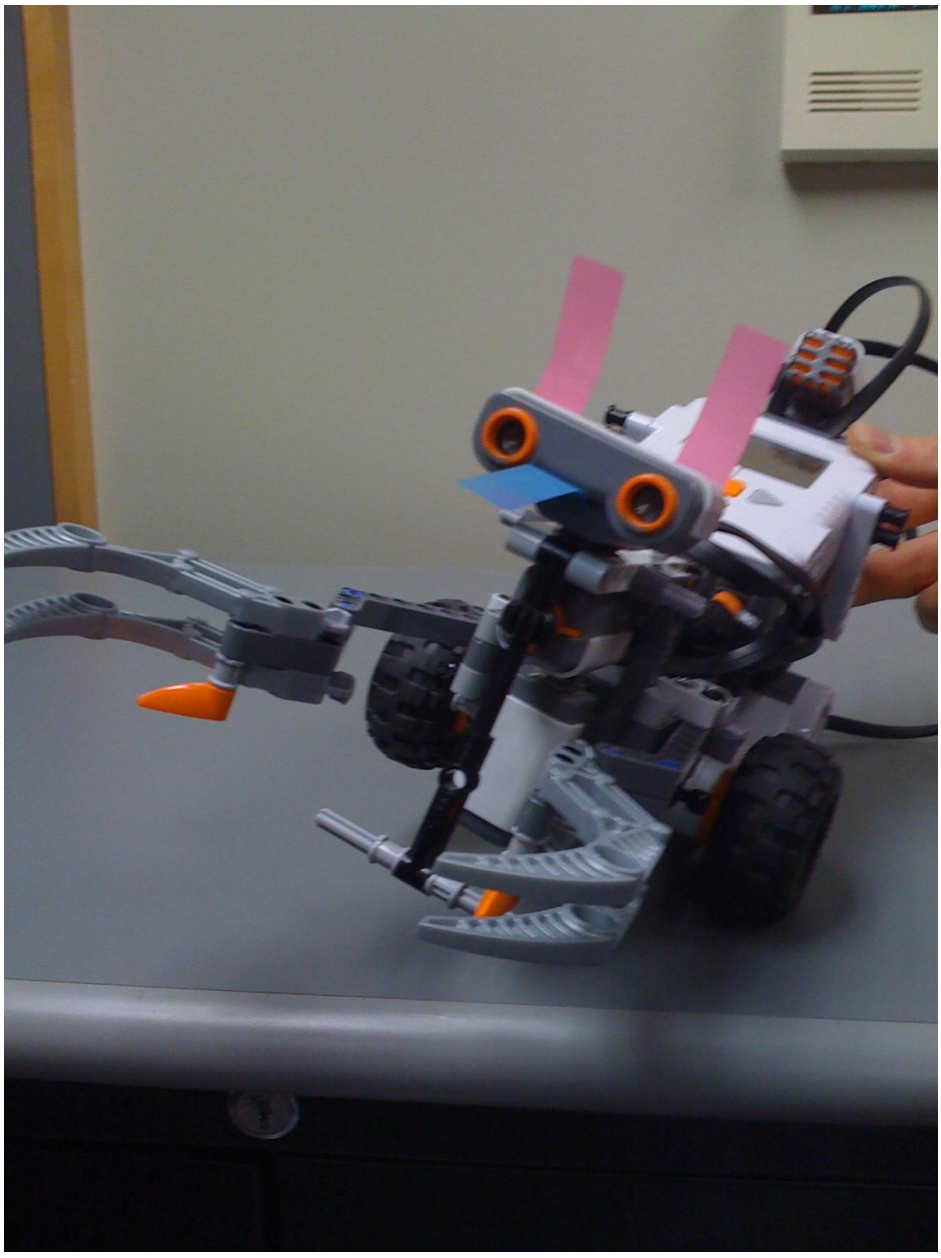
Lab Report 5: Search and verify



Rana Muaddi 260163563

Ionut Georgian Ciobanu 260242300

Contents

1. Field of view for each sensor

After various attempts to carefully build a good design of a robot that can successfully read distances to a ball on the floor and measure the light reflected off the ball in such a way the robot can distinguish between different ball colors, we arrived at a design we were happy with.

