

# Computers in Engineering

## Lab #1

### Multi-Agent Cooperation

#### Objective

The objective of this lab project is to program two robots to cooperate in performing a simple task. The task involves a rover robot which moves around an oval track first receiving a foam ball from a second robot equipped with a robotic arm and then delivering the ball back to be retrieved from the “arm robot”. To simplify your effort, we have programmed the arm robot for you. You will be responsible for programming a pre-built rover robot to navigate a path around a track, detect the proper stopping points for ball pick-up and drop-off actions, and communicate with the arm robot to carry out drop-off and pick-up operations.

#### Introduction

The robots in this lab project are constructed using Lego MindStorms, a special Lego kit for amateur robotics. The robots 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 Not Quite 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 NQC 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.

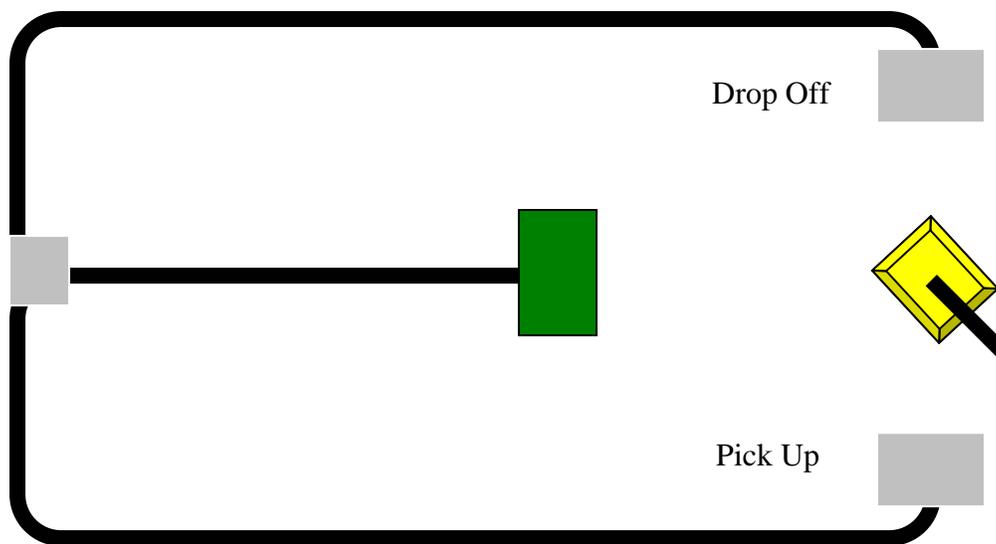
NQC source programs are standard text files, just like any C program, except that they should have a filename extension of “.nqc” instead of “.c”. As such, they can be written outside of lab using your favorite text editor. However, you will not be able to compile or test these programs until you are 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 rover should make a left turn and follow the portion of track leading to the robot arm. The rover should continue to travel the track toward the robot arm until the rover reaches a green rectangle indicating the pick-up location. Once the rover has come to rest at the

pick-up location, the rover should send the drop-off message to the robot arm via the infrared link to begin the pick-up operation. This will cause the arm robot to deposit a foam ball onto the rover. When the arm robot notifies the rover (via the infrared link) that the operation is complete, the rover robot will then turn around and travel down the approach track to the reflective marking indicating the intersection with the main oval track. A proper turn should be executed so that the rover travels on the side of the main oval track opposite to the side the rover started on. The rover will follow the track around to the other side of the track where a second green marker has been placed indicating the approach to the drop-off location. Once the rover reaches the drop off location, the rover should send the pick-up message to the robot arm robot as above to facilitate the unloading of the foam ball.

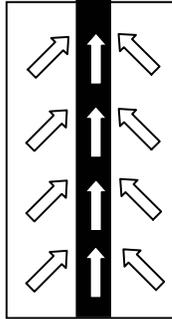
### Program Design



**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.

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



**Figure 2:** The 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.

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 reflective intersection marker is encountered.
- Once the intersection marker is encountered, the rover will have to make a left turn to follow the approach track to the pick-up location.
- The robot will stop and send a “pick-up” message to the arm robot.
- The robot will wait for a message back from the arm robot, indicating that the arm has placed a foam ball on the rover robot.
- The robot will reverse orientation (turn around) and head back toward the intersection marker.
- The robot will then turn left to proceed to the drop-off location.
- The robot will follow the bottom half oval track to the “drop-off” location.
- The robot will stop at the “drop-off” location and send a “drop-off” message to the arm robot.
- The robot will wait for a signal back from the arm robot indicating that it has removed the foam ball from the rover.

