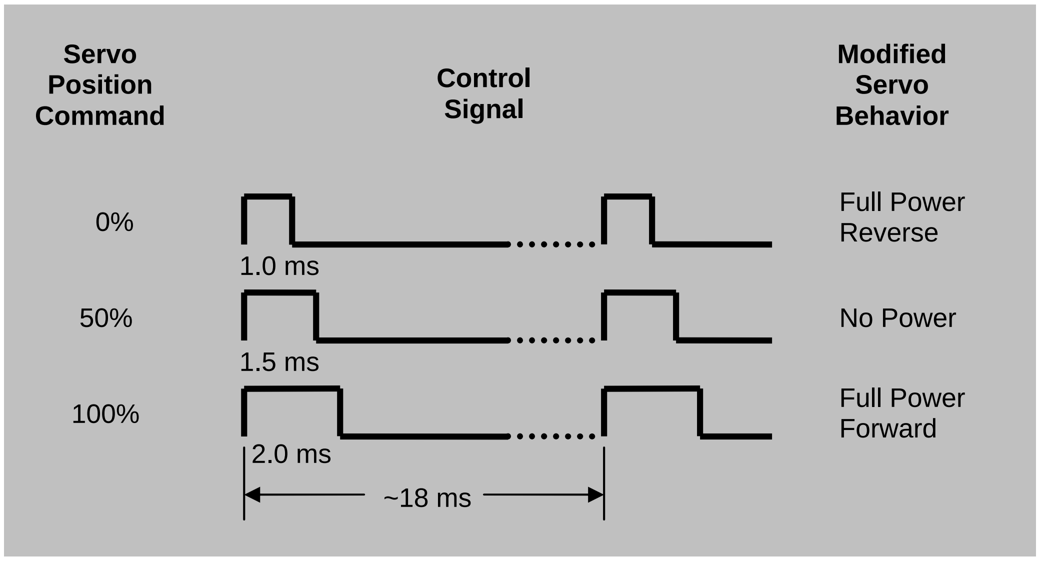

Figure 2 - Servo Control Signaling

Fortunately, the details of controlling the servos are managed by the RoboJDE class library and RoboJDE virtual machine, so the application doesn't need to manage the servo control signals directly. Instead, the application makes a simple method call to set the power level.

Controlling the Robot's Behavior

The previous sections described how the robot senses its environment and how the servo motors enable it to move. Software implements the robot's intelligence, linking its behavior to the data it collects from its surroundings via sensors. Even for a simple robot there are nearly unlimited possibilities as to how the robot can behave. A particular behavior may work well against one opponent and work poorly against another opponent.

The mini-sumo example application implements a finite state machine in software to control the robot's behavior. This state machine is depicted in Figure 3.

A state machine is a way to organize a program into a number of easy to understand states that each performs a well defined function to operate the robot in a specific situation (state). Events that reflect a change in the robot's situation cause a transition from one state to another.

The state machine diagram shown in Figure 3 provides a graphical representation of the states and the events that cause transitions between states. The states are represented as named bubbles and the events that cause state changes are shown as arrows named by the event that caused the change.

The software implementing a particular state only deals with the actions and transitions which must be handled when the system is in that state. The software for other states handles the cases relevant to those states. This helps to structure the program into small, well defined and easy to understand pieces.

