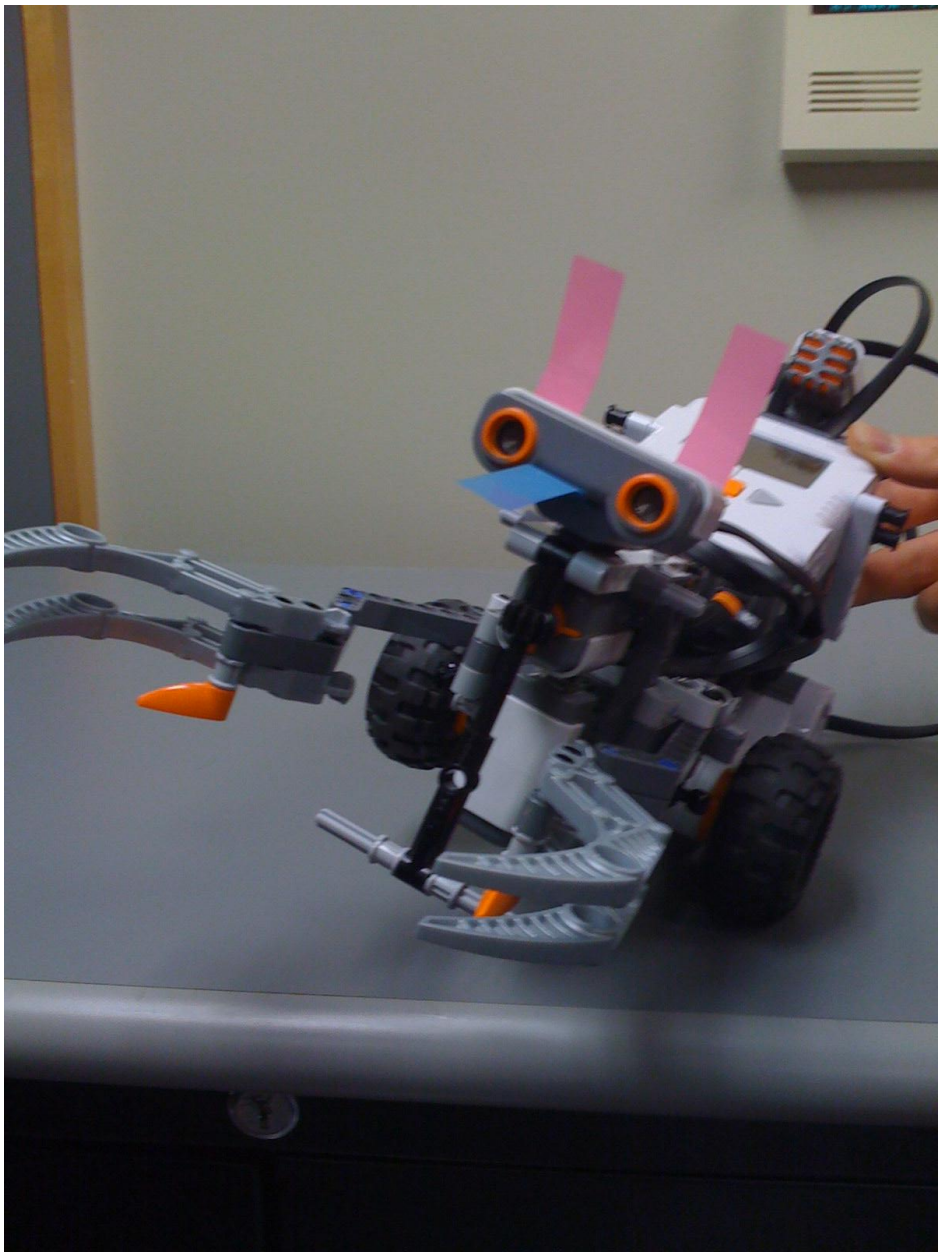
Lab Report 6: Bluetooth Rendez-vous



Rana Muaddi 260163563

Ionut Georgian Ciobanu 260242300

Contents

1. NXC Code

a. Master Code

```
#include "BTlib.nxc"

#define MAILBOX_OFFSET 0 //Master writes to mailboxes 0-4


#define SEARCHING 0

#define BALLFOUND 1

#define MAX_TRIES 40


//Value used to make sure we adjust for the initial heading.

int startDir;




//Global variables used

int status = SEARCHING;

string msgX = "", msgY="", msgS="", msg="";       // text received from slave


//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;
```

```
//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 = 1150;

//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 fi=0;
long arc = 0;
int tupi= 6283;
long startAngle = 0;
```

```
//Go forward a specified number of milimeters
void GoForward(long distMM, int power){
 //   RotateMotorEx(OUT_AC, power, distMM*2, 0, true, true);
          int started = MotorRotationCount(OUT_A);
          OnFwdSync(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 Rotate(int degrees, int power){
    if (degrees < 0)
        RotateMotorEx(OUT_AC, power, (degrees*23)/10, -100, true, true);
    else
        RotateMotorEx(OUT_AC, power, (degrees*23)/10, 100, true, true);
}

//Move backwards a specified number of milimeters
void GoBackwards(long distMM, int power){
 //    RotateMotorEx(OUT_AC, -power, distMM*2, 0, true, true);
          int started = MotorRotationCount(OUT_A);
          OnRevSync(OUT_AC, power, 0);
          while (MotorRotationCount(OUT_A) > started - (distMM*22)/10);
          Off(OUT_AC);
```

```
}

//After certain operations on the heading, it might end up being an invalid value.
//This function fixes it
int NormalizeHeading(int Heading){
   Heading %= 360;
   if (Heading < 0)
     Heading += 360;

   return Heading;
}

//Distance between two points on a circle
//It is trickier than it seems
//TODO: this might return 360, which is not valid
//It is a low-priority bug as it implies indentical headings
int CircleDistance(int NewHeading, int CurrentHeading){
    int diff = NewHeading - CurrentHeading;
    if ( abs(diff) > 180)
       if (diff > 0)
          diff = -(360-diff);
       else diff = 360 + diff;

    return diff;
}
```

