Lab report 2 – Odometry

Rana Muaddi 260163563

Ionut Georgian Ciobanu 260242300

Curly - the robot

1. Problem analysis

To calculate the radius R of the wheels:

-Use the MotorTachoCount() function to display the rotation of the wheels on the robot's screen.
-Mark the initial position of the robot and move it forward carefully without sliding until the rotation reading of both wheels is 360 degrees.
-Mark the final position of the robot and measure the distance between the initial position and final position.
-This distance is the circumference C of the wheel. C=2*pi*R. C=180mm
- To find R divide C by 2pi. R=C/(2*pi).

To calculate the distance X between the wheels:

- We marked both wheels of the robot with ink and made it rotate 360 degrees in its place by rotating the wheels in opposite directions.
- The diameter of the circle produced is the distance between the wheels.

To calculate distance moved by a wheel in mm:

-when a wheel rotates 360 degrees the wheel moved by circumference C
360 degrees → C mm

=> distance moved = degrees moved*C/360 = degrees moved*0.5

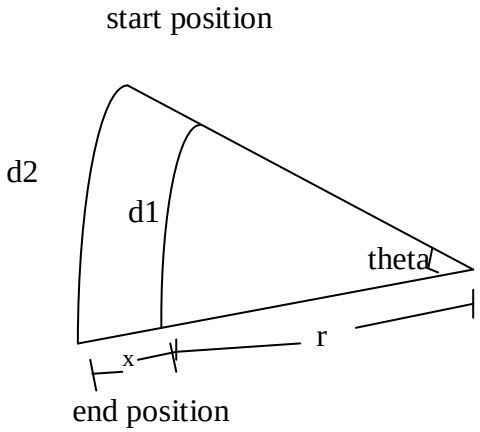
we created a constant called deg2Decimm which stores the constant 0.5 mm as a conversion rate.

To get the angle theta which represents the heading of the robot:

use the following diagram and some geometry to find theta in radians
d1 distance moved by left wheel
d2 distance moved by right wheel
x distance between wheels
r radius of inner arc
theta angle rotated

d1= r*theta
d2 = (r+x)*theta
= r*theta + x*theta
d2 = d1 + x*theta

=> theta = (d2-d1)/x



To get the length of arc moved by robot

-we used the following diagram and some geometry to calculate the length of th arc moved by the robot.

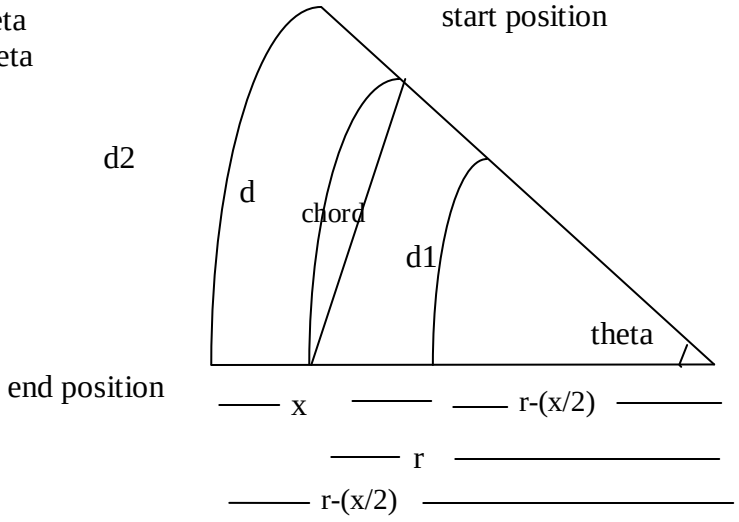
d1 = (r-x/2)*theta
d2 = (r+x/2)*theta
d = r*theta

Since it is difficult to find r we find 2*theta directly from d1 and d2:

d1 = r*theta - (x/2)*theta
d2 = r*theta + (x/2)*theta

d1+d2 = 2(r*theta)

=> r*theta = (d1+d2)/2



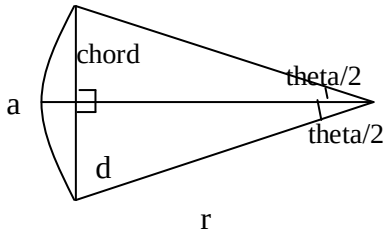
To find the absolute distance moved by the robot:

a is arc length
d is chord length or absolute distance moved

- Draw angle bisector which is also an altitude
  - d/2 = r*sin(theta/2)
  - d = 2r*sin(theta/2)