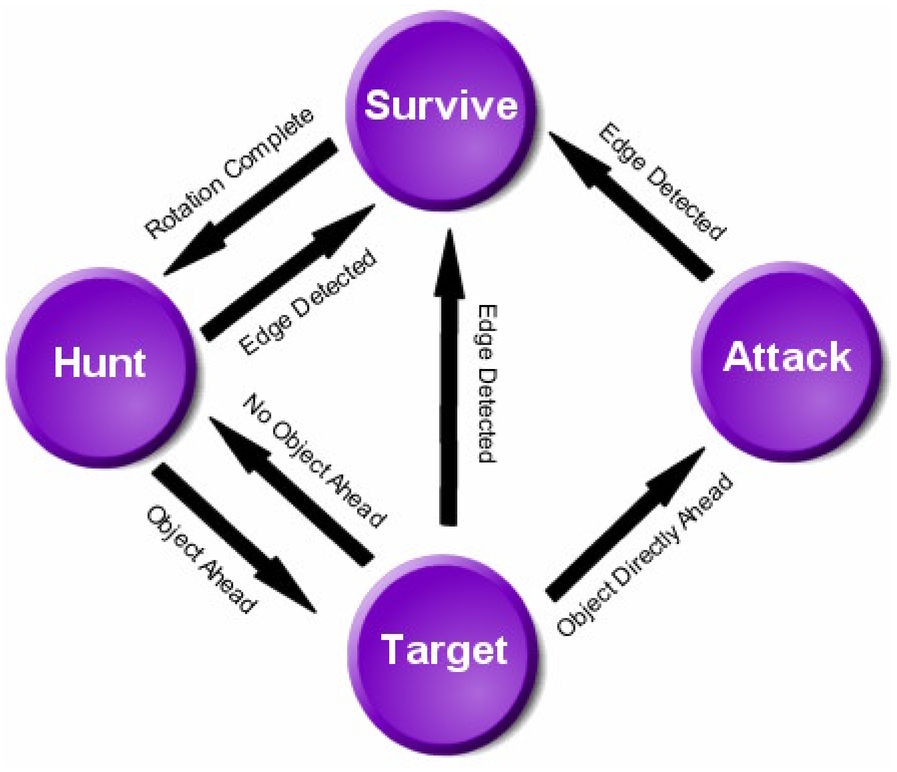

Figure 3 - Mini-Sumo Software State Machine

The mini-sumo example application has four states: SURVIVE, HUNT, TARGET, and ATTACK. These states are described in Table 1.

Table 1 - State Machine Summary

State	Description	Actions
SURVIVE	The robot enters this state when it detects the dohyo edge. Its goal is to survive by not going off the dohyo. The robot rotates away from the sensor that sensed the edge to face back toward the center of the dohyo.	1. Rotate away from the line sensor that sensed the edge. 2. Switch to HUNT state when rotation is complete.
HUNT	The robot is not at the edge of the dohyo but hasn't sensed the opponent. The robot moves around in an arcing pattern so its range sensors will sweep across the ring in hopes of sensing the	1. Switch to SURVIVE state if dohyo edge detected. 2. Switch to TARGET state if range sensors

	opponent.	indicate an object ahead. 3. Otherwise, drive in an arc by applying more power to one wheel than the other.
TARGET	The opponent has been sensed ahead. Aim the robot to face the opponent directly.	1. Switch to SURVIVE state if dohyo edge detected. 2. If still sensing opponent, but opponent is not directly ahead turn slightly to aim at opponent. 3. If opponent is directly ahead and close switch to ATTACK state. 4. If opponent is ahead but not close, move straight forward. 5. Otherwise, switch to HUNT state.
ATTACK	The opponent has been found and aiming is complete. Drive straight ahead at full power to push the opponent off the dohyo.	1. Switch to SURVIVE state if dohyo edge detected. 2. Otherwise, drive straight forward at full power.

Mini-Sumo Hardware

All the hardware components you will need to build your mini-sumo robot are listed in Table 2.

