ECSE 211- Lab Report 1

Group 34

Rana Muaddi 260163563

Ionut Georgian Ciobanu 260242300

ECSE 211 lab report 1

1. a)
Touch sensor: values of 0 and 1; After pressing the button halfway the reading becomes 1.

Sound sensor(db): background noise %17; whispering %30; normal speech %70; whistling %100; clapping %100
Sound sensor (dBa): background noise %3; whispering %10; normal speech %20; whistling %100; clapping %30
We can't comment on the precision as we do not have another, more reliable, reading.

Measurements were taken about 50 cm away from the sensor.
The Sound Sensor can measure sound pressure levels up to 90 dB – about the level of a lawnmower (from the User Guide).

Light sensor (reflected light): blocking with finger: 64%, mirror: 90%; white background 66%; black background 40%
Light sensor (ambient light): ceiling 51%; wall 21%; blocking with finger 2%
We can't comment on the precision as we do not have another, more reliable, reading.
The light sensor allows the attached light source to be turned on or off.

Motor rotation
The built-in Rotation Sensor measures the Motor rotations in degrees (accuracy of +/- one degree) or full rotations (from the User Manual).
We can get the motor rotation in either rotations or degrees. For the purposes of this class we're probably going to use the degrees reading more, as it provides more resolution. From our tests, one full rotation was indicated correctly, as well as one degree. Our measurement for rotation was done using a protractor, so we could not get very reliable data.

b)
Ultrasound sensor: min 5 cm ; max 255 cm. Theoretical precision is +/-3 cm. For distances greater than 15 cm our data was precise within +/- 2 cm (relative error from 13.3% for small distances to 0% for greater distances, with a standard deviation of .7).

Measurement using a book			
Real dist [cm]	**Ultrasonic sensor [cm]**	**Absolute error [cm]**	**Relative error [%]**
15	17	-2	-13.33
30	30	0	0
45	44	1	2.22
60	59	1	1.67
75	75	0	0
90	89	1	1.11
100	100	0	0

Table 1: Book data

Measurement using a wall			
Real dist [cm]	**Ultrasonic sensor [cm]**	**Absolute error [cm]**	**Relative error [%]**
30	30	0	0
60	59	1	1.67
90	89	1	1.11
120	120	0	0
150	150	0	0
180	181	-1	-0.56
210	211	-1	-0.48
240	241	-1	-0.42
254	254	0	0

Table 2: Wall data

The NXT manual indicates that:

Large sized objects with hard surfaces return the best readings. Objects that are curved or very small or thin can be difficult to detect. Two or more sensors operating in the same room may interfere.

We have confirmed this through experiments, by trying clothing, books, bent sheets of paper, the ball we were provided with, coins, some Lego parts, etc.

The smallest part we could measure was a 1 cm Lego part, but not very reliable. Anything greater or roughly the same size as the distance between the two emitters of the ultrasonic sensor will be measured relatively precise if it is well aligned with the sensor.

Please see the tables 1 and 2 for accuracy information. On average, the relative error was 0.47% for the wall and 2.62% for the book.

2.a)

If the right button is pressed, the robot displays "Button 1 pushed" on the screen and the robot starts moving forward.

If the left button is pressed, the robot displays “Button 2 pushed” on the screen and the robot starts moving backwards.

If the middle (orange) button is pressed, the robot displays “Button 3 pushed” on he screen and if the robot was moving then it stops moving, if the robot was not moving nothing happens.

