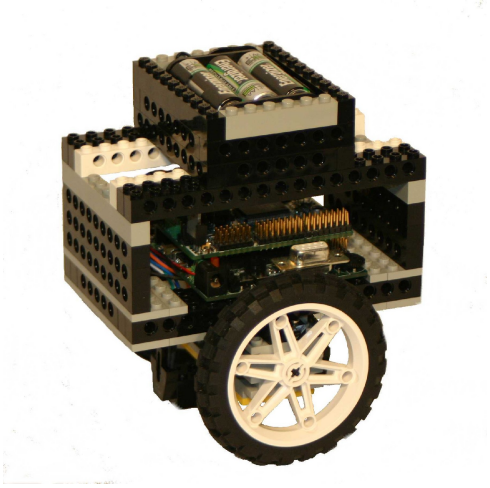
Building a Balancing Robot with the IntelliBrain™ Robotics Controller and Lego® Bricks



Introduction

Who says building a balancing robot has to be expensive and difficult? With just two inexpensive infrared photoreflectors, an IntelliBrain™ robotics controller and a handful of Lego® parts you can build and program your own balancing robot ("BalanceBot") before you know it!

Theory of Operation

The balancing robot shown above is a highly unstable two wheeled robot. The largest mass, the battery pack, is positioned above the axle, making the robot an inverted pendulum. The robot will naturally tend to tip over, and, the further it tips, the stronger the force causing it to tip over.

Unlike the Weebles toys that were marketed several decades ago, "Weebles wobble but they don't fall down!" the BalanceBot will not wobble, it will just quickly fall down without a feedback control system to stabilize it. While this sounds like a difficult problem that would require a gyroscope or accelerometers and programming advanced algorithms, it is amazingly easy and can be accomplished with just two inexpensive Fairchild QRB1134 photoreflectors, an IntelliBrain robotics controller and a short Java™ program.

The key to keeping the BalanceBot upright is the ability to measure the robot's lean, allowing the control software to power the motors in a way that will eliminate all but the slightest lean.

