Peter Huynh

October 14, 2016

CSCI 3302

Lab 3.3

Navigation

$$\begin{pmatrix} \dot{x_R} \\ \dot{y_R} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \frac{r\dot{\phi_l}}{2} + \frac{r\dot{\phi_r}}{2} \\ 0 \\ \frac{\dot{\phi_r} r}{d} - \frac{\dot{\phi_l} r}{d} \end{pmatrix} \tag{3.63}$$

for ϕ_l, ϕ_r

$$\begin{aligned} \dot{\phi_l} &= (2\dot{x}_R - \dot{\theta}d)/2r \\ \dot{\phi_r} &= (2\dot{x}_R + \dot{\theta}d)/2r \end{aligned} \tag{3.64}$$

Goal:

- Turning a discrete path into a robot trajectory

Required:

- A Sparki robot
- A working implementation of Dijkstra's algorithm returning a list of waypoints
- Working odometry and feedback control code to drive to any point on the plane

Provided with an obstacle-free path consisting of discrete waypoints, you will need to generate a trajectory for the robot to follow. We will do this by first calculating X-Y coordinates in world coordinates for each waypoint and then use feedback control to drive the robot there.

.

Instructions:

1. Write code that calculates an X-Y coordinate in the real world based on the I-J coordinates of your 2D map. This coordinate should be the center of each grid cell.
2. Implement a state machine that pulls an entry from the route, calculate its X-Y coordinate and then use feedback control to drive there. Pull the next coordinate of the stack once the robot is close enough to the last waypoint.

Deliverables

Provide a write-up describing your approach including a picture of the state machine you implemented. Answer the following questions:

1. How did you chose the distance at which you pull the next waypoint? What happens if you do this too early or too late?

   I pulled the next waypoint about midway in a full traversal, when the x and/or y coordinates were approximately halfway through a node. Snippet below:

```
// Odometry - Keepin track of distance traveled
if(fmod(x, 60) >= 30 || fmod(y, 60) >= 30)
{
  rho   = sqrt(sq(x - (columnGoal2 * 60)) + sq(y - (rowGoal2 * 60)));
  alpha = theta - atan2((y - (rowGoal2 * 60)), (x - (columnGoal2 * 60)));
  mu    = thetaGoal2 - theta;
}
else
{
  rho = sqrt(sq(x - (columnGoal * 60)) + sq(y - (rowGoal * 60)));
  alpha = theta - atan2((y - (rowGoal * 60)), (x - (columnGoal * 60)));
  mu    = thetaGoal - theta;
}
forwardSpeed    = P1 * rho;
rotationalSpeed = P2 * alpha + P3 * mu;

rightWheel = ((2 * forwardSpeed) + (rotationalSpeed * WHEEL_DISTANCES)) / (2 * WHEEL_RADIUS);
leftWheel  = ((2 * forwardSpeed) - (rotationalSpeed * WHEEL_DISTANCES)) / (2 * WHEEL_RADIUS);


if(rightWheel > WHEEL_VELOCITY/2) rightWheel = WHEEL_VELOCITY;
if(leftWheel > WHEEL_VELOCITY/2) leftWheel = WHEEL_VELOCITY;

leftWheelPercent = (100 * (leftWheel/WHEEL_VELOCITY));
rightWheelPercent = (100 * (rightWheel/WHEEL_VELOCITY));

sparki.motorRotate(MOTOR_LEFT, DIR_CCW, leftWheelPercent);
sparki.motorRotate(MOTOR_RIGHT, DIR_CW, rightWheelPercent);


// Determines X and Y using half of summed wheel speeds
halfLeftRight = (rightWheel + leftWheel) / 2;
x += cos(theta * (PI/180)) * halfLeftRight;
y += sin(theta * (PI/180)) * halfLeftRight;
theta = (((rightWheel * WHEEL_RADIUS) / WHEEL_DISTANCES) - ((leftWheel * WHEEL_RADIUS) / WHEEL_DISTANCES)) * (180/PI);
//theta += rotationalSpeed * (180/PI);
if(theta >= 360) theta = fmod(theta, 360);
if(theta <= -360) theta = -fmod(abs(theta), 360);

// Set current Column and Row values from X and Y, done for scaling
currentColumn = x / 60;
currentRow = y / 60;
```

   Note: goal1 and goal2 values are the waypoints, which is determined in another part of the code. First I determine the immediate next waypoint and then check the adjacent nodes of that waypoint to determine the second waypoint. I repeat this process upon each traversal.