Ultrasonic sensor:

Using the ball as a reference, the ultrasonic sensor gave reliable readings for distances less than 40 cm. If the ball was further there was a chance it might not detect it, or generate very noisy readings. This is due to the surface of the ball being round.

The sensor can detect the ball's presence anywhere within its view region (illustrated below). Once the ball's **presence** is detected, the robot cannot simply drive forward to point at the ball (the ball may be to the left or to the right). To drive the robot to point at the ball we use an algorithm which is described in part 4.

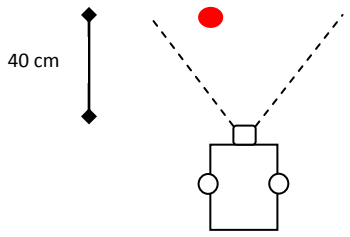


Figure 1: Ultrasonic sensor field of view

Light sensor:

The light sensor's functionality here is to distinguish the color of the ball. To do that, the light sensor has to be sufficiently close to the ball to shine its light on the ball and measure the reflected light from the ball. The red ball reflects more light than the blue ball, which reflects more light than the floor. We set an interval of reflected light values for the red and blue ball, and the floor. Fortunately those intervals do not overlap so the differentiation between colors was not very difficult.

One note, however, is that those intervals change according to where the experiment is performed due to ambient light.

To give reliable interval readings, the light sensor should be very close to the ball that it is almost touching it. This imposes interesting constrains on the positioning of the sensor and robustness (stability) of the robot in motion. For example, having the sensor very close to the ball and then suddenly stopping after detecting it might cause the sensor to jerk and hit the ball.

Our mounting of the sensors/design:

We had to criteria to satisfy when building our robot:

1. The ultrasonic sensor should be mounted low enough and fixed to read horizontally so that it can read distances and detect the ball (but not too low, in order to not detect the ground). The ideal position is at half the ball's diameter from the ground, such that the sensor is most probable to detect it.
2. The light sensor should be very close to the ball when directed at it.

To satisfy the criteria we first fixed the ultrasonic sensor low in front of the robot. Then we fixed the light sensor pointing vertically downwards at a height equal to the ball's diameter. While placing it horizontally or at an angle might allow for better readings – as the floor would not be visible to the sensor - we noticed that this positioning gave us a good range of movement around the ball, and eventually led to a more robust detection. Our sensor was positioned less than 3 mm from the top of the ball, giving an excellent signal-to-noise ratio. Also, one smart tweak would be to realize that while the blue ball is harder to detect – the reading of the light sensor is very close to the reading from the floor -, as long as actual ball detection is robust, if the checking for the ball did not conclude that it was red, then clearly the ball could only be blue.

To make our odometry more precise, we incorporated the compass sensor. We mounted the sensor over the rotational center or the robot at a height that ensures the sensor is far from the motors and the brick.