```
//Rotate to the absolute heading (in degrees, between 0 and 359)
//This usually converges in less than 10 iterations
//This is great, given that it positions the robot at exactly the required heading
void GoToHeading(int NewHeading, int power){
int tries = 0;
int newTemp = NewHeading;
        while(theta*57/1000!=NewHeading){
  int diff = CircleDistance(NewHeading, theta*57/1000);

    if (tries > 1)
                              Rotate(diff, 24);
    else Rotate(diff, power);

                            tries++;

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

   Wait(40);
        }
}

//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/10;
    long dy = yMM-y/10;
    long dxs = dx*dx;
    long dys = dy*dy;
    dys += dxs;
    long distance = Sqrt(dys);

    //You gotta just love this variable
    long tempuraSauceInAWeirdSpoon;
    tempuraSauceInAWeirdSpoon =  dx*100 / distance;
    long heading = Asin(tempuraSauceInAWeirdSpoon);
    heading = NormalizeHeading(heading);
    GoToHeading(heading, power);
    GoForward(distance, power);
}



//Print various information to the screen
task PrintInfo()
{
 while (true){
```

```
    ClearScreen();
    TextOut(0, LCD_LINE1, "X: ");
                    NumOut(80, LCD_LINE1, x/100);

                    TextOut(0, LCD_LINE2, "Y: ");
                    NumOut(80, LCD_LINE2, y/100);

                    TextOut(0, LCD_LINE3, "Theta: ");
                    NumOut(80, LCD_LINE3, theta*57/1000);

      // display the latest data
   TextOut(0, LCD_LINE6, "slaveX:            ");
   TextOut(80, LCD_LINE6, msgX);
   TextOut(0, LCD_LINE7, "slaveY:            ");
   TextOut(80, LCD_LINE7, msgY);
   TextOut(0, LCD_LINE8, "slaveStatus:       ");
   NumOut(80, LCD_LINE8, status);
   TextOut(0, LCD_LINE5, "              ");

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



//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);
}

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

//Correct the distance from an arc to a chord
long correction2Cord(long theta){
    long a = theta/2;
    if (theta==0)
      return 1;
    else return (Sin(a)/a);
}
```

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

      //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;