Table 2 - Mini-Sumo Hardware

Qty	**Part**	**Supplier**
	Controller	
1	IntelliBrain main board	www.ridgesoft.com
	Chassis, Sensors and Actuators	
1	Mark III chassis kit	www.junun.org
2	Servo (GWS SO3N 2BB)	www.junun.org
1	Pair of molded wheels	www.junun.org
2	Sharp GP2D12 range sensor	www.junun.org
2	Fairchild QRB1134 IR photo-reflector	www.junun.org
2	JST cable for GP2D12	www.junun.org

IntelliBrain Robotics Controller

The IntelliBrain robotics controller mounts directly on the Mark III mini-sumo chassis, and fits within the 10 cm x 10 cm mini-sumo size limit. The RoboJDE development environment included with the IntelliBrain controller includes the example mini-sumo application discussed previously. To use this program open the project:

```
Program Files\RoboJDE\Examples\MiniSumo\IntelliBrainMiniSumo.rpj
```

Mark III Chassis, Sensors and Servo Motors

The Mark III chassis comes unassembled, so you must assemble it yourself. (There is a fully assembled version, but it includes a controller card which you do not need when you use the IntelliBrain controller.) Directions for assembling the Mark III are available on the Junun web site (www.junun.org).

Note: *When attaching the GP2D12 sensors, insert a 1/16 inch thick #4 screw size washer between the scoop and the sensor. Otherwise, the sensor will point down slightly due to contact between the IntelliBrain controller board and the GP2D12 sensor.*

The Mark III uses hobby aircraft servos which must be modified for continuous rotation. You may purchase the servos pre-modified, or modify them yourself according to the directions on the Junun web site. Be sure your servos are modified before assembling the Mark III.

Attach the sensors to the Mark III according to the directions on the Junun web site. The mini-sumo application only requires two line sensors, left and right, so you will not need to install a line sensor at the center of the scoop.

Sensor and Servo Wiring

You will need to attach Molex connectors on the sensor wires to connect the sensors to the IntelliBrain controller. The parts and tools you will need are listed in Table 3. See the *IntelliBrain User Guide* for more information on connecting various sensors and effectors to the IntelliBrain controller.

Table 3 - Sensor Connector Parts and Tools

Part	**Molex Part Number**	**Digi-Key Part Number**
Crimp Terminals	16-02-1109	WM2555-ND
3 Circuit Housing	50-57-9003	WM2801-ND
4 Circuit Housing	50-57-9004	WM2802-ND
Universal Crimp Tool	63811-1000	WM9999-ND
Insertion Tool	11-02-0022	WM9911-ND
Wire Cutter & Stripper		

Figure 4 shows which ports you should use when connecting the sensors to the IntelliBrain controller. It also shows the order of the color coded sensor and servo wire connections to the IntelliBrain controller.

Connecting Servos

The servo motors have connectors pre-installed, so they may be plugged in without attaching a connector, however, BE VERY CAREFUL TO CONNECT THE SERVO PLUG TO THE INTELLIBRAIN IN THE CORRECT ORIENTATION! The black wire should be on the pin at the front edge of the board. Plugging the servo in the wrong way around will likely destroy the servo's control circuitry. The left servo connects to servo port 1, the port nearest the left edge of the board. The right servo connects to servo port 2.

