ChenHao Cheng
Adam Siefkas
Charles Davies
Kyle Hartland Brown

CSCI 3302
lab3.3 **Fall 2017, CU-Boulder**

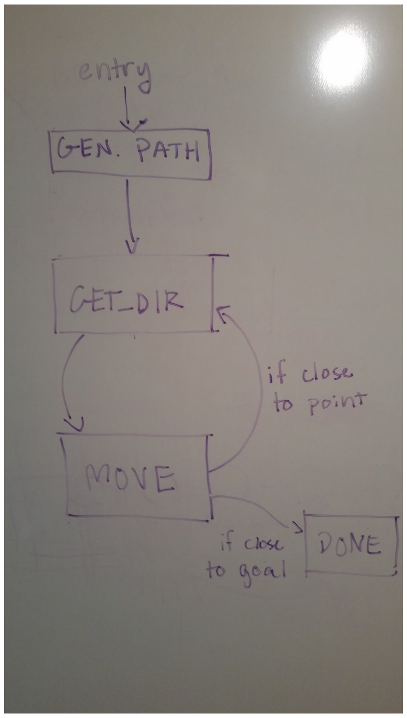

Figure1: State machine

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?**

   Explanation:

   We chose a tolerance of 2 cm to pull the next waypoint. We chose this number because it is close enough to ensure sparki is in the correct square, but is far enough that sparki wont start dancing in an effort to get closer to the point.

   If the chosen tolerance value is too large, then it will try to move on to the next waypoint without actually reaching the first waypoint. This could cause problems because we are trying to avoid obstacles. If we move on to the next waypoint too early, then the robot might try to turn into an obstacle.

   If the chosen tolerance value is too small, then sparki will try forever to get to the perfect point. This might mess up the orientation of sparki. If sparki is trying to move in a straight line, but is off by a little bit, when he gets close to the point, he will

ChenHao Cheng
Adam Siefkas
Charles Davies
Kyle Hartland Brown
Fall 2017, CU-Boulder

turn to get closer to the point. This could also cause him to run into obstacles if he is turned too far.

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

   Explanation:

   Should an unforeseen obstacle appear, we could use the ultrasound sensor (US) to identify it and its range. Sparki could use information about its rotation, and the range to the obstacle to find the (x,y) coordinates of the obstacle. Once this is calculated, we could store update the obstacle array to include it. We could then run Dijkstras again to update our path, and avoid it.

