Using Multiple Sensors In An Obstacle Avoidance Algorithm For Lego Mindstorms Robots

Steve Pilling
Computer Science
Simpson College
701 North C Street Indianola, Iowa 50125
pilling@simpson.edu

Abstract

The overall goal of this project was to build and program an obstacle avoidance robot. The paper discusses the hardware and software used in the project. First we describe the contents of the Lego Mindstorms Robot kit. Then we discuss the major problems that were encountered over the course of the work with the kit and its components. In the software development phase we had a choice to use either ROBOLAB, the Lego software provided to program the robot, or leJOS, a firmware replacement that allows programming in Java. We discuss some of the advantages and disadvantages of each software environment. The paper outlines how basic Java programming structures can be used in combination with leJOS. Finally, the paper presents the current algorithm for obstacle avoidance.

Introduction

The objective of this project is to eventually program a Lego Mindstorm Robot to traverse a grid and avoid any obstacles while in search of the goal. Provided with the Lego Mindstorms Robot (Lego Mindstorms Robotics Invention System 2.0, from here on RIS) is ROBOLAB, the standard program for use in loading instructions on to the robot. After working for some time with ROBOLAB it was found that while program was very good, it would not be able to do everything necessary to reach the goal. To fix this problem, the leJOS was found. leJOS is a program that allows programming of the RIS using Java instead of the supplied software.

The Problem

The specific problem is to program a robot to search a six by six grid. This grid is to have randomly placed obstacles, whose position the robot will record and then avoid. The robot is to search the grid for a goal square which is colored black. When set to run, the robot will know its starting coordinates, but will need to find the goal square through searching. Upon finding this square the robot is to return to its starting position. The work toward this is being completed in separate modules. One of the most recent hindrances is the tracking of the robots coordinates. This is due to the clumsy configuration of sensors currently used to detect the robot crossing a line.

Hardware

In this project the RIS and supplied parts, which include the Robotics Control System (RCX), motors, light sensors, and bump sensors, have been employed. The RCX is the programmable hardware to which all other pieces of the kit can be attached. It has three "ports" for connecting sensors and three ports for connecting motors (Constructopedia, 4-5). Motors have the ability to turn in a forward or reverse direction which is controlled by the RCX. A light sensor uses a projected light to differentiate between colors. It returns a percentage value to the RCX based on the amount of light returned to the sensor. Bump sensors have a single button on it that when pressed sends a value of true or 100 percent to the RCX.

In order to solve problems and better help in obstacle avoidance, another sensor was purchased from a third-party vendor, Techno-Stuff.com. This sensor, called a Dual Infrared Proximity Detector, allows detection of objects to the left (returning a value of 22), the right (returning a value of 48), and directly in front (returning a value of 0) of the robot. If there are no objects ahead the sensor returns a value of seventy-five. The sensor can detect objects at a distance of around five inches through use of infrared light, and only requires the use of one port (Sevcik, np).

Figure 1: An RCX

Hardware Problems

Bump Sensors

The original arrangement of sensors attempted to use bump sensors for avoiding obstacles. These arrangements were ineffective in detecting obstacles because in order to be depressed the button had to touch a flat surface, parallel to the face of the sensor. If the sensor wasn't flat against the surface the button could not be touched by the obstacle due to the way the button is positioned in the sensor. Because of this many objects could not be detected, and would end up being pushed by the robot. One of the original designs required three bump sensors spaced an inch and a half apart attached pointing away from the front of the robot. These three sensors were to be used, along with two light sensors for following a line, to find an obstacle to either side or directly in front of the robot. The problem here was the RCX only has three sensor ports, and this design required five.


Figure 2: Bump Sensor

Due to the way in which the sensors were constructed, more than one sensor can be connected to each port on the RCX at one time. The drawback of this is that the values returned for that port are then the sum of the percentages returned by the sensors connected to that port. Consider a light sensor and a bump sensor together on the same port. If the light sensor is detecting white it will return a value around 35% and if the bump sensor is not being pressed then it will return false or 0% for a total of 35%. If the bump sensor is then pressed the value returned to the port will be 100%, but the program will not know whether the light sensor is detecting white or black. As you can see connecting two bump sensors on the same port would be useless for their current application. To solve these problems the infrared sensor was purchased reducing the number of sensors needed for the design at the time to three. At the present time no bump sensors are being employed in any robot sensor arrangements.

