**Computers in Engineering**
**Lab #2**
**Maze Treasure Searching**

## Objective

The objective of this lab project is to program the robot to tracking the maze. The goal is to pick up all the silver treasure in the maze. The speed of tracking is important here so that you can be the first one to discover the treasure.

## Introduction

The robots in this lab project are constructed using Lego MindStorms, a special Lego kit for amateur robotics. The robot in lab have already been designed and built for you. These robots are controlled by an *RCX module*, a big Lego brick with an embedded Hitachi H8 microcontroller, along with memory and a simple, multitasking operating system. The rover robot that you will program is configured with two light sensors that you will use to navigate the track. Your lab station will include a Linux-based host computer on which you will compile programs and download them to the RCX module via an infrared link.

There are several programming languages and environments available for developing RCX programs. The language that will be used in lab is called NQC, which stands for **N**ot **Q**uite **C**. Although NQC isn't exact, or complete, C syntax, it is very similar. See the section on Programming in NQC for more information. Additional details about NCQ and its associated API can be found in the NQC Programmers Guide. Information about the MindStorms robots and a number of NQC programming examples can be found in the text **Dave Baum's Definitive Guide to Lego Mindstorms, Second Edition**. Several copies of this book are on reserve in the Engineering Library.

NQC source programs are standard text files, just like any C program, except that they should have a filename extension of ".**nqc**" instead of ".c". As such, they can be written outside of lab using your favorite text editor. However, you will not be able to compile or test these programs until you are in the lab. You can save your source files in your CSS account which will be accessible in the lab.

## Description

The rover robot will use light sensor(s) (i.e. LED reflectance sensor) to follow a line around a track in either direction (clockwise or counterclockwise). At the silver square (the treasure), the rover should turn around then going for the next treasure. At the intersection (you need a design to detect it), the rover should always make the left turn. If rover running to a dead end, it needs to turn around and keep searching the treasure. If a sharp corner is reached, you need to decide to make the right turn or left turn. Rover will stop after all treasure has been found. In the diagram below, there are three silver targets the rover needs to reach.

**Program Design**

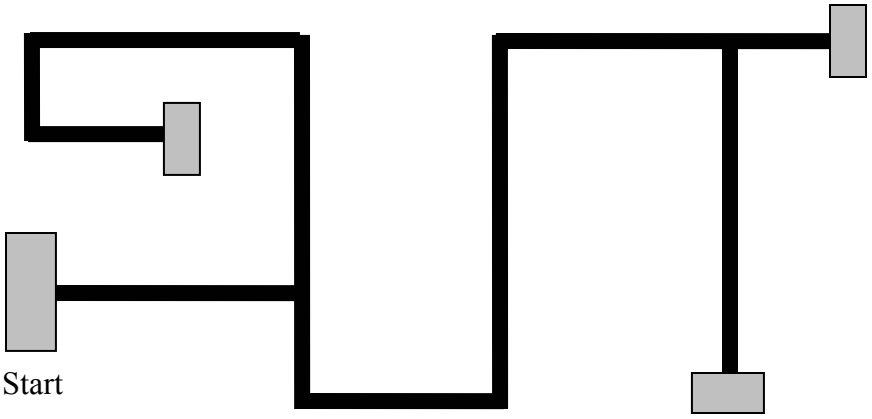


**Figure 1**: The track that the rover robot will follow.

There are various methods for tracking lines. The choice of which method to use is up to the programmer. Each rover is equipped with two touch sensors and two light sensors. The light sensors are attached to the front of the rover, centered across its width, and separated by a distance just slightly greater than the width of the track. The light sensors face downward so that the light sensing LEDs are just above the track surface. These sensors use photo detectors to measure the amount of light reflected back from the track surface. Differing levels of reflectivity measured by the sensor can be used to differentiate dark versus light areas on the track, reflective versus non-reflective surface characteristics, and, to a limited extent, even different colors. The light sensor produces an integer value between 0 and 100, where 0 indicates the darkest (least reflective) surface and 100 indicates the lightest (most reflective) one. You will need to empirically determine the appropriate sensor values corresponding to white surface, black surface, green surface, and silver markers on the track.