2. NXC Code

```
#define BLUE 1
#define RED 2
#define FLOOR 3
#define UNKNOWN 4

#define MAX_RANGE 41

#define LEFT -1
#define RIGHT 1
#define STATIC 0

#define US_SENSOR S4
#define COMPASS   S2
#define LIGHT     S3

#define MAX_ROWS 3

#define MAX_TRIES 40


//Lab 5 specific variables
int heading;
int ToMove = 0;

//Indicate what ball was found
string SFound;
int USDist;

//Linear distance the wheel travels by rotating one degree
//in tens of a milimeter
int decimmPERdeg = 5;

//Distance between wheels in deci milimeters
int distanceBetweenWheels = 1730;

//Get the distance covered by a wheel by rotating a number of degrees
```

```
//in tenths of a millimeter
long deg2Decimm(long degrees){
    return degrees*decimmPERdeg;
}

//Find out the angle of the arc described by the robot, theta
//Theta is in thousands' of degrees (mili degrees)
long getmTheta(long wLeftDist, long wRightDist){
    long tmp1 = wRightDist-wLeftDist;
    long tmp2 = tmp1 * 1000;
   return (tmp2/distanceBetweenWheels);
}

//Theta - angle the robot turned
//x,y - current position of robot
//dx, dy - distance robot moved since last sample
long theta, x, y, dx ,dy;

//variables used for internal stuff
long wLeftDeg = 0;
long wRightDeg = 0;
long wRightDist = 0;
long wLeftDist = 0;
long arc = 0;
long startAngle = 0;



//Go forward a specified number of milimeters
void MoveFW(long distMM, int power){
        int started = MotorRotationCount(OUT_A);
        OnFwdSync(OUT_AC, power, 0);
        while (MotorRotationCount(OUT_A) < started + (distMM*22)/10);
        Off(OUT_AC);
}


//Go backwards a specified number of milimeters
```

```
void MoveBW(long distMM, int power){
	int started = MotorRotationCount(OUT_A);
	OnRevSync(OUT_AC, power, 0);
	while (MotorRotationCount(OUT_A) > started - (distMM*22)/10);
	Off(OUT_AC);
}

//Rotate, relative to the current position, a specified number of degrees
//Negative values mean rotation to the left, positive values to the right
void RotateHelper(int degrees, int power){
	int initialHeading = theta;
	long computed =  (degrees*308)/100;

	if (degrees < 0)
		RotateMotorEx(OUT_AC, power, computed, -100, true, true);
	else
		RotateMotorEx(OUT_AC, power, computed, 100, true, true);
}



//Rotate to the absolute heading (in degrees, between 0 and 359)
void GoToHeading(int degrees, int power){
int tries = 0;
	while(theta!=degrees){
		int diff = degrees-theta;
		if ( abs(diff) > 180)
			if (diff > 0)
				diff = -(360 - diff);
			else diff = 360 + diff;

			RotateHelper(diff, power);
			tries++;

			//Just in case the algorith does not converge
			if (tries > 40)
      return;
		}
```

```
}


//Rotate the robot PRECISELY by that many degrees
void Rotate(int degrees, int power){
	int newHeading = theta+degrees;
	newHeading = newHeading % 360;
	if (newHeading<0)
		newHeading+=360;

	GoToHeading(newHeading, power);
}


//Sample the reading of the US sensor a couple of times to get
//an accurate reading.
int GetDistance(int SensorPort, int samples){
  int i = 0, value = 0;
  long sum = 0;
  int err = 0;
  int smp = 0;
  while( i < samples){
    value = SensorUS(SensorPort);
    if (value != 255){
      sum += value;
      smp++;
    }
    else
    {
      err ++;

    }
    i++;
  }
  if (err < samples /2 )
    return sum / smp;
  else return 255;
}
```

```
//Move to the absolute position given by X and Y
//Very nice function to use if a lot is known about the map
void GoToXY(long xMM, long yMM, int power)
{
	long dx = xMM-x;
	long dy = yMM-y;
	long dxs = dx*dx;
	long dys = dy*dy;
	dys += dxs;
	long distance = Sqrt(dys);
	long tempuraSauceInAWeirdSpoon;
	tempuraSauceInAWeirdSpoon =  dx*100 / distance;
	long heading = Asin(tempuraSauceInAWeirdSpoon);
	GoToHeading(heading, power);
	MoveFW(distance, power);
}

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


		//Lab 5 specific
		TextOut(0, LCD_LINE4, "USDist: ");
		NumOut(80, LCD_LINE4, USDist);

		TextOut(0, LCD_LINE5, "Light ");
		NumOut(80, LCD_LINE5, Sensor(LIGHT));

		TextOut(0, LCD_LINE6, "found: ");
		TextOut(60, LCD_LINE6, SFound);

		TextOut(0, LCD_LINE7, "ToMove: ");
```

```
                NumOut(60, LCD_LINE7, ToMove);

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



//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;
}



// Get the length of the arc described by the robot
long getArc(long wLeftDist, long wRightDist){
        return (wRightDist+wLeftDist)/2;
}




//This task updates the current position based on wheel rotation
//It samples as often as possible. The more samples, the better the precision.
//Theta = current heading
task UpdateValues()
```

```
{
        while(true){

                USDist = SensorUS(US_SENSOR);


                //Lab 5 specific
                //heading = 2*SensorUS(COMPASS);
                //td = SensorUS(US_SENSOR);
                //if (td <255)
                //  distance = td;

                //Lab2
                //Get the values from the global variable
                long newWLeftDeg = MotorRotationCount(OUT_A);
                long newWRightDeg = MotorRotationCount(OUT_C);


                //Update theta
  //theta = getmTheta(deg2Decimm(wLeftDeg) , deg2Decimm(wRightDeg)) / 10;
                theta = SensorHTCompass(COMPASS);

                //Only need to do something if the robot moved
                if(newWLeftDeg!=wLeftDeg || newWRightDeg!=wRightDeg){
                        //Compute the distance each wheel moved
                        wLeftDist = deg2Decimm(newWLeftDeg - wLeftDeg);
                        wRightDist = deg2Decimm(newWRightDeg - wRightDeg);