Light Sensors

The light sensors provided with the RIS have worked properly so far. The only problem encountered did not have so much to do with the sensor itself but rather with the environment in which the sensor was located. As could be expected, in different lighting environments the values returned by a light sensor for white and black changed. In one test area the reading for white was 35 in another it was 40. The problem came about because the instructions in the robot were to follow a black line on a white surface and thus needed to be able to differentiate between black and white. With the values changing, it was necessary to find these new values and reenter the newly edited instructions before running them in the new environment.

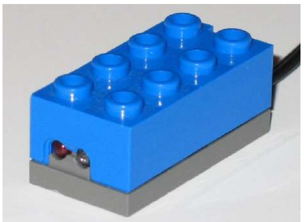

Figure 3: Light Sensor

Motors and Chassis

The largest problem caused by the motors provided with the RIS was the period of time they took to complete a turn to the left or right. Much time was spent in experimentation trying to determine the proper amount of time to make robot to turn 90 degrees. In the current set of instructions more time is required to make a left turn than a right turn. One cause of this may be due to the tendency of the robot to veer to the right, which could likely be a problem with the wheels or the connecting parts (wheel axels that may be bent).

Figure 4: Chassis

Another probable cause of the increased time to turn left could be the motor design. The experiments showed that when using another chassis, assembled in a different manner, the robot again veered to the right slightly. If the motor was not designed to have the same power moving forward as backward it could be causing the problem.

Figure 5: A Motor

When the motors are attached to the chassis, the axes coming out of the motors will be pointing in opposite directions. For the robot to move forward one motor axis would spin clockwise and the other counter-clockwise. This is seen as less likely cause because this seems to be one of the intended applications of the motors.

Additional Hardware

Up until this point most problems stemming from the non-availability of sensor ports have been solved by searching for ways to use fewer sensors to perform the same actions. This was a reason the infrared sensor was purchased. While this has worked so far, it is a noticeable problem for the future.

Figure 6: Dual Infrared Proximity Detector

At some point more than three sensors will be required to fulfill the objectives of the robot. One solution to this problem is the use of an active multiplexer. A multiplexer is used so that multiple sources can be connected to a single sensor port. The multiplexer being used currently was purchased from Mindsensors.com (Active Sensor Multiplexer v2, np).


Figure 7: Active Multiplexer

An active multiplexer is one that can support sensors that require power, such as light sensors and the infrared sensors. Another type of multiplexer offered from Mindstorms.com, a passive one, is made to be used with sensors that do not require extra power, such as bump sensors. While using the active multiplexer, only one of the sources can be read from at a given time. The robot can choose which sensor to listen to, but it requires time to change between them.

Software

The program chosen to program the RCX was leJOS, pronounced “Ley-J-oss” or “Lay-hoss” (leJOS, np). The current version is leJOS 2.1.0. leJOS is an open source program that is freely available for download. It allows the use of Java to program the robot by replacing the firmware on the RCX (leJOS, np). Firmware is the software installed on the RCX that tells it how to interpret the stored instructions. The instructions are written in Java using extra classes developed to control the RIS specifically. The compiler program used to convert the Java code to an RCX understandable file is not extremely user friendly, but this was remedied by using an extension for the Java compiler BlueJ. This extension, called the BlueJ Mindstorms Tool (BlueJ Mindstorms Tool, na), allows the user to compile and then download the instructions to the RCX using BlueJ rather than using a command prompt. BlueJ is also available free online (BlueJ, np).

These are not the only pieces of software available. The reason leJOS has been selected to program the instructions is because it uses Java, the base programming language taught at Simpson College. BlueJ has been chosen because it is the Java compiler currently used by Simpson College. It is very possible that there are other equally good or perhaps even better choices.

LeJOS Versus ROBOLAB

On the way toward the objective the instructions went through several stages of development. The first instructions were written using ROBOLAB from Lego. Eventually leJOS was found and employed. One advantage of ROBOLAB over leJOS is the ability to load more than one set of instructions into the RCX at a time. The RIS with original firmware allows the user to switch between up to five programs by pressing a button on the RCX. leJOS firmware is only capable of loading one program at a time. As far as ease of use is concerned, ROBOLAB is fairly good straight out of the box. It uses a drag and drop GUI. leJOS does not come with a GUI, but it offers a greater amount of controls. This plus the fact that leJOS can be combined with BlueJ, which does use a GUI, easily balances out this loss.

Both ROBOLAB and leJOS software can use sensor listeners, but leJOS again offers greater control. One of the many examples of increased control leJOS has is that the programmer can also use listeners for the buttons on the robot, whereas the original firmware does not give the programmer access to these buttons directly.