Next page is the code:

```
#include <Sparki.h>

enum robotStates
{
  MOVE,
  GET_DIR,
  DONE
} state;

const float maxspeed=0.0285;    // [m/s] speed of the robot that you measured
const float alength=0.0851;     // [m] axle length
//const float alength=0.1;
const float blockWidth = 0.148; // in cm
const float blockHeight = 0.105; // in cm
const int NUMNODES = 16;
const int LARGE_INT = 100;
const int BLOCK_WEIGHT = 50;

//float Xi = 0, Yi = 0, Thetai = 0;
// where the robot is in the world
float Xi = blockWidth / 2.0, Yi = blockHeight / 2.0, Thetai = 0.0;
// where the robot is in the world
float Xg = 0.0;     // Where the robot should go
float Yg = 0.0;
float Thetag = 0;
float Xrdot, Thetardot;    // how fast the robot moves in its coordinate system
float phildotr = 0, phirdotr = 0; // wheel speeds that you sent to the motors

float alpha, rho, eta; // error between positions in terms of angle to the goal
, distance to the goal, and final angle
float a = 0.05, b = 0.75, c = 0.0; // controller gains

int pathPoint = 0;
int path[16];
int cleanPath[16];
int blockGoal = 6;
int blockedSquares[4][4] = {{1,1,1,1},
                            {0,0,0,1},
                            {0,1,1,1},
                            {0,0,0,1}};

//function definitions

void getCenterCoords(int block);
void moveToGoal();
void dijkstra(int startIndex, int goalIndex);
int minNotChecked(int checked[], int arr[], int size);
int isEmpty(int arr[], int size);
int inBounds(int neighbor, int cur);
int pathWeight(int current, int neighbor);
int isPath(int current, int neighbor);
void cleanUpPath();

void setup() {
  //prepare path
  for (int i = 0; i < 16; i++)
  {
    path[i] = 99;
  }
  // put your setup code here, to run once:
  sparki.beep();
  state = GET_DIR;
  dijkstra(0,6);
  //cleanUpPath();
  int i = 15;
  while(path[i] == 99){
    i--;
    pathPoint = i;
  }

  Serial.begin(9600);
  while(!Serial){;}
  Serial.println(pathPoint);
  for(int j = 0; j < 16; j++){
      Serial.println(path[j]);
  }
}

void loop()
{
  // put your main code here, to run repeatedly:
    switch (state){
      case MOVE:
      {
        getCenterCoords(blockGoal);
        moveToGoal();          // holds state until close to goal
        sparki.beep();
        state = GET_DIR;
        break;
      }

      case GET_DIR:
      {
//        Serial.print("pathPoint: ");
//        Serial.println(pathPoint);
        if (pathPoint == -1){state = DONE; break;}

        blockGoal = path[pathPoint];
        pathPoint--;

        state = MOVE;
        break;
      }

      case DONE:
      {
        sparki.clearLCD();
        sparki.println("DONE!");
        sparki.updateLCD();
        break;
      }
    }

    delay(100);
}

void getCenterCoords(int block)
{
  Xg = (block % 4) * blockWidth + blockWidth / 2;
  Yg = (block / 4) * blockHeight + blockHeight / 2;
  Serial.print("Xg: ");
  Serial.println(Xg);
  Serial.print("Yg: ");
  Serial.println(Yg);
  Serial.print("block: ");
  Serial.println(block);
}

void moveToGoal()
{

  float tolerance = 0.02; //one cm
  float diffX = abs(Xg-Xi);
  float diffY = abs(Yg-Yi);

//  Serial.print("diffX: ");
//  Serial.println(diffX);
//  Serial.print("diffY: ");
//  Serial.println(diffY);
//  Serial.print("Xg: ");
//  Serial.println(Xg);
//  Serial.print("Yg: ");
//  Serial.println(Yg);


  while ((diffX > tolerance) || (diffY > tolerance))
  {
    long int time_start = millis();
```

ChenHao Cheng
Adam Siefkas
Charles Davies
Kyle Hartland Brown
Fall 2017, CU-Boulder

```
int threshold = 700;
Serial.print("diffX: ");
Serial.print(diffX);
Serial.print("");
Serial.print("");
Serial.print("");

Serial.print(" ,diffY: ");
Serial.println(diffY);
// CALCULATE ERROR
rho   = sqrt((Xi-Xg)*(Xi-Xg)+(Yi-Yg)*(Yi-Yg));
//alpha = Thetai-atan2(Yi-Yg,Xi-Xg)-PI/2.0;
alpha = atan2(Yg-Yi,Xg-Xi)-Thetai;
eta   = Thetai-Thetag;

// CALCULATE SPEED IN ROBOT COORDINATE SYSTEM
Xrdot = a * rho;
//Xrdot=0;
Thetardot = b * alpha; //+ c * eta;

// CALCULATE WHEEL SPEED
phildotr = (2 * Xrdot - Thetardot * alength) / (2.0);
phirdotr = (2 * Xrdot + Thetardot * alength) / (2.0);

// SET WHEELSPEED

float leftspeed = abs(phildotr);
float rightspeed = abs(phirdotr);

if(leftspeed > maxspeed)
{
  leftspeed = maxspeed;
}
if(rightspeed > maxspeed)
{
  rightspeed = maxspeed;
}
leftspeed = (leftspeed/maxspeed) * 100;//100
```

ChenHao Cheng
Adam Siefkas
Charles Davies
Kyle Hartland Brown
Fall 2017, CU-Boulder

```
rightspeed = (rightspeed/maxspeed) * 100;//100

if(rho > 0.01)  // if farther away than 1cm
{
  if(phildotr > 0)
  {
    sparki.motorRotate(MOTOR_LEFT, DIR_CCW,leftspeed);
  }
  else
  {
    sparki.motorRotate(MOTOR_LEFT, DIR_CW,leftspeed);
  }
  if(phirdotr > 0)
  {
    sparki.motorRotate(MOTOR_RIGHT, DIR_CW,rightspeed);
  }
  else
  {
    sparki.motorRotate(MOTOR_RIGHT, DIR_CCW,rightspeed);
  }
}
else
{
  sparki.moveStop();
}

sparki.clearLCD(); // wipe the screen

sparki.print(Xi);
sparki.print("/");
sparki.print(Yi);
sparki.print("/");
sparki.print(Thetai);
sparki.println();
sparki.print(alpha/PI*180);
sparki.println();

sparki.updateLCD(); // display all of the information written to the screen
```