                        //Update the values for each wheel
                        wLeftDeg = newWLeftDeg;
                        wRightDeg = newWRightDeg;


                        //Get the length of the arc we moved
                        arc = getArc(wLeftDist, wRightDist);

                        //Compute dx and dy
                        dx = (arc * Sin(theta)) /100 ;
```

```
                dy = (arc * Cos(theta)) / 100;

                //Update our position
                long tempx = x + dx;
                long tempy = y + dy;

                y = tempy;
                x = tempx;
            }
        }

}


//This function checks the current position for the ball, and returns the color
//Direction indicates whether we saw the ball while moving to the left (-1) or right (1)
//or just by probing staticly (0)
int CheckIfBall(int Dist, int direction){


 int found = UNKNOWN;
        int light = Sensor(LIGHT);
         int tmp;

 //Sweep the area for the maximum value
 int initial = theta;
        for (int i = 0; i < 30; i++){
  tmp = Sensor(LIGHT);
  if (tmp > light)
    light = tmp;

  RotateHelper(3, 30);
}

 //Orient towards the ball
 GoToHeading(initial, 30);

        for (int i = 0; i < 30; i++){
```

```
  tmp = Sensor(LIGHT);
  if (tmp > light)
    light = tmp;

  RotateHelper(-3, 30);
 }

        if (light < 35 & light > 29){
                found = BLUE;
                SFound = "BLUE";

        }
        else if (light < 70 && light > 40){
                found = RED;
                SFound = "RED";

        }
        else if (light < 25 && light > 17){
                found = FLOOR;
                SFound = "FLOOR";

        }
        else {
                found = UNKNOWN;
                SFound = "UNKNOWN";

        }

        return found;
}


//Find the edge of the ball, where direction indicates the edge to be found (LEFT or
RIGHT)
int findBallEdge(int dir){
        while (GetDistance(US_SENSOR, 11) < MAX_RANGE){
                if (dir == LEFT)
                        RotateHelper(-4,30);
```

```
                else
                        RotateHelper(4,30);
                Wait(100);
        }
        return theta;
}

//Find ball heading
int FindBallHeading(int startDirection){
 //Save initial position
 int initial = theta;

 int R, L;

        R = findBallEdge(RIGHT);
        //As you are now pointing PAST the ball, move back a little
        RotateHelper(-4, 30);
        L = findBallEdge(LEFT);

        int angle = (L+R)/2;

        return angle;
}


//Sweep <step> degrees at the time, for a maximum of 90 degrees, and check if there is
a ball
int FindBall(int step){
int i;
  for (i=0; i <= 90+step ; i+=step){
                //Try until you get a valid value
                Wait(100);
                int distToBall = SensorUS(US_SENSOR);

                if ( distToBall < MAX_RANGE ){
   RotateHelper(10,30);
   return distToBall;
  }
```

```
   RotateHelper(step, 30);
  }
  return -1;
}

//The main function, incorporating the ball searching algorithm
//As well as the movement of the robot
//It essentially sweeps the area to the right (and front) of the robot,
//then locates the ball, moves close to it and detects its color.
void SearchForBalls(int step){
          int i = 0, row =0, col = 0;
          int current = theta;
          int distToBall, ToReverse ;

  //Move around in a square
          while(true){
                    for (col = 0; col < 3; col++){
     ToReverse = 0;
                              if (col == 2)
                              {
                                        Rotate(90, 30);
                                        current = theta;
                              }
                              else
                              {  //Find the ball
       distToBall = FindBall(10);

       if (distToBall > 0 && distToBall != 255){
         //Find the ball heading
                                                  int ballHeading = FindBallHeading(RIGHT);
                                                  GoToHeading(ballHeading, 30);

                                                  distToBall = SensorUS(US_SENSOR);
                                                   int dist = distToBall;

                                                   int step = 10;
                                                   int Moved = 0;
```

```
                                                    //Move close to the ball
                                                    do{
          dist = SensorUS(US_SENSOR);

          MoveFW(step, 40);
          Moved += step;

         } while (dist > 9 && Moved < (distToBall*10)/2 )

         Off(OUT_AC);
         //Turn on the searchlight
         SetSensorLight(LIGHT);
                                                     CheckIfBall(dist, RIGHT);

         //Display the value on the screen, look for another ball if needed
                                                    Wait(20000);

       }

                                  }
                                  GoToHeading(current,30);

                                  MoveFW(300, 50);
                            }
 }

}


task Check()
{


          SearchForBalls(15);


}
```

```
//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
//Initializes variables
task main()
{
	x=0;
	dx=0;
	y=0;
	dy=0;
	wRightDist = 0;
	wLeftDist = 0;

//	SFound = "UNKNOWN";

	//This is the real US sensor
	SetSensorLowspeed (US_SENSOR);
	SetSensorLowspeed(COMPASS);

  //The heading of the robot.
	theta = SensorHTCompass(COMPASS);


	start UpdateValues;
	start PrintInfo;
	start Check;

}
```