       //Compute theta based on the two wheels
```

```
      long newTheta = getmTheta(deg2Decimm(wLeftDeg) , deg2Decimm(wRightDeg));

      //Adjust theta so that it's always between 0 and 359 degrees
      newTheta = newTheta % tupi;
      if (newTheta < 0)
        newTheta += tupi;

      //Fi is the angle of the vector joining the old and new position
      fi = (newTheta + theta)/2;

      //Update theta
      theta = newTheta;



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

      //Correct such that we get the chord of the arc, for better precision.
//    long correction = correction2Cord(theta);
//    correctedVector = arc * correction;

       //Convert to degrees, compute dx and dy
       long cfi = fi*57/1000;
       dx = (arc * Sin(cfi)) /100 ;
```

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

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

        y = tempy;
        x = tempx;
      }
    }

}



//Continually update readings from slave
task recieveFromSlave(){
  while (true) {
    // notify about reception
    TextOut(0, LCD_LINE5, "Receiving...    ");

    // receive and display the data
    msg = BTReceiveMessage(1, MAILBOX_OFFSET+0, true);       //receieve slave's x position
    if (msg != "") msgX = msg;
```

```
   msg = BTReceiveMessage(1, MAILBOX_OFFSET+1, true);       //receive slave's y position
   if (msg != "") msgY = msg;
   msg = BTReceiveMessage(1, MAILBOX_OFFSET+2, true);       //receive slave's status
   if (msg != "") msgS = msg;
  //  msg = BTReceiveMessage(1, MAILBOX_OFFSET+3, true);       //receive slave's status
  // if (msg != "") startDir = StrToNum(msg);

            status = StrToNum(msgS);                        //convert status to number

   // wait a bit
   Wait(100);
  }
}

//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;
```

```
    theta=0;
    wRightDist = 0;
    wLeftDist = 0;



   start UpdateValues;
   start PrintInfo;
   start recieveFromSlave;                    //constatly receives messages from the slave


  while (status == SEARCHING);                //wait as long as the slave is searching for the ball
  Wait(1000);
  //convert x and y positions to numbers
  int toX = StrToNum(msgX);
  int toY = StrToNum(msgY);

//   go to heading(with offset)
  GoToXY(toX*10, 0, 40);                //go to position required by slave
  GoToXY(toX*10, toY*10, 40);
  GoToXY((toX-5)*10, toY*10, 40);

}
```

b. Slave Code

```
#include "BTlib.nxc"

#define MAILBOX_OFFSET 0


#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



#define SEARCHING 0

#define BALLFOUND 1

//Lab 5,6 specific variables

int heading = 0;

int ToMove = 0;

long distance = 0;

int togox = 0, togoy = 0;

int status = SEARCHING;



//Temp value used for debugging

int newTemp;

//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);
}

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

```
}



//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, startDir;

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



//Comments in Master.nxc
//Adjust for 0-359 interval
int NormalizeHeading(int Heading){
  Heading %= 360;
  if (Heading < 0)
```

```
    Heading += 360;


  return Heading;
}


//Comments in Master.nxc
//Get distance on a circle
int CircleDistance(int NewHeading, int CurrentHeading){
  NewHeading = NormalizeHeading(NewHeading);
    CurrentHeading = NormalizeHeading(CurrentHeading);


  diff = NewHeading - CurrentHeading;
                    if ( abs(diff) > 180)
                              if (diff > 0)
                                        diff = -(360 - diff);
                              else diff = 360 + diff;


 return diff;
}


//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 finalHeading = NormalizeHeading(theta + degrees);
 long RoughComputed;


   RoughComputed = (degrees*33)/10;


 //Rough positioning, witihin 6 degrees
```

```
 if (degrees < 0)
                    RotateMotorEx(OUT_AC, power, RoughComputed, -100, true, true);
          else
                   RotateMotorEx(OUT_AC, power, RoughComputed, 100, true, true);


 //Fine positioning - This code will be improved, for now it is inactive.
/*
          if (degrees < 0)
   while (theta > finalHeading)
                    RotateMotorEx(OUT_AC, power, 10, -100, true, true);
          else
   while (theta < finalHeading)
                   RotateMotorEx(OUT_AC, power, 10, 100, true, true);*/
}