   Pulling the next waypoint determines the point of rotational movement. For example, if I were to pull the next waypoint too early, Sparki would attempt to reach that waypoint prior to nearly reaching the previous one. This would cause Sparki to start arcing too soon. The opposite applies for pulling too late, Sparki would start the curved movement too far into the next node.

2. What do you need to do should an unforeseen obstacle appear? Try to use tools/algorithms from previous exercises to solve this problem.

   Currently, I have that algorithm commented out. However, if an unforeseen obstacle appears, I would use the object detection via the ultrasound sensor to detect distances of unknown objects and then update the maps accordingly. Snippet below:

```
// Set current Column and Row values from X and Y, done for scaling
currentColumn = x / 60;
currentRow = y / 60;

// Object Detection, changes world map if sees object within 20 cm COMMENTED OUT
cmDetect = sparki.ping();
// if(cmDetect != -1)
// {
//   if(cmDetect < 10)
//   {
//     // Northern Object
//     if(currentColumn+1 <= 3 && (theta <= 5 && theta >= -5) || (theta <= 365 && theta >= 355) || (theta <= -355 && theta >= -365))
//       bestPathMap[currentRow][currentColumn+1] = worldMap[currentRow][currentColumn+1] = 0;
//     // East
//     else if(currentRow + 1 <= 3 && (theta >= 85 && theta <= 95) || (theta >= -275 && theta <= -265))
//       bestPathMap[currentRow+1][currentColumn] = worldMap[currentRow+1][currentColumn] = 0;
//     // West
//     else if(currentRow-1 >= 0 && (theta >= 265 && theta <= 275) || (theta >= -95 && theta <= -85))
//       bestPathMap[currentRow-1][currentColumn] = worldMap[currentRow-1][currentColumn] = 0;
//     // South
//     else if(currentColumn-1 >= 0 && (theta >= 175 && theta <= 185) || (theta >= -185 && theta <= -175))
//       bestPathMap[currentRow][currentColumn-1] = worldMap[currentRow][currentColumn-1] = 0;
//   }
// }
```

   Sparki detects objects based on its current theta position. If it see a new obstacle, it updates the adjacent node based off theta.

Snippet of determining Waypoints:

```
// Checks boundary limits, if in bounds, adjust compass values, if not, set to 0 for
blockage
  if(currentColumn+1 <= 3) north = bestPathMap[currentRow][currentColumn+1];
  else north = 0;
  if(currentRow + 1 <= 3) east = bestPathMap[currentRow+1][currentColumn];
  else east = 0;
  if(currentRow - 1 >= 0) west = bestPathMap[currentRow-1][currentColumn];
  else west = 0;
  if(currentColumn - 1 >= 0) south = bestPathMap[currentRow][currentColumn-1];
  else south = 0;

  // Now Does Traversal, based on compass values and theta for rotation
  // Proritizes Clockwise Motion
  if(north == 1)
  {
    thetaGoal = 0;
    columnGoal = currentColumn + 1;
    rowGoal = currentRow;

    if(currentColumn + 2 <= 3) north2 = bestPathMap[currentRow][currentColumn+2];
```

```
    else north2 = 0;
    if(currentRow + 1 <= 3) east2 = bestPathMap[currentRow+1][currentColumn+1];
    else east2 = 0;
    if(currentRow - 1 >= 0) west2 = bestPathMap[currentRow-1][currentColumn+1];
    else west2 = 0;
    if(currentColumn >= 0) south2 = bestPathMap[currentRow][currentColumn];
    else south2 = 0;

    // Now Does Traversal, based on compass values and theta for rotation
    // Proritizes Clockwise Motion
    if(north2 == 1)
    {
      thetaGoal2 = 0;
      columnGoal2 = currentColumn + 2;
      rowGoal2 = currentRow;
    }
    else if(east2 == 1)
    {
      thetaGoal2 = -90;
      columnGoal2 = currentColumn + 1;
      rowGoal2 = currentRow + 1;
    }
    else if(south2 == 1)
    {
      thetaGoal2 = 180;
      columnGoal2 = currentColumn;
      rowGoal2 = currentRow;
    }
    else if(west2 == 1)
    {
      thetaGoal2 = 90;
      columnGoal2 = currentColumn + 1;
      rowGoal2 = currentRow - 1;
    }
    else
    {
      thetaGoal2 = thetaGoal;
      columnGoal2 = columnGoal;
      rowGoal2 = rowGoal;
    }
  }
```

NOTE: I've attached two copies of my code. One with working forward kinematics through the maze and another with the odometry inverse kinematics. The inverse version is fairly buggy, but follows the procedure of this lab. The Forward Kinematics version works flawlessly, but is a different approach to what this lab requires.