**Sensor Mode**

*Raw Mode*: integer between 0 and 1023. It's the actual reading of the sensor's voltage level, converted to a digital number from 0 to 1023(inclusive). Note: The voltage levels are differed by different colors.
Mode constant: SENSOR_MODE_RAW

*Boolean Mode*: Sensor's processed value is set to the Boolean value. It's the default mode for touch sensor. For sensors other than the touch sensor, a proper slope parameter is often essential for meaningful Boolean values. The rule is: raw value > 562, Boolean = 0; raw value < 460, Boolean = 1; otherwise Boolean unchanged. This conversion method is mentioned later in hysteresis.
Another conversion method may be used by add nonzero slope. Each time the sensor is read (every 3ms), its raw value is compared to the previous raw value. If the

absolute value of the difference is less than the slope parameter, the Boolean value remains unchanged, otherwise, the Boolean value will be set to indicate whether the raw value increased (Boolean = 0) or decreased (Boolean = 1). Special cases exist at the extremes of the raw values (smallest, and greatest). The conversion is demonstrated in the following example.
Mode constant: SENSOR_MODE_BOOL
Example: SetSensorMode( SENSOR_2, SENSOR_MODE_BOOL + 10 )
This configures sensor 2 to be a light sensor in Boolean mode with a slope of 10. So change > 10, value read by sensor is 0; change < -10, value read is 1; current > (1023-10), value read is 0; current < 10, value read is 1.
Note: change = (current raw value - previous value); current = current raw value.

*Percentage mode*: the raw value is converted into a value between 0 and 100 using formula valuePercentage = rawValue/1023*100. It's default for light sensor. (As what we used in lab1)
Mode constant: SENSOR_MODE_PERCENT

To set the sensor mode using the command line:
SetSensorMode(sensor, mode);
"mode" is the constant system variables described above in each mode.
"sensor" is the sensors you want to defined.

**Tracking Algorithm**

As shown in figure 1 above, the track consists of one solid black line on a white background.

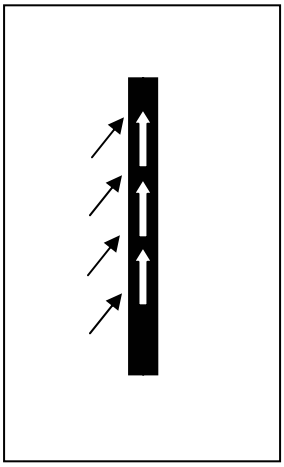

Figure 2 Tracking Algorithm

The algorithm for following the track attempts to move the robot along boundary while making course corrections to keep the light sensor(s) positioned along the boundary of the track near the white to black transition. One method would be left sensor tracks the white on the left of black track, the right sensor is tracking the black track. When left

track hits black, make the rover adjust the direction by turn left. If right sensor hit white, adjusting the robot by turn right.

Instead of using the control structures describing above, NQC provides another mechanism, called even monitoring, which can also solves this problem. There three steps to use events: configuring the event, monitoring the event, and handling the event when it occurs. Events are configured using the function SetEvent(number, source, type), where number specifies which event you are configuring(0-15), source is the thing that the event should read(i.e. sensor), and type is the kind of condition that will trigger the event. For example, to configure event 0 to be triggered when a touch sensor attached to sensor port 1 is pressed, you could use the following statement:

```
SetEvent(0, SENSOR_1, EVENT_TYPE_PRESSED);
```

The 2nd step, monitor: we use monitor statement, which looks a little bit like a while loop. Instead of a condition, the argument to a monitor statement determines which events are going to be monitored inside the body of the statement. Events are specified using masks instead of event number. The macro EVENT_MASK converts an event number into a mask, and multiple event masks can be combined by adding them together. For example, the following code monitors events 1 and 2:

```
Monitor(EVENT_MASK(1) +  EVENT_MASK(2))
{
    // the body of the monitor statement
}
```

An example program that uses events is in the Appendix sample2.

The following outlines the basic flow of the program for the rover robot:

- The robot will traverse the track using the black track as a guide until the intersection is encountered.
- Once the intersection marker is encountered, the rover will have to make a left turn to follow the approach track to silver treasure.
- The robot will stop and turn around.
- The robot will reverse orientation (turn around) and head back toward the next silver treasure.
- The robot will stop after it reached last (3rd) treasure is found.

**NQC Programming**

**The NQC Main task**

The main task of a NQC program starts with the *task* keyword, and looks similar to the main function of a C program. Also, as in C, the *main* task is the entry point of the program when it executes. In NQC, functions to be called by the main task have headers that begin with the keyword *void* ( NQC functions cannot return a value).

```
task main( ) {
```

```
  // Trace around the track
  traverse_track ( );
 // Enter approach
 turnLeft();

  traverse_track ( );
          …
          …
          …
  traverse_track( );

  Exit;
}

//Function definitions
void traverse_track( ) {
   // Function code
}
void turnLeft() {
// Function code
}
void turn_around();
void turnRight() {
// Function code
}
```

NQC functions can have parameters, like C functions. Parameter passage can be by-value or by-reference. See the NQC Programmers Guide for further details.