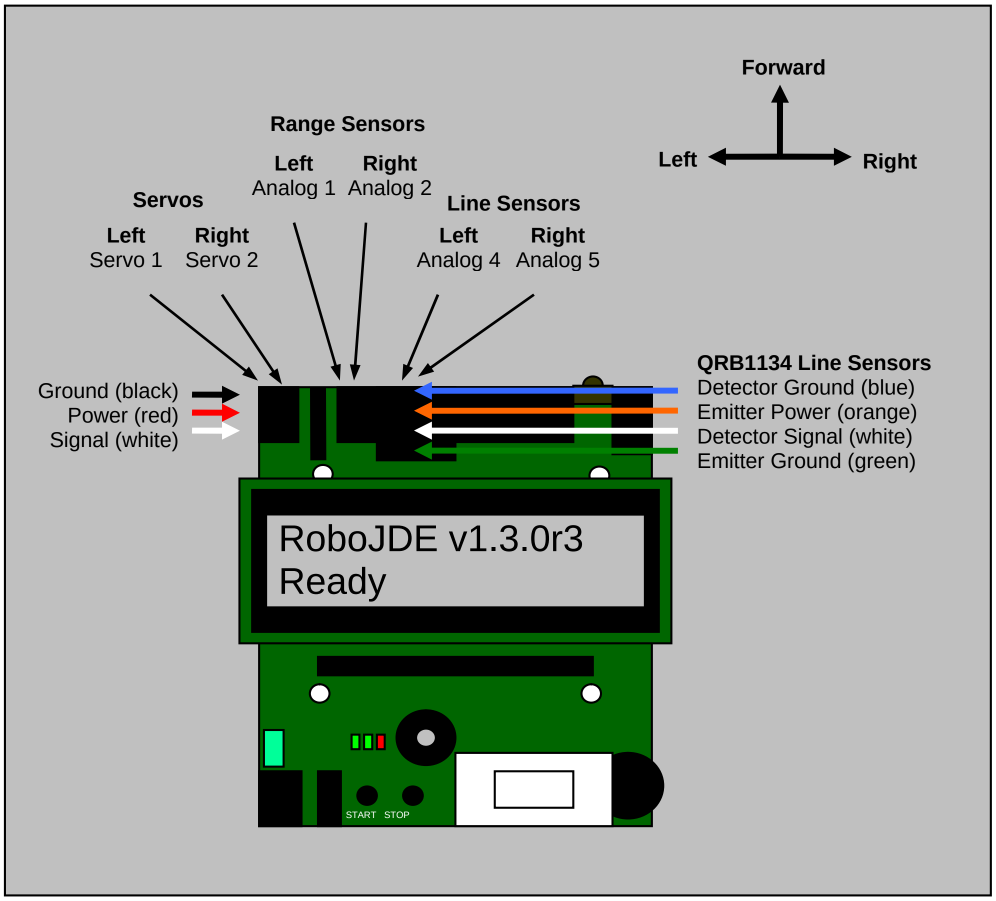

Figure 4 – Sensor and Servo Connections

Connecting Line Sensors

Trim the line sensor leads to appropriate length (about 6 inches) and attach a 4 circuit Molex connector to each sensor. The wires should be arranged in the connector in the following order: blue, orange, white, and green, as shown in Figure 5. Connect the left line sensor to analog port 3 and the right line sensor to analog port 4, as indicated in Figure 4.

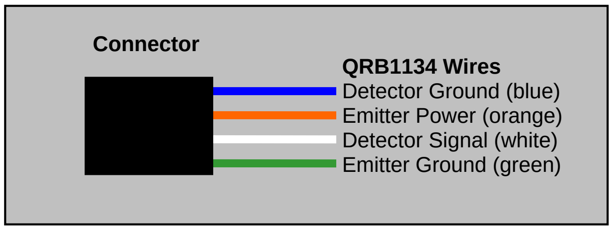

Figure 5 - Line Sensor Connector

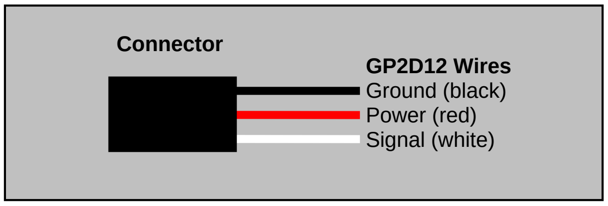

Figure 6 - Range Sensor Connector

Connecting Range Sensors

Trim the range sensor leads to appropriate length (about 4 inches) and attach a 3 circuit Molex connector to each sensor. The wires should be arranged in the connector in the following order: black, red, and white, as shown in Figure 6. Connect the left range sensor to analog port 1 and the right range sensor to analog port 2, as indicated in Figure 4.