Both pieces of software are capable of writing instructions that can keep variables, but leJOS has the ability to use standard Java arrays to store values. The ability to use arrays, in combination with the coordinates of the robot, will be useful in recording the location of any obstacles encountered.

During normal runs of any instructions written with either program, any data recorded during a run is lost once it is finished. leJOS allows access to what it calls a persistent memory area which can keep data between runs and will remain as long as the instructions are not reloaded.

leJOS has some other interesting capabilities, such as communicating with a computer or even with another RIS using the same infrared port used to load instructions. This however is not used in the current project.

Basic Instructions

Most of the instructions for directing the robot contain sensor listeners. These are written very similarly to Java's action listeners. Listeners are bits of code set to run on the occurrence of a particular event. In Java GUI's this may be the clicking of a button, and in leJOS this may be the changing of a sensor value.

When the value being detected by a sensor is changed it runs a method in response. Listeners have three parameters: the sensor detecting the change, the value returned the last time the value changed, and the new value that has just been detected. When a sensor listener is triggered any other instructions being executed at that time are interrupted and the listener instructions are run. The following code is a basic setup for a light sensor, returning a percent value, and its sensor listener for the first sensor port, S1.

```
Sensor.S1.setTypeAndMode (SENSOR_TYPE_LIGHT,
SENSOR_MODE_PCT);
Sensor.S1.activate ();
Sensor.S1.addSensorListener (new SensorListener()
{
	public void stateChanged (Sensor src, int oldValue, int newValue)
	{
		// code to be run upon sensor change
	}
});
```

In this setup each sensor port will have its own sensor listener. A single sensor listener may be used for multiple sensors. This would be accomplished in the same manner as it would be for an action listener in Java.

Three types of programs can be made: those that cause the robot to run through a set of instructions once then stop, those that run a predetermined number of times then stop, and those that repeat the instructions until some event happens then stop. This was a problem in early versions of the instructions. The instructions, built of sensor listeners, were intended to run forever but would run once and stop. It was found that the problem can be solved by embedding the instructions into an infinite loop that will keep them running until the robot is shut off.

Current Programs

As of yet the program has been developed in modules. One was to program the robot to be able to avoid obstacles. Another was to keep track of the robot's coordinates and the direction it was facing in a grid.

The first module consisted of the robot following a line, then upon encountering an obstacle turning right or left to avoid it (see Figure 9). Two light sensors were used to track the line. If one of the light sensors detected white while following the black line the robot would turn until the sensor read black again. The robot would determine the best way to avoid any obstacles by using the infrared sensor to determine if the obstacle was more to the left (the robot would turn right), the right (the robot would turn left), or center (the robot would turn 180 degrees and go back in the direction it came from).

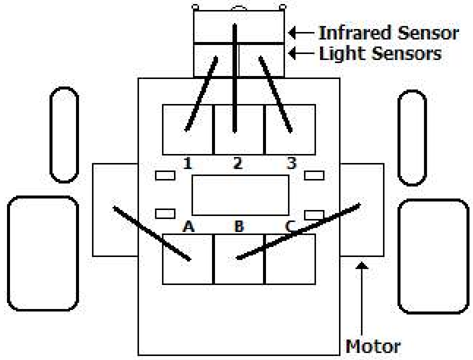

Figure 8: Physical layout of obstacle avoidance robot

The following figure shows the algorithm implemented in this module. The values 15, 35, and 65 are used in the listener for S2 in case the values returned by the infrared sensor are not exactly what were listed by the manufacturer as expected.