b)

```
//Global variables used


//Mutex used to control the Drive function

mutex shouldMove ;


//Variable used to store the last good reading (!=255) from the ultrasonic sensor

int dist = 0;


//Bool variable used to help stop anytime.

bool wantMove;




//Turns left in two steps, the number of degrees passed

void TurnLeft(int degrees)
```

```
{
    if (wantMove){
        RotateMotor(OUT_A, 75, degrees);
        RotateMotor(OUT_C, -75, degrees);
    }
}

//Turns right in two steps, the number of degrees passed
void TurnRight(int degrees)
{
 if (wantMove){
     RotateMotor(OUT_A, -75, degrees);
     RotateMotor(OUT_C, 75, degrees);
 }
}

//Move forward for the specified amount of time
//Ideally the argument should be more than 100ms,
//otherwise it will not be very precise
void GoForward(int miliseconds)
{
     long startTime = CurrentTick();

          //This loop is used for timing. For performance reasons we only check if the counter
          //exceeded the value every 100 ms
     while ( (CurrentTick() < startTime + miliseconds) && wantMove)
     {
```

```
    OnFwd(OUT_AC, 75);

    Wait(100);

    };



    Off(OUT_AC);

    return;

}


//Move backwards for the specified amount of time

//Ideally the argument should be more than 100ms,

//otherwise it will not be very precise

void GoBackward(int miliseconds)

{

    long startTime = CurrentTick();


    OnRev(OUT_AC, 75);

        //This loop is used for timing. For performance reasons we only check if the counter

        //exceeded the value every 100 ms

    while ( (CurrentTick() < startTime + miliseconds) && wantMove)

   {

      Wait(10);


   };



    Off(OUT_AC);
    return;
}

//Print sensor information to the screen
task PrintInfo()
{
 while (true){
    ClearScreen();

    TextOut(0, LCD_LINE1, "Push Sensor: ");
    NumOut(80, LCD_LINE1, Sensor(S1));

    TextOut(0, LCD_LINE2, "Sound(dB): ");
    NumOut(80, LCD_LINE2, Sensor(S2));

    TextOut(0, LCD_LINE3, "Light: ");
    NumOut(80, LCD_LINE3, Sensor(S3));

    TextOut(0, LCD_LINE4, "Distance: ");

    //Adjust erroneus readings from the sensor
    int newDist = SensorUS(S4);
    if (newDist < 255)
    {
       dist = newDist;
    }
    NumOut(80, LCD_LINE4, dist);
```

```
     Wait(250); //just so that we do not overload the processor
 }
}

//Controls movement of the robot
//This task is only awaken when the shouldMove mutex is released
//(initially acquired such that it puts Drive to sleep)
//by pressing the right button.
//It releases the mutex as soon as it is done with the actual movement.
//The mutex is acquired again (and hence the Drive task put to sleep) by pressing the
//left button.
task Drive()
{
     while(true){

     Acquire(shouldMove);



     TurnLeft(90);
     GoForward(500);
     TurnRight(60);
     GoBackward(500);
     GoForward(1000);

     Release(shouldMove);
     if (wantMove)
```

```
          Wait(1000);
    }


}


//Simply returns the button that was pressed
byte waitButton() {
         if(ButtonPressed(BTNRIGHT,false)) {
         return 1;
         }
         if(ButtonPressed(BTNLEFT,false)) {
         return 2;
         }
         if(ButtonPressed(BTNCENTER,false)) {
         return 3;
         }


         return 0;
}



//This task runs continuously and checks if a button was pressed
//If the right button was pressed it releases the shouldMove mutex
//and consequently resumes the Drive task (movement)
//If th left button was pressed it acquires the shouldMove mutex
//which puts the Drive task to sleep (and stops movement).
task WaitB()
```

```
{
  bool mine = true;
  while (true)
  {
    int buttonP;
     buttonP = waitButton();
     switch(buttonP)
        {
        case 1: {

             if (mine){
                Release(shouldMove);
                mine = false;
             wantMove = true;
             }
           };break;
        case 2: {
          if(mine==false){
            wantMove = false;
            Acquire(shouldMove);
            mine = true;
          };

        }
     }
  }
}
```

```
//Initializes the sensors on the correct ports

//Acquires the shoulMove mutex such that when it is first started

//the Drive task is put to sleep (as it tries to acquire the same mutex)

//Starts the tasks

task main()

{

    SetSensorTouch(IN_1);

    SetSensorSound(IN_2);

    SetSensorLight(IN_3);

    SetSensorLowspeed(IN_4);


    Acquire(shouldMove);


    start WaitB;

    start PrintInfo;

    start Drive;

}
```