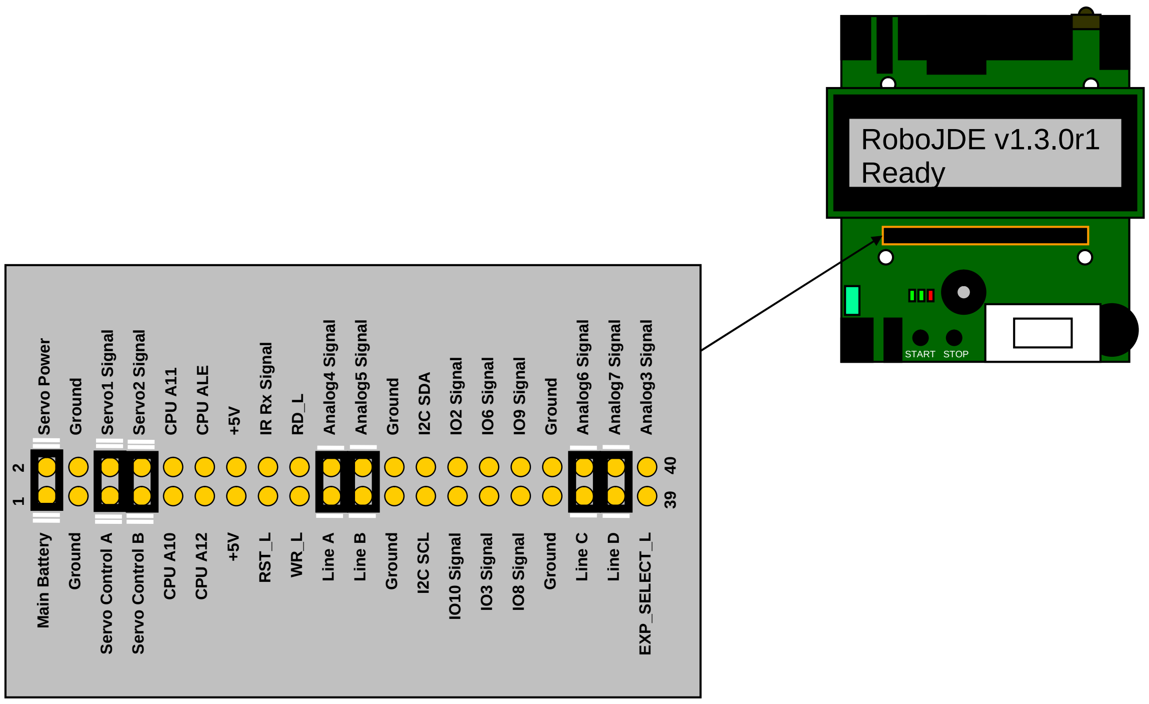

Figure 7 - Expansion Header Jumpers

Installing Expansion Header Jumpers

Ensure jumpers are installed on the expansion board header as shown in Figure 7. Note: If you have the IntelliBrain expansion board installed, the jumpers are not necessary.

Mini-Sumo Software

Once you have constructed your mini-sumo robot you must download the example application to the robot. Loading the software and testing your robot is quick and easy. The following sections discuss how to load the software, test that the sensors are functioning, and run your robot.

Installing RoboJDE on Your Host PC

If you haven't already installed RoboJDE on your host PC, you must install it from the CDROM that came with your IntelliBrain controller. Once RoboJDE is installed, launch it from the Windows Start menu. Use the Tools->Settings menu item to set the Board Type to IntelliBrain, the Serial Port to the COM port your PC will be using to communicate with your robot, and the Baud Rate to 38400 (unless you configured your IntelliBrain to use a different baud rate).

Connecting the IntelliBrain to the Host PC

Connect a serial port cable between the COM port on the PC and the DB9 connector at the rear of the IntelliBrain controller board. *Note: The serial cable must be a straight-through male to female (extension) cable.*

Installing a RoboJDE License Key

If you haven't already done so, install the license key provided on the label on the back of the CDROM case. Select the Tools->Install License Key menu item and enter the key in the license key dialog.