In addition to the main task, an NQC program can specify additional tasks with headers beginning with the keyword *task.* These tasks can be started by the main task and will execute concurrently with the main task and each other. This allows an NQC program to carry out several different activities simultaneously—e.g. monitoring a sensor with one task while controlling a motor with another. Tasks can be started and stopped using **start** *taskname* or **stop** *taskname* in the main task.

**Reading Information from the Light Sensor**

Each sensor in the robot must first be initialized before it can be used. This allows NQC and the CX operating system to provide easily accessible values and ranges for the instruments rather than just raw data. Sensors are initialized with the SetSensor( ) function.

```
SetSensor( SENSOR_X, SENSOR_LIGHT );
```

Since SENSOR X can change from SENSOR 1, SENSOR 2 and SENSOR 3 based on how the robot is built, we will use a #define to make things easier. First, let us assume that the light sensor is attached to SENSOR 1.

```
#define EYE SENSOR_1

SetSensor( EYE, SENSOR_LIGHT );
```

Then EYE can be used to return the value of light currently measured by the sensor. For example:

```
if ( EYE < 14 ) {
...
}

while( EYE >= 57 )  {
...
}
```

The sample program posted on the class webpage can be used to display the light sensor values for various regions of the track on the LCD display on the top of the RCX brick. Simply download the program and execute it. Then place beam of the eye over the region you wish to measure and observe the value displayed on the LCD.

**Using the Motors**

As above, let us also #define the left and right motors such that our program can be easily modified if the robot is reconfigured. The motors are designated by the output ports they are connected to (OUT_A, OUT_B, OUT_C). In this lab the robots are configured as follows

```
#define LEFTMOTOR OUT_A
#define RIGHTMOTOR OUT_C
```

There are three things to set to operate the motor. First is the power of the motor (How fast it will turn.) Power values range from 0 to 7. It is recommended to turn at a slower speed ( 1 to 3 ) and go straight at a higher speed ( 5 to 7 ). To set the power of the motors to 4:

```
SetPower( RIGHTMOTOR, 4 );
SetPower( LEFTMOTOR, 4 );
```

However NQC will also let you combine the statements into one:

```
SetPower( RIGHTMOTOR + LEFTMOTOR, 4);
```

Next, the motor direction must be set. This is done via Fwd( ) and Rev( ) functions:

```
Fwd( RIGHTMOTOR + LEFTMOTOR );
Rev( RIGHTMOTOR + LEFTMOTOR );
```

To turn the robot, simply have one motor go in one direction and the other motor in the other direction.

```
Fwd( RIGHTMOTOR );
Rev( LEFTMOTOR );
```

The motors must also be turned on and off.

```
On( RIGHTMOTOR + LEFTMOTOR );
Off( RIGHTMOTOR + LEFTMOTOR );
```

The motors do not need to be constantly set. Once a parameter of the motor is set, it will stay that way regardless of what other parameters are set. Other functions combine the Fwd, Rev and On functions:

```
OnFwd( RIGHTMOTOR + LEFTMOTOR );  /* combines On and Fwd */
OnRev( RIGHTMOTOR + LEFTMOTOR );  /* combines On and Rev  */
```

See Dave Baum's Definitive Guide to Lego MindStorms for further examples.

**Debugging Techniques**

When programming in C, we usually use print statements to track our program. Here we can make the robot play sound to track the program process.

```
PlaySound( sound );          //play a system sound
PlayTone( freq, duration )  //play a note of given frequency and duration
ClearSound()                 //clear any buffered sounds
```
//sound constant: SOUND_CLICK, SOUND_DOUBLE_BEEP, SOUND_DOWN, SOUND_UP, SOUND_LOW_BEEP, SOUND_FAST_UP. These will make the robot to play different kinds of sound.

**Decision making using hysteresis**

It's often important to keep the system from reacting too quickly. If the system reacts too quickly, you may end up with oscillation movements of the rover. To avoid problems with oscillations, decisions can be made using hysteresis. This is demonstrating in Figure 3 shown below.

Assume you are initially in State1. As V increases, the state remains unchanged until the threshold V2 is reached. When V=V2, the state changes from State1 to State2. Now assume that you are in State2 and V begins to decrease. You stay in State 2 until the threshold V1 is reached. When V=V1, the state changes from State2→State1. Thus, when V is between the V1 and V2 thresholds, you can be in either State1 or State2.

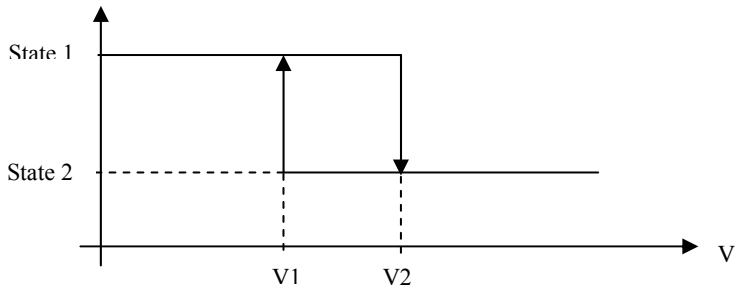

Figure 3

**Lab Procedure**
**SETUP:**

The host computers in the lab use the Red Hat Linux operating system, which is very similar to the HP-UX environment used in CSS computer labs. You will be able to log in using your normal login ID and password and will have access to your directories and files. It is recommended that you create a folder in your home directory to store your NQC files.