## **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();
// Talk to the arm for pickup
traverse_track ( );

communicate_to_arm( 1 );
// Turn the robot around
turn_around();
// Trace around the track
traverse_track ( );
// Talk to the arm for delivery
communicate_to_arm( 2 );

Exit;
}

//Function definitions
void traverse_track( ) {
    // Function code
}
void turnLeft() {
// Function code
}
void communicate_to_arm(int message_number) {
// Function code
}
void turn_around();

```

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.

## **Infrared Communication**

The following functions are used for infrared communication:

```
#define SIGNAL_TRACKER_PICK_UP 1 //signal arm to pick up ball  
#define SIGNAL_TRACKER_DROP_OFF 2 //signal arm to drop off ball  
#define SIGNAL_ARM_DELIVERED 3 //signal rover ball is dropped  
#define SIGNAL_ARM_RECEIVED 4 //signal rover ball is picked up
```

```
SendMessage( unsigned char ); // argument is 1 to 255  
ClearMessage( );  
Message( ); // Default return (no message) is 0.
```

To communicate to the arm:

```
SendMessage( SIGNAL_TRACKER_PICK_UP );  
SendMessage( SIGNAL_TRACKER_DROP_OFF );
```

To receive a message from the arm:

```
ClearMessage( ); // clears out any previous message  
until ( Message( ) != 0 ); // waits for new message to arrive
```

Message() will stay at its current value until it is cleared via ClearMessage( ), so it can be used directly to check the incoming signal. For example:

```
if ( Message() != SIGNAL_ARM_RECEIVED ) {  
    // Your code goes here  
}  
  
while ( Message() != SIGNAL_ARM_DELIVERED ) {  
    // Your code goes here  
}
```

The specific values of the SIGNALs above will be given to you in lab. They can easily be set-up with the #define.

## 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:**

- 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.

### **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 advantages and disadvantages of a light-based tracking system rather than a mechanical tracking system—e.g. with respect to obstacle avoidance.
- How would you modify your NQC Rover program to incorporate the use of the touch sensors in addition to the light sensor?

### **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.** Rover robot using Touch Sensors (**You may ignore this part for your lab, since you are not using bumper, but you can use this for learning the syntax**)

```

#define LEFT_MOTOR OUT_A
#define RIGHT_MOTOR OUT_C
#define NORMAL_SPEED 7
#define TURN_SPEED 3
#define CLOCKWISE 1
#define COUNTERCLOCKWISE 0

task main() {

    // Start the task
    start check_bumpers;

    // Start Roving Forward
    SetPower( RIGHT_MOTOR + LEFT_MOTOR, NORMAL_SPEED);
    OnFwd( RIGHT_MOTOR + LEFT_MOTOR);
}

task check_bumpers( ) {
    while (true) {
        if (RIGHT BUMPER == 1) {
            // back up, turn left and move forward
            OnRev( RIGHT_MOTOR + LEFT_MOTOR);
            Wait(100);
            OnFwd( RIGHT_MOTOR);
            OnRev( LEFT_MOTOR);
            Wait(100);
            OnFwd( RIGHT_MOTOR + LEFT_MOTOR);
        }
        if (LEFT BUMPER == 1) {
            // back up, turn right and move forward
            OnRev( RIGHT_MOTOR + LEFT_MOTOR);
            Wait(100);
            OnFwd( LEFT_MOTOR);
            OnRev( RIGHT_MOTOR);
            Wait(100);
            OnFwd( RIGHT_MOTOR + LEFT_MOTOR);
        }
    }
}
}

```

## 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](http://www.engineering.uiowa.edu/~cie/Labs/NQC_Guide.pdf)

Updated by  
Fan Yang

*March 1, 2006.*