Building and Loading the Mini-Sumo Example Application

To load the mini-sumo example application, click the "Open Project" button on the tool bar then browse to and select the project file"

```
Program Files\RoboJDE\Examples\MiniSumo\IntelliBrainMiniSumo.rjp.
```

Next, click on the "Download" button on the tool bar. This will compile the example application and download it to the robot. Finally, press the "Run" button on the tool bar to start the application. Alternatively, you can press the "START" button on the robot to start the application. By default, the application will be stored in RAM and will be lost when the robot is powered off. If you want the IntelliBrain to remember the program when power is switched off, you must load the application to flash memory. To do this, use the drop down selector on the tool bar to set the load location to FLASH and repeat the download process. Note: Loading programs to flash memory is slower and the life of the flash memory is limited, so it is usually desirable to load your programs to RAM when you are developing and debugging them.

Testing the Sensors

The example mini-sumo application has built-in sensor tests. These tests start running whenever you start the application. Pressing the START button a second time ends the tests and starts the mini-sumo application.

There is one test for each of the four sensors. You can scroll through the tests by rotating the thumbwheel. The LCD screen displays the name of the sensor under test and the current value reported by the sensor. The line sensors should read a high value (~950) when over a black surface and a low value (~40) when over a white surface. The range sensors should read approximately 500 when 4 inches from a wall. The range sensor reading should drop as the robot moves away from the wall. Be sure to check that each sensor reads correctly and that the left and right sensors are not swapped.

Running the Mini-Sumo

Place the robot on a mini-sumo dohyo, power it on and press the START button twice. When you release the button the robot will begin its five second countdown, as required by the mini-sumo rules. Once the countdown is complete it will start moving. If everything is working correctly, the robot will wander around the dohyo in an arcing pattern and spin around each time it reaches the edge of the dohyo. If you place an object similar in size to the robot on the dohyo, the robot will find it and push it off the dohyo.

Congratulations! Your robot is now ready for mini-sumo competition!

Software Implementation

The mini-sumo example application has been implemented in Java. If you are not familiar with Java programming, Java is a high-level programming language with a syntax derived from C and C++. If you have experience with C, C++ or another high-level programming language, understanding the mini-sumo example source code should not be difficult. If you would like to learn more about Java programming, the *Java Primer* on the RidgeSoft web site contains a brief overview of Java programming. There are also many good educational resources available, including the book, *Just Java* by Peter van der Linden (Sun Microsystems Press).

The mini-sumo example is primarily implemented by the class `BasicMiniSumo.` (A class can be thought of as a source file which defines the data and methods related to an object. In this case the object is the mini-sumo controller.) This class implements all of the control algorithms in a way that is independent of the actual robot controller being used. This allows the same program to run on other robot controllers, such as the Sumo11, with only a few minor differences in the main class file.

`IntelliBrainMiniSumo.main()` *Method*

The application starts in the `IntelliBrainMiniSumo.main()` method by retrieving a reference to the `Display` object from the class `HandyBoard` using the following statement:

```
Display display = IntelliBrain.getLcdDisplay();
```

The Display object provides a method to print text on specific lines of the LCD screen. The program identifies itself to the user with the following statement:

```
display.print(0, "iBrainMiniSumo");
```

Following this, an object of the `BasicMiniSumo` class is instantiated. (Instantiating a class in Java creates an object that is an instance of the class by allocating memory for the object and initializing the object.) This is done by using the `new` operator to allocate the memory and call the constructor method (the method named `BasicMiniSumo`). References to a number of objects are passed to the constructor method in addition to a number of constants that may be used to tune the robot's behavior. The class `BasicMiniSumo` requires `Motor` objects for the two drive motors. However, the robot actually uses servos that have been modified for continuous rotation. The class `ContinuousRotationServo` from the class library implements the `Motor` interface to an encapsulated `Servo` object. (An interface is a list of methods that, when implemented by a class, provides a subset of functionality that is useful independent of other features of the class.) This creates a façade that makes a `Servo` object controllable as a `Motor`, allowing motor control software to control the modified servo just as it would control a conventional motor. The `ContinuousRotationServo` class also allows the sense of rotation to be reversed, which is necessary for the right servo. Forward rotation for the right servo is actually reverse from the robot's perspective. The left servo is attached to servo port 1 and the right servo is attached to servo port 2. The `Motor` façade objects are created by the following statements:

```
new ContinuousRotationServo(IntelliBrain.getServo(1), false)
new ContinuousRotationServo(IntelliBrain.getServo(2), true)
```