**PRE-LAB:**

- Develop the complete pseudo code for the rover robot program and develop your initial version of the NQC program to implement this pseudo code. Note that certain values—e.g. the light sensor values corresponding to various regions of the track surface—will have to be determined empirically once you get into the lab. So use #defines to specify these values in your program.
- Make a list of all values that you will need to determine or tune empirically in the lab and describe the process via which you intend to determine or tune these values.
- Be prepared to give the lab instructor the following items when you arrive for your first lab session:
    - A copy of your pseudo code
    - A printout of your first iteration of the NQC program for the rover robot.
    - A lab procedures: including the data collection table: the list of values/parameters to be empirically determined or tuned in the lab and the process by which you will do this; and the detailed plan you will follow in the lab time.
- **IMPORTANT: If you have not completed the items described in the above bullet prior to coming to lab, you will be asked to leave and work on it elsewhere. It is unfair to your classmates to expect the TA to devote time to you when you have not completed the lab preparation. The lab sessions will be brief and are intended to test and verify your work done prior to the lab, the sessions are not intended for writing the pre-lab with the assistance of the TA. See the TA before the lab with any questions. The equipment used in the lab will be available for inspection as well.**

**IN LAB:**

- Downloaded from class website, download it to the rover by typing "nqc -firmware firmware_fileName"
- Save the sample program 1 from the appendix into a file named sample1.nqc
- To compile and download the program, open a shell and "cd" into the directory containing the sample1.nqc program. Then type "nqc -d sample1.nqc". This will compile your program and automatically download it to the RCX of your rover robot. Since the download takes place over the infrared link, your rover robot will need to be in close proximity to the infrared transceiver module on the host computer.
- Record the average values (out of 10 samples) of the colors: white, black, and silver.
- Using these values you should be able to finish the missing parts of the rover track program you wrote in preparation for this lab.
- Compile, download, and test the rover robot track program you wrote for this lab project, until you have it working correctly
- **Once you have proper operation, have the T.A. verify the operation of the rover track robot. T.A is going to record the time your rover finish the work. Out of a total of 100 points. You will be awarded 15 points for successfully completing the course. Another 15 points will be awarded for speed. The fastest team will get 15 points and the slowest team will get zero points.**

**Post-lab**

In addition to the general lab report format guidelines given on the class web page, your lab report needs to include the following elements:

- Briefly discuss the tracking scheme you designed; how you identified intersection and track corner and dead end.

**Appendix**

**Sample 1.** Light Sensory Display Program Listing

```
task main( ) {
  SetSensor( SENSOR_1, SENSOR_LIGHT );   //set up 2 light sensors
  SetSensor( SENSOR_3, SENSOR_LIGHT );
  while( true ) SelectDisplay( DISPLAY_SENSOR_1 );
}
```

**Sample 2.** The program expects a touch sensor to be attached to sensor port 1, and a motor to be attached to output A. The program begins by setting up the sensor,

configuring an event to trigger whenever the sensor is pressed, and turning on the motor. The program then enters a while loop that will repeat forever. Within this loop, the program plays alternating low and high notes while monitoring the event. When the event is triggered, the program will interrupt what it is doing to run the catch clause, which calls ClearSound() to turn off whatever sound is currently playing and then reserves the direction of the monitor.

```
#define MY_EVENT 1

task main()
{
	SetSensor(SENSOR_1, SENSOR_TOUCH);
	SetEvent(MY_EVENT, SENSOR_1, EVENT_TYPE_PRESSED);

	On(OUT_A);
	While(true)
	{
		monitor(EVENT_MASK(MY_EVENT))
		{
			while(true)
			{
				PlayTone(440, 50); //
				Wait(50);
				PlayTone(880, 50); //
				Wait(50);
			}
		}
		catch
		{
			ClearSound();
			Toggle(OUT_A);
		}
	}
}
```

**References**

More information about the Lego MindStorm kits can be found at:

http://www.legomindstorms.com

More information about NQC can be found at:

http://bricxcc.sourceforge.net/nqc/

The NQC Programmers guide can be found at:

http://www.engineering.uiowa.edu/~cie/Labs/NQC_Guide.pdf

Created by
Fan Yang

*March 10, 2006.*