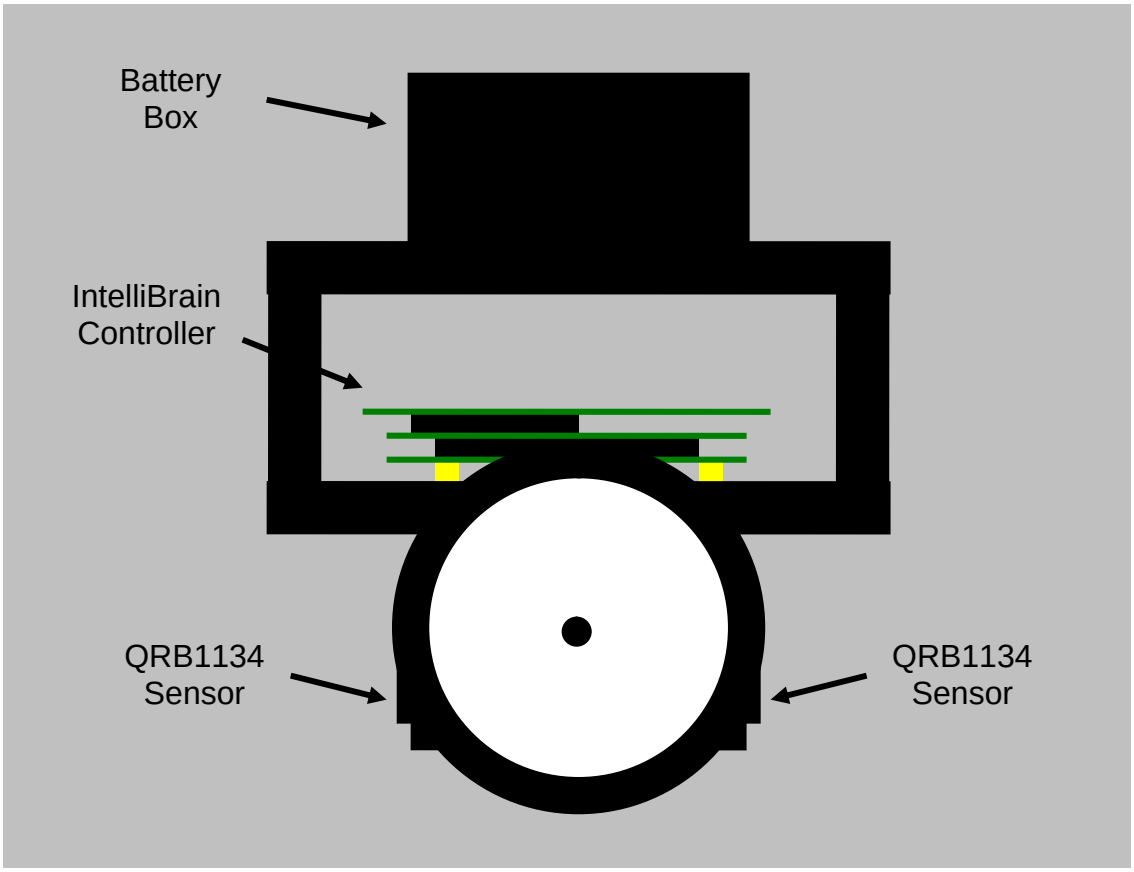


Figure 1 - BalanceBot Side View

Measuring the BalanceBot's Lean

The BalanceBot uses two Fairchild QRB1134 infrared photoreflector sensors mounted about ½ inch above the ground and equidistant from the axle, as shown in Figure 1, to measure how far it is leaning, if at all. These sensors measure reflectivity and are sensitive to the color and texture of the floor. By taking the difference in the readings of two sensors mounted equidistant from the axle centerline, the BalanceBot can determine the direction and magnitude of its lean. Assuming the floor has a uniform color and texture taking the difference of the sensor readings will factor out most of the characteristics of the floor, generating a value that is nearly proportional to the robot's lean. The difference between the sensor readings will be near zero when the robot is perfectly upright. If the BalanceBot is leaning, the difference will be positive or negative depending on the direction of its lean. The magnitude of the difference will increase as the lean increases. Built-in variations in the sensor readings due to differences in mounting height and sensor differences are cancelled out by the software by using the IntelliBrain controller's thumbwheel to calibrate the set point (difference measurement) that corresponds to the BalanceBot being perfectly upright.

Keeping the BalanceBot Upright

The software running on the IntelliBrain controller is responsible for keeping the BalanceBot upright. The software uses a Proportional-Integral-Differential (PID) control algorithm to do this. The software measures the lean then inputs it into PID control algorithm to determine how to power the motors in order to stabilize the robot and stand it upright.

As its name suggests, the PID algorithm consists of a three part equation with proportional (P), integral (I), and differential (D) terms. These terms determine the controller output (power to the motors) from its input signal (the difference between the two sensor readings).
The proportional term increases the motor power as the BalanceBot leans further over and decreases the motor power as the BalanceBot approaches the upright position. A gain factor, `pGain`, determines how much power to apply to the motors for any given lean, as follows:

```
proportionalTerm = -pGain * sensorDifference;
```

While the proportional term is effective at responding to the lean, once the BalanceBot reaches the upright position it will proceed to tip in the opposite direction until the proportional control term increases the motor power enough to reverse the robot's motion, rotating it back in the other direction. The robot will oscillate back and forth, just as a car with worn out shock absorbers bounces for a long time when the car goes over a bump.

The differential term of the PID algorithm acts as a damper reducing oscillation. This term resists rotation with a resistance proportional to the speed of rotation. Another gain factor, `dGain`, determines how much power is applied to the motors in the opposite direction of motion according to the following equation:

```
differentialTerm = dGain * (previousDifference - sensorDifference);
```

The `previousDifference` is the difference in sensor readings on previous iteration of the control algorithm.