The `BasicMiniSumo` class also requires `AnalogInput` objects for each of the sensors, which are obtained as follows:

```
IntelliBrain.getAnalogInput(4) // left line sensor
IntelliBrain.getAnalogInput(5) // right line sensor
IntelliBrain.getAnalogInput(1) // left range sensor
IntelliBrain.getAnalogInput(2) // right range sensor
```

References to these objects are passed to the `BasicMiniSumo` constructor. The constructor initializes the `BasicMiniSumo` object's member variables, `mLeftMotor, mRightMotor, mLeftLine`, `mRightLine`, `mLeftRange`, and `mRightRange`, respectively, with these references. (Member variables are variables holding a specific object's data. The member variables exist as long as the object exists.)

Once the `BasicMiniSumo` object is constructed and initialized, control is passed to it by calling its `go()` method:

```
sumo.go();
```

`BasicMiniSumo.go()` *Method*

This method calls the `displaySensorReadings()` method to start the sensor test described previously. The sensor test terminates when the START button is pressed. When the start button is released, the 5 second countdown loop is started. (At the start of a mini-sumo match the robot is required to sit idle for 5 seconds after the contestant steps away from it.) The program loops 5 iterations, displaying the number of seconds remaining, sleeping 1000 milliseconds (1 second) each time through the loop. The sleep statement is as follows:

```
Thread.sleep(1000);
```

Finite State Machine Control Loop

When the countdown is complete, the program enters an infinite while loop that executes the control state machine until the robot is turned off or reset, as shown in the following code fragment:

```
while(true) {
      …           // sensor sampling code
      switch (state) {

      case SURVIVE:
            …     // survive code
            break;

      case HUNT:
            …     // hunt code
            break;

      case TARGET:
            …     // target code
            break;

      case ATTACK:
            …     // attack code
            break;
      }
}
```

Each time through the loop, the program samples the sensors and then switches to the code for the current state.

Sensor Sampling

The line sensor analog input is sampled and converted to a boolean value which is true if the sensor is detecting the edge of the dohyo. This is done by sampling the `AnalogInput` and comparing it to a threshold value. If the reading is below the threshold value it is assumed that the sensor is over the edge of the dohyo. (The sensor reading is lower when over white than when over black.) This is done as follows:

```
	boolean atLeftEdge = (mLeftLine.sample() < mEdgeThreshold);
	boolean atRightEdge = (mRightLine.sample() < mEdgeThreshold);
```

The range sensors are analog inputs. The sensors will read high when close to an object. The range sensors are sampled and the average and the difference of the two range sensors are calculated using these statements:

```
	int leftRange = mLeftRange.sample();
	int rightRange = mRightRange.sample();
	int rangeDifference = leftRange - rightRange;
	int rangeAverage = (leftRange + rightRange) / 2;
```

If the opponent is sensed, the `rangeAverage` will be higher than when the opponent is not in front of the sensors. The `rangeDifference` is used to target the opponent. If one sensor senses the opponent, but the other does not, the `rangeDifference` will be high. If the opponent is straight ahead the `rangeAverage` will be high and the `rangeDifference` will be low.

Each state needs to check for the edge of the dohyo. To avoid duplicating the check in several states and to accelerate transitions to the SURVIVE state, this check is done before the switch statement:

```
	if (atLeftEdge || atRightEdge)
		state = SURVIVE;
```

The behavior of the robot in various states is described in the Theory of Operation section. The following sections provide the code fragments for each state.