3. Validation

To test our code, we placed a ball in various locations on the specified area and started the robot. The robot sweeps the area in a square pattern, moving only on the outer squares, until it sees the ball. After every square, it stops and turns 90 degrees to the right to check for balls in that region. Then the robot moves towards the ball, adjusts its heading precisely and reads the ball color by sweeping 30 degrees to each side, on the arc that includes the ball. The maximum value read is taken as the correct reading for the color of the ball.
The robot always starts from the bottom left corner.

Most values read are not taken at face value: when critical, readings from the sensors are sampled multiple times, in order to get as good a value as possible.

The code was tested module by module: we first fine-tuned the odometry with the new compass sensor. We then iteratively designed the robot and performed various tests to find the best position/orientation of the compass. We tested the compass for accuracy and correctness (when rotating the robot 90 degrees, does the compass indicate it correctly?). We tested whether the reading from the compass is accurate, and adjusted our code in order to calibrate it.

We then re-tested the control code for moving in a straight line and rotating in place. They were both further improved to give very good accuracy.

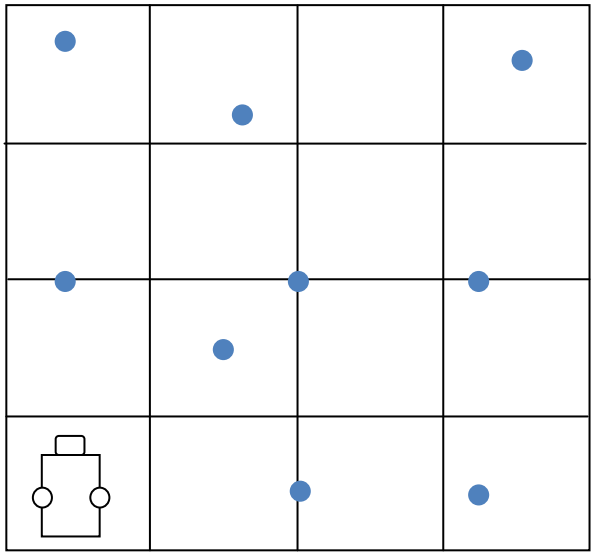


Figure 2: testing scheme (different ball locations)

4. Analysis of performance

Our algorithm is as follows:

To sweep the entire area:
REPEAT (until back to original block or ball found)
- Rotate clockwise by small increments to a max of 90 degrees
- After each increment read distance from US sensor
- Go to the next block
- If block number 4 is reached rotate 90 degrees clockwise

If at any time a ball was detected:
- find ball heading
- move towards the ball slowly
- rotate 30 degrees left, back to the original position, and then 30 degrees right while reading values from light sensor. Save maximum reading.
- use that reading to determine the color of the ball.

This turns out to be a comprehensive, fast algorithm to sweep the area. It covers the whole surface completely and goes (in the absence of any ball) over the course in less than 80 seconds. If a ball is found, the most time-consuming part is finding the correct heading – this is done by sampling each distance reading 10 times and finding the heading within one degree. This approach proved to be very successful at both eliminating noise and pinpointing the location of the ball very accurately.

The code itself is neatly divided into loosely coupled functions, with APIs at several layers. This allows us to easily combine them and quickly change the behavior if needed, as well as easily debug the code.