Finally, neither the proportional nor differential terms of the algorithm will remove all of the lean because both terms go to zero as the orientation of the robot settles near vertical. The integral term sums the accumulated error (sensor differences summed over time) and applies power in the opposite direction indicated by the sum to drive the lean to zero, as follows

```
integralTerm = -iGain * sumOfDifferences;
```

The output of the control algorithm at any point in time is the sum of these terms:

```
motorPower = proportionalTerm + integralTerm + differentialTerm;
```

Rather than implementing this algorithm directly, the BalanceBot example application uses the `PIDController` class from the RoboJDE class library. All the BalanceBot application has to do is specify the gain constants when constructing the `PIDController` and then call the `control()` method at regular intervals. The BalanceBot uses a high priority thread to run the PID controller while running the user interface at a lower priority. This allows the BalanceBot to display data on the LCD screen and take input from the thumbwheel without interfering with its ability to maintain its balance.

Building the BalanceBot

Parts List

Controller

- IntelliBrain controller with expansion board

Sensors

- 2 Fairchild QRB1134 infrared sensors with Molex connectors

Lego® Parts

- 4 1 x 10 plates
- 1 2 x 8 plate
- 6 2 x 10 plates
- 1 4 x 10 plate
- 2 2 x 4 bricks
- 6 1 x 6 beams
- 4 1 x 8 beams
- 12 1 x 10 beams
- 2 1 x 14 beams
- 10 1 x 16 beams
- 1 long (~10 inches) electric wire with bricks
- 4 whip antennas
- 2 9V mini motors
- 2 motorcycle wheels with tires

Miscellaneous

- 6 cell (2 x 3) AA battery holder with leads
- 6 AA batteries
- permanent double sided tape

Assembly Instructions

The BalanceBot chassis consists of an undercarriage topped by a platform on which the IntelliBrain controller is mounted, as shown in Figure 1. A 4 beam high wall extends up from both the front and rear edges of the platform. Six beams span between the two walls, forming an enclosure around the IntelliBrain controller. A battery box sits on top of the beams and encloses the battery pack. There is no top on the battery box.

Undercarriage

1. Attach 2 1 x 10 plates to the outer top rows of a 4 x 10 plate.
2. Attach the motors on top of either end of this assembly leaving 4 knobs spacing between the motors.
3. Attach 2 2 x 4 bricks to the top of a 2 x 8 plate.
4. Attach this assembly crosswise on the bottom of the 4 x 10 plate under the motors, such that the blocks extend equally forward and rear of the centerline of the axles.
5. Use double sided tape to attach QRB1134 sensors to the forward and rear ends of these blocks.
6. Cut the motor wire into two pieces and strip the cut ends of the wire so they can be attached to the motor ports on the IntelliBrain later.
7. Attach the motor wires to the top of the motors with the wire running toward the center of the robot.

Platform and Wheels

1. Attach the ends of 2 1 x 16 beams to the top ends of 2 2 x 10 plates, forming a rectangle with the plates on the bottom.
2. Attach 2 more 2 x 10 plates next to the previous two.
3. Attach 2 more 1 x 16 beams next to the previous two.
4. Attach 2 1 x 6 beams between the ends of the 1 x 16 beams.
5. Attach 2 1 x 14 beams next to the 1 x 16 beams.
6. Attach 2 2 x 10 plates to the top ends of the beams.
7. Attach 2 1 x 10 plates next to these plates.
8. Trim the whip antenna pieces at the top of the base.
9. Insert the antenna pieces in the mounting holes on the bottom of the IntelliBrain controller.
10. Attach the controller to the platform with the right edge of the controller next to a 1 x 10 plate. This will leave a gap at the left end for wires.
11. Attach the platform to the undercarriage such that it is centered between the motors and over the axles.
12. Feed the motor and sensor wires through the gap at the left of the controller.
13. Attach the front sensor (to the right of the IntelliBrain controller) to analog port 5.
14. Attach the rear sensor to analog port 4.
15. Attach the left motor to motor port 1 and the right motor to motor port 3.
16. Check that the motor jumper is set for power from the main battery (see IntelliBrain User Guide).
17. Attach the battery leads to the main battery header (see IntelliBrain User Guide).
18. Download this BalanceBot example application to the robot and press the START button.