SURVIVE State

```
	if (atLeftEdge) {
		rotate(-135);
		searchClockwise = true;
	}
	else if (atRightEdge) {
		rotate(135);
```

```
            searchClockwise = false;
      }
      state = HUNT;
```

HUNT State

```
      if (rangeAverage > mTargetThreshold)
            state = TARGET;
      else if (searchClockwise)
            arc(mHuntPower, mHuntFactor);
      else
            arc(mHuntPower, -mHuntFactor);
```

TARGET State

```
      if (rangeDifference > mTargetThreshold)
            spin(false, mSpinPower);
      else if (-rangeDifference > mTargetThreshold)
            spin(true, mSpinPower);
      else if (rangeAverage > mAttackThreshold)
            state = ATTACK;
      else if (rangeAverage > mTargetThreshold)
            forward(mTargetPower);
      else
            state = HUNT;
```

ATTACK State

```
      forward(mAttackPower);
```

Motor Control Methods

The `BasicMiniSumo` class contains the following methods that control the motors in order to achieve certain motions:

- `forward(int power)`
- `arc(int power, int factor)`
- `spin(boolean clockwise, int power)`
- `rotate(int degrees)`
- `stop()`

Forward

The `forward` method attempts to move the robot straight forward by setting the power to both motors to the same level.

```
      mLeftMotor.setPower(power);
      mRightMotor.setPower(power);
```

Arc

The `arc` method attempts to move the robot forward arcing to the left or right by applying slightly more power to one motor than the other.

```
      mLeftMotor.setPower(power + factor);
      mRightMotor.setPower(power - factor);
```

Spin

The `spin` method attempts to spin the robot in place without a specified stopping point by running the motors in opposite directions.

```
      if (clockwise) {
            mLeftMotor.setPower(power);
            mRightMotor.setPower(-power);
      }
      else {
            mLeftMotor.setPower(-power);
            mRightMotor.setPower(power);
      }
```

Rotate

The `rotate` method attempts to rotate the robot a specified number of degrees by running the motors in opposite directions for a fixed time per degree of robot rotation.

```
      if (degrees < 0) {
            degrees = -degrees;
            mLeftMotor.setPower(mRotatePower);
            mRightMotor.setPower(-mRotatePower);
      }
      else {
            mLeftMotor.setPower(-mRotatePower);
            mRightMotor.setPower(mRotatePower);
      }
      try {
            Thread.sleep(degrees * mRotateFactor);
      }
      catch (InterruptedException e) {}

      stop();
```

Stop

Turns off the motors.

```
      mLeftMotor.setPower(0);
      mRightMotor.setPower(0);
```

Enhancing the Mini-Sumo Robot

The example application presented here implements a fairly simple control algorithm. There are many alternatives you may experiment with to improve your robot's competitiveness. The following list includes some ideas on how to improve your robot:

- tune the control constants to enhance the robot's performance
- improve the aiming in the targeting state

- add steering in the attack state
- add more sensors to detect the opponent from any direction
- add sensors to sense being pushed backward off the dohyo
- add shaft encoders to sense and better control the robot’s motion
- add different types of sensors such as sonar
- add new states such as escaping when being pushed by the opponent
- implement a behavior system using the BehaviorArbiter class
- experiment with new motion algorithms in the existing states
- randomize the behavior of the robot to make the robot less susceptible to weaknesses of a particular behavior
- use multi-threading to execute portions of the control software concurrently, for example, sampling sensors at the same time as executing maneuvers

Conclusion

Building a robot is a fun and rewarding way to learn about and enjoy electronics, mechanics and software development. The IntelliBrain robotics controller is very powerful and flexible, allowing you to build highly intelligent robots. RoboJDE provides a modern software development environment, a rich robotics software library, example applications, and a robust execution environment that will make developing your robotic software a pleasure. And, because Java is a modern, standardized and widely used language, the programming skills you acquire through robotics can be applied to many other fields.