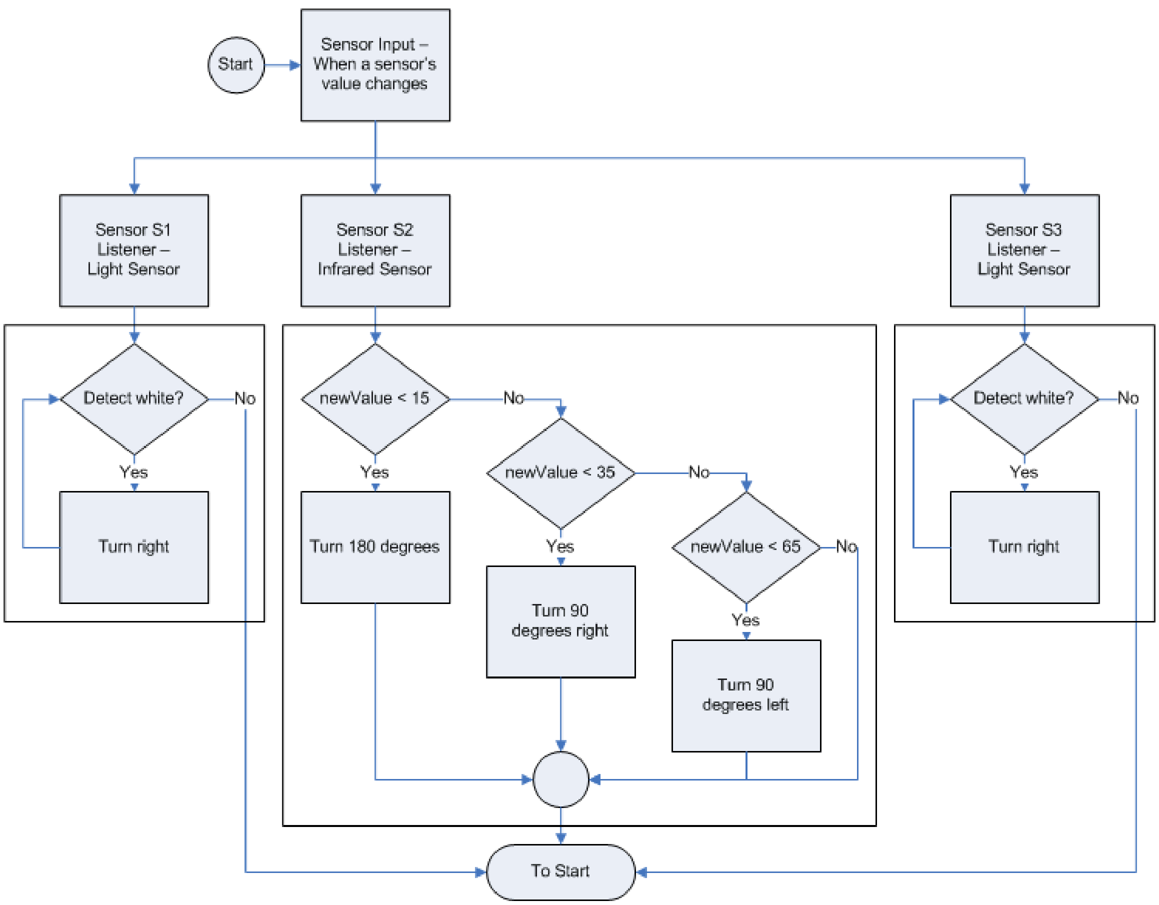

Figure 9: Obstacle avoidance algorithm

The second module was a challenge because of the difficulty in determining if the robot has crossed a line. The latest configuration of the second module uses one light sensor, pointing downward, that continually checks to see if the robot crosses a line. The check is accomplished by comparing the old value to the new value. If the old value is white and the new is black the robot concludes that it must have crossed a line. The problem here was that the sensor would sometimes sense more than one line where there was only one, and other times would detect no line where there was a line. Again this is partially due to the environment in which the robot is running, bad lighting or an uneven floor. Also the coordinates were stored as integer numbers which only told which square the robot was in at the time. While this number helps it would be better if even ½ squares or squares and lines could be kept track of separately. In the original configuration, if one of the obstacles were placed on a line then the obstacle would be shown as blocking both whole squares on either side of the edge rather than just the edge.

Future Objectives

Because of leJOS's ability to use arrays, the position of the obstacles and open squares should be easy to track. The intent is to eventually combine the modules that have been developed into one set of instructions so that the robot will be able to follow the grid lines, detect, record, and avoid obstacles, while tracking its position on the grid.

Acknowledgements

The Department of Computer Science at Simpson College supported this project by providing the necessary hardware. Many thanks go to Dr. Lydia Sinapova and Svet Sinapov for their helpful comments and valuable advice.

References

Active Sensor Multiplexer v2. Mindsensors.com. 01 Dec. 2005 <http://www.mindsensors.com/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=20>.

BlueJ. 01 Dec. 2005 <http://www.bluej.org/>.

BlueJ Mindstorms Tool. University of Paderborn. 01 Dec. 2005 <http://ddi.uni-paderborn.de/mindstormstools/bjmt>.

Constructopedia. LEGO Group, The. 1999.

leJOS. 01 Dec. 2005 <http://lejos.sourceforge.net>.

Sevcik, Pete. "Dual IR Proximity Detector." Techno-Stuff. 01 Dec. 2005 <http://www.techno-stuff.com/DIRPD-T.htm>.