//Rotate to the absolute heading (in degrees, between 0 and 359)
void GoToHeading(int NewHeading, int power){

NormalizeHeading(NewHeading);

int tries = 0;
newTemp = NewHeading;
```

```
        while(theta!=NewHeading){
  diff = CircleDistance(NewHeading, theta);

    if (tries > 1)
                          RotateHelper(diff, 24);
    else RotateHelper (diff, power);

                          tries++;

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

   Wait(40);
        }
}

//Rotate the robot EXACTLY by that many degrees (controlled)
void Rotate(int degrees, int power){
        int newHeading = theta+degrees;
 newHeading = NormalizeHeading(newHeading);

        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;
	distance = Sqrt(dys);
	long tempuraSauceInAWeirdSpoon;
	tempuraSauceInAWeirdSpoon =  dx*100 / distance;
	heading = Asin(tempuraSauceInAWeirdSpoon);
	GoToHeading(heading, power);
	MoveFW(distance/100, power);
}

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

```
//Lab 5 specific
TextOut(0, LCD_LINE2, "STATUS: ");
NumOut(80, LCD_LINE2, status);

        TextOut(0, LCD_LINE1, "Heading: ");
NumOut(80, LCD_LINE1, theta);

TextOut(0, LCD_LINE3, "Distance: ");
NumOut(80, LCD_LINE3, distance);

TextOut(0, LCD_LINE4, "X: ");
NumOut(60, LCD_LINE4, x/100);

TextOut(0, LCD_LINE5, "Y: ");
NumOut(60, LCD_LINE5, y/100);

TextOut(0, LCD_LINE6, "TO GO X: ");
NumOut(60, LCD_LINE6, togox/100);

TextOut(0, LCD_LINE7, "TO GO Y: ");
NumOut(60, LCD_LINE7, togoy/100);

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

```
        }
}
```

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

      long newTheta = SensorHTCompass(COMPASS)-startDir;
      //Update theta
       theta= newTheta;

      //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);

       //Convert to degrees, 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 < 10; i++){
  tmp = Sensor(LIGHT);
  if (tmp > light)
    light = tmp;


  RotateHelper(3, 30);
}


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


          for (int i = 0; i < 10; i++){
  tmp = Sensor(LIGHT);
```

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

  RotateHelper(-3, 30);
 }

 GoToHeading(initial, 30);

 if (light < 70 && light > 40){
                found = RED;
                SFound = "RED";
                status = BALLFOUND;
        }
        else if (light < 25 && light > 17){
                found = BLUE;
                SFound = "BLUE";
                status = BALLFOUND;
  }
        return found;
}


//Find the edge of the ball, where direction indicates the edge to be found (LEFT or RIGHT)
int findBallEdge(int dir, int precision){
        while (GetDistance(US_SENSOR, 11) < MAX_RANGE){
```

```
		if (dir == LEFT)
			RotateHelper(-precision,30);
		else
			RotateHelper(precision,30);
		Wait(100);
	}
	return theta;
}

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

 int R, L;

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

 //Note: L+R / 2 does not work for L=359 and R = 4
 int toAdd = CircleDistance(L,R);
	int angle = L + abs(toAdd)/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;
        ballHeading = FindBallHeading();
                                                GoToHeading(ballHeading, 30);

                                                distToBall = SensorUS(US_SENSOR);
```

```
                                        int dist = distToBall;


                                        int step = 10;
                                        int Moved = 0;


                                        //Move close to the ball
while( !(dist < 9 || Moved > (distToBall*10)/2 )){
  dist = SensorUS(US_SENSOR);


  //This is where we can make things much better,
  //By correcting the heading more often or by other means
  MoveFW(step, 40);
  //GoToHeading(ballHeading,30);
  Moved += step;


  //check heading every 7 cm
  /*if (Moved%70 == 0){
                          ballHeading = FindBallHeading(RIGHT);
                                        GoToHeading(ballHeading, 30);
  } */
}


 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(1000);


                                        status = BALLFOUND;


                                        Off(OUT_AC);
                                        //Turn it off
                                        Wait(90000); //Just wait, no real purpose
    }


                                    }
                                    GoToHeading(current,30);