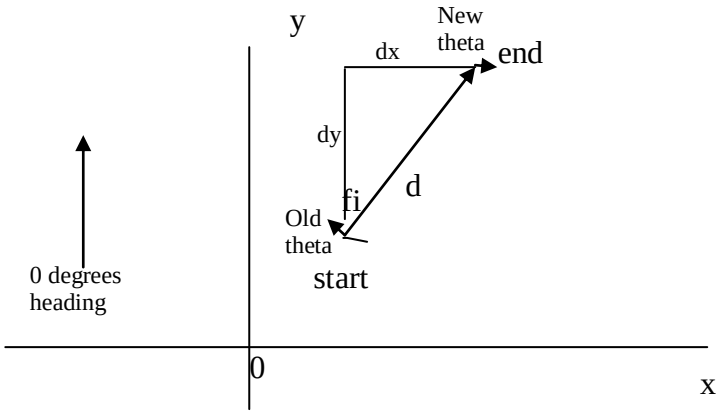
  - d/a = [2r*sin(theta/2)] /[r*theta]
  - d/a = sin(theta/2)/(theta/2)

=> d = [sin(theta/2)/(theta/2)]*a



To find X and Y positions

- find the angle fi first to find dx and dy
- d = distance moved by robot
- fi = average of old theta and new theta
- using trigonomentry:
- dx = d*sin(fi)
- dy = d*cos(fi)



Finally the absolute position of the robot is calculated by summing up all the dx's and dy's separately
x = x + dx
y = y + dy

The heading of the robot is calculated with respect to the positive y axis. Clockwise rotations are positive and counter-clockwise rotations are negative.

2. Our Algorithm

There are two core distinct values to be computed: heading and Cartesian position.

The heading is measured in degrees, relative to an initial point of reference, with values from 0 to 359 degrees. This is the exact same system used in navigation, with the only difference that in navigation the initial point of reference is the magnetic north.
The heading is measured using the total rotation of each of the wheels, in degrees. By multiplying by a constant factor, we can find the actual rotation of the robot. As this value (theta) is always computed based on the tacho readings of the two wheels, it's as precise as the readings. There is no accumulated computation or approximation error.

The Cartesian position is the projection of the current distance from an initial point on two perpendicular axes - let's call the X and Y. The values stored (and displayed) are the projected components on the two axes. These values are computed from the distances the two wheels traveled. These small differences are then added to the total displacement on each axis. Thus, we are essentially integrating over time.

Pseudo code:

```
At every step
  If any of the wheels moved
    Update heading
    Compute dx, dy
    Add dx, dy to total displacement
```

3. Code listing

```
//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;

//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);
}

//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);
}

//Rotate to the absolute heading (in degrees, between 0 and 359)
void GoToHeading(int degrees, int power){
    int diff = degrees-theta*57/1000;
    if ( abs(diff) > 180)
      if (diff > 0)
        diff = diff-360;
      else diff = -(360 + diff);

    Rotate(diff, power);
}

//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);
    GoForward(distance, power);
}

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

    TextOut(0, LCD_LINE1, "X : ");
    NumOut(60, LCD_LINE1, x/100);

    TextOut(0, LCD_LINE2, "Y : ");
    NumOut(60, LCD_LINE2,  y/100);
```

```
    TextOut(0, LCD_LINE3, "Theta: ");
    NumOut(60, LCD_LINE3, (theta*57)/1000);

    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()
{
    Acquire(shouldMove);

    GoForward(20000,100);

    Wait(1000);

   GoToXY(100, 500, 70);
   Wait(3000);

   Rotate(90, 70);
   GoToHeading(10, 70);
   Wait(3000);
   GoForward(500, 70);
   Rotate(90, 60);
   GoForward(20000, 100);
   Rotate(90, 60);
   Rotate(90, 60);
   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;
}


//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;
      }
    }

}

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


    Acquire(shouldMove);
```

```
    start UpdateValues;
    start WaitB;
    start PrintInfo;
    start Drive;
}
```

4. Results of the experiment in Step 4

We performed a couple of runs at various powers.
We made the robot (named Curly) go in a square with 5m sides at 30, 70 and 100% power.
We then had a more complicated shape with 4 turns, totaling 20 meters.

Absolute error [cm]				
Shape	**Distance**	Run 1 (30% power)	Run 2 (70% power)	Run 3 (100% power)
Square	5 meters(side)	3.7	5	6.9
Irregular	20 meters (total)	4.6	6.2	6.7

5. Discussion of results

In both cases, the faster the robot went (greater power) the more imprecise it got. At 100% power, it achieved a precision of roughly 7 cm from where it was supposed to be at the end of the run. Greater power increases the error due to the momentum of the robot - it keeps moving even after the motors have stopped. It also gets fewer samples per second, thus errors from approximating the chord with an arc are accumulating faster.

We need to differentiate between the information of the robot's position (odometry) and actually moving the robot to a certain position.
Assuming we are getting exact information on how the robot moves and based on this we issue commands, we could then position the robot almost correctly. But there would still be errors in positioning the robot, caused by non-recorded movement: sliding and skidding.
With the odometry errors, when trying to drive Curly, we will thus have added errors from two processes. We then expect that we will always be able to give relatively precise relative movement commands, but over a long distance the absolute position readings will be off by a lot.