19. Hold the robot about 1/2 inch over a light colored surface tipping it so the front is lower.
20. Note the rotation of the axles. The axles should rotate so the robot will move toward the lower side. If one or both axles spin in the wrong direction, switch power off and reverse the connection of the motor's leads to the motor port.
21. Attach two motorcycle wheels to the axles.

Battery Platform and Box

1. Build 2, 4-beam-high end walls by attaching 1 x 10 beams on top of each other.
2. Attach these to the front and rear edges of the platform.
3. Attach 6 1 x 16 beams between the left and right ends of the walls, leaving 4 knobs gap between the beams.
4. Arrange 2 1 x 10 beams and 2 1 x 6 beams to form an 8 x 10 rectangle.
5. Attach 4 1 x 8 beams on top of this.
6. Attach 2 1 x 10 beams on top and 2 1 x 6 beams to form the third row.
7. Slide the battery pack through the box so the lead wires run through the box, then attach the box to the robot, centered over the axles.
8. Set the battery pack in the box.

Calibration

The PID controller uses three gain constants and a set point to control how the robot responds to the feedback signal. The value of the gain constants is critical to the robot's ability to balance, its response to being bumped, and its nervousness (tendency to shake). The set point is determined by the thumbwheel position and accounts for differences in readings between the sensors that may be due to differences in the sensors or differences in the mounting positions (distance above the floor) of the sensors.

The value of the gain constants depends on specific characteristics of each robot. Variations in design, battery voltage, weight and motor friction may require the constants to be calibrated for your BalanceBot. A commented block of code in the `main()` method of the BalanceBot example provides an example of how to use the thumbwheel to calibrate one gain at a time. One procedure for calibrating the gains is as follows:

1. Modify the example application to set the `CALIBRATING` constant to `true` and the `iGain`, `dGain` and `setPoint` to `0.0f`. Then use the thumbwheel to calibrate the `pGain`.
2. Adjust the thumbwheel so the robot recovers from being tipped, but does not overreact, tipping even further the other way. The robot should wobble in a constant amplitude oscillation, but not respond wildly and fall over.
3. Modify the code to set the `pGain` value to approximately 2/3 the value determined in the previous step. The gain is displayed on the LCD screen. Then modify the calibration code so the thumbwheel calibrates the `dGain`.

4. Adjust the thumbwheel so the robot recovers smoothly from being tipped without overshooting or jittering once upright.
5. Modify the code to set the `dGain` to the value determined in the previous step and set the `CALIBRATING` constant to `false` to allow adjusting the `setPoint`.
6. Adjust the thumbwheel so the robot doesn't have a preference to run in one particular direction because of a differences in the sensor mounting positions and variations in the sensors themselves.
7. Modify the code to set the `setPoint` to the value determined in the previous step. Set the `CALIBRATING` constant to `true`, and modify the calibration code so the thumbwheel controls the `iGain`.
8. Adjust the thumbwheel so the robot fully recovers from being tipped. With the `iGain` set to zero the robot will recover to almost vertical but will tend to have a small tip which will cause it to move continuously forward or backward. Adjust the `iGain` so the robot stays in one place and does not oscillate badly.
9. Modify the code to set the `iGain` to the value determined in the previous step and set `CALIBRATING` to `false`.

Conclusion

The BalanceBot example demonstrates how a PID control algorithm can be used to keep an unstable robot upright using only two inexpensive sensors for feedback. By executing the control algorithm on a high priority thread, the task of keeping the robot balanced can be given priority over other tasks. The example application only uses a small portion of the IntelliBrain controller's CPU and memory resources, leaving plenty of computing power to experiment further by adding additional capabilities such as navigation and behavior based intelligence.