                                    MoveFW(300, 50);
                                }
 }


}



task Check()
{
```

```
            SearchForBalls(15);

}


      //Slave code

//Regularly send data to the master
task communicateToMaster(){

  while (true) {



                        if(status==SEARCHING){
     togoy = y;
     togox = x;
                        }
                        else{
            togoy = y + 20*Cos(theta);
                togox = x+ 20*Sin(theta)+2500;
                        }
   // send the values
   BTSendMessage(0, MAILBOX_OFFSET+0, NumToStr(togox/100));        //send x position
   BTSendMessage(0, MAILBOX_OFFSET+1, NumToStr(togoy/100));        //send y position
   BTSendMessage(0, MAILBOX_OFFSET+2, NumToStr(status));       //send status
```

```
   // wait a bit
   Wait(100);
  }
}



//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.
  startDir = SensorHTCompass(COMPASS);
  theta = SensorHTCompass(COMPASS)- startDir;
  theta = NormalizeHeading(theta);


        start UpdateValues;
        start PrintInfo;
        start Check;
        start communicateToMaster;


}
```

2. Validation

To test our code, we placed a ball somewhere in the area specified. We placed the slave on the bottom left corner and the master on the left of the slave. We pair the robots using Bluetooth and start the robots.

The slave sweeps the area until it finds the ball. As it does that, it sends its position (x and y) to the Master. The master stays in its place and displays the slave's position on its screen.
Once the ball is found the slave sends a flag to the Master that the ball was found and sends it the x and y coordinates it should go to, which is the position on the other side of the ball. The Master goes there and stops. While moving, the master uses the wall follower code to make sure it does not bump into the slave.
The code was tested module by module: we first fine-tuned the odometry of both robots. The slave operating using the compass sensor and the master using the wheel rotation counts.
We then re-tested the control code for moving in a straight line and rotating in place. They were both further improved to give maximum possible accuracy.

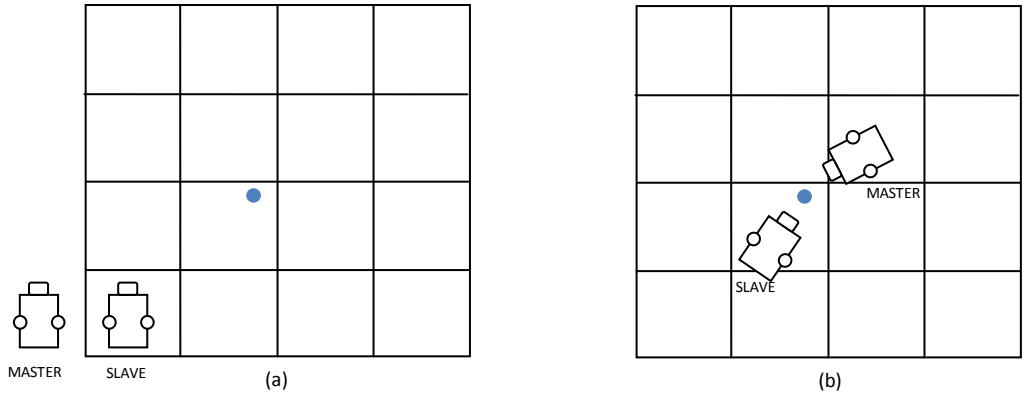


Figure 1: beginning (a) and ending (b) positions of master and slave

3. Performance Analysis

The first robot takes the standard time (less than 90 seconds) to sweep the whole area and find the ball. Then the second robot moves at the location directly, making sure to avoid the first one. This takes in the range of 1-10 seconds. We did not need to integrate the odometry as we had already done so for lab 5.

Our control code is as precise as it can possibly be: the robot is, after a couple of iterations (inside the Rotate() function), heading exactly at the indicated position. The odometry code is less precise by using the compass, than our initial odometry code. Thus, the ideal way to use the compass is just for checking the rotation indicated by the wheels versus the absolute one given by the compass. This helps detect cases when the robot's wheels are moving but the robot itself is not, or when accumulated errors are more than 1 degree.

Sending data over Bluetooth was easy using the provided library. To make sure that we have the latest information on the slave before proceeding to the rendez-vous location, we wait one between getting the notification that the ball was found and actually starting to move.