ChenHao Cheng
Adam Siefkas
Charles Davies
Kyle Hartland Brown
Fall 2017, CU-Boulder

```
    // perform odometry
    Xrdot = phildotr / 2.0 + phirdotr / 2.0;
    Thetardot = phirdotr / alength - phildotr / alength;

    Xi = Xi + cos(Thetai) * Xrdot * 0.1;
    Yi = Yi + sin(Thetai) * Xrdot * 0.1;
    Thetai = Thetai + Thetardot * 0.1;
    diffX = abs(Xg-Xi);
    diffY = abs(Yg-Yi);
//    Serial.print("milliSec: ");
//    Serial.println(millis() - time_start);
    while(millis()<time_start + 100); // wait until 100ms have elapsed
  }
}

void dijkstra(int startIndex, int goalIndex)
{

  int cur = startIndex;
  int checked[16] = {0};
  int parents[16] = {0};

  int dist[16];

  for (int i = 0; i < NUMNODES; i++)
  {
    dist[i] = LARGE_INT;
  }

  dist[cur] = 0;

  while (!isEmpty(checked, NUMNODES))
  {

    cur = minNotChecked(checked, dist, NUMNODES);
    checked[cur] = 1;
    int neighbors[4] = {cur-1,
                        cur+4,
                        cur+1,
                        cur-4};

    for (int i = 0; i < 4; i++)
    {
      if (inBounds(neighbors[i], cur) && isPath(cur, neighbors[i]))
      {
        int altDist = dist[cur] + 1;
        if (altDist < dist[neighbors[i]])
        {
          dist[neighbors[i]] = altDist;
          parents[neighbors[i]] = cur;
        }
      }
    }

    int curPath = goalIndex;

    int i = 0;
    while (curPath != startIndex)
    {
      path[i] = curPath;
      curPath = parents[curPath];
      i++;
    }

    for (int i = 0; i < NUMNODES; i++)
    {
      Serial.println(path[i]);
    }
  }
}

int minNotChecked(int checked[], int arr[], int size)
{
  //ret index of minimum value
  int min = 999;
  int minIndex;

  for (int i = 0; i < size; i++)
  {
    if ((arr[i] < min) && (!checked[i]))
    {
      min = arr[i];
      minIndex = i;
    }
  }

  return minIndex;

}

int isEmpty(int arr[], int size)
{
  //zero in arr is not checked, one is checked
  for (int i = 0; i < size; i++)
  {
    if (arr[i] == 0)
    {
      return 0;
    }
  }
  return 1;
}

int inBounds(int neighbor, int current)
{
  int currentX = current%4;
  int currentY = current/4;

  int neighborX = neighbor%4;
  int neighborY = neighbor/4;

  //valid coords would be (+1,0), (-1,0), (0,+1), (0,-1)
```

```
    int diffX = abs(neighborX-currentX);
    int diffY = abs(neighborY-currentY);

    if ((neighbor > 15) || (neighbor < 0))
    {
      return 0;
    }

    if ((diffX == 1) && (diffY == 0))
    {
      return 1;
    }

    else if ((diffX == 0) && (diffY == 1))
    {
      return 1;
    }

    return 0;
  }

  int isPath(int current, int neighbor)
  {
    int currentX = current%4;
    int currentY = current/4;

    int neighborX = neighbor%4;
    int neighborY = neighbor/4;

    if ((blockedSquares[currentX][currentY] == 0) ||
    (blockedSquares[neighborX][neighborY] == 0))
    {
      return 0;
    }
    return 1;
  }

  void cleanUpPath()
```

```
{
int i = 15;
int j = 0;
while(i>0){
  if(path[i] != 0){
    cleanPath[j] = path[i];
    j++;
  }
  i--;
}
}
```