/\*

\*Team Id: 3310

\*Author List: Rachit Rathi, Aashay Phirke, Sudhanshu Lodha, Sarah Sonje

\*Filename: NS\_Task\_1\_Sandbox.cpp

\*Theme: Nutty Squirrel

\*Functions: Task\_1\_1, linefollow,node\_x (x is from 1 to 12)

\*Global Variables: NONE

\*/

#include "NS\_Task\_1\_Sandbox.h"

#include<stdio.h>

#include<stdlib.h>

#include <limits.h>

/\*

\*

\* Function Name: forward\_wls

\* Input: node

\* Output: void

\* Logic: Uses white line sensors to go forward by the number of nodes specified

\* Example Call: forward\_wls(2); //Goes forward by two nodes

\*

\*/

void forward\_wls(unsigned char node)

{

int x = node;

while (x != 0)

{

unsigned char left\_sensor, centre\_sensor, right\_sensor;

left\_sensor = ADC\_Conversion(1);//Read values from left sensor

centre\_sensor = ADC\_Conversion(2);//Read values from centre sensor

right\_sensor = ADC\_Conversion(3);//Read values from right sensor

printf("\n %d %d %d ", left\_sensor, centre\_sensor, right\_sensor);

forward();

if (left\_sensor >= 100 && centre\_sensor >= 100 && right\_sensor >= 100)//FORWARD

{

\_delay\_ms(1000);

stop();

\_delay\_ms(500);

x--;

}

}

}

/\*

\*

\* Function Name: left\_turn\_wls

\* Input: void

\* Output: void

\* Logic: Uses white line sensors to turn left until black line is encountered

\* Example Call: left\_turn\_wls(); //Turns right until black line is encountered

\*

\*/

void left\_turn\_wls(void)//For Turning Left at the Node

{

left();

\_delay\_ms(150);

while (ADC\_Conversion(2) < 200)

{

left();

velocity(200, 200);

}

stop();

\_delay\_ms(2000);

}

/\*

\*

\* Function Name: right\_turn\_wls

\* Input: void

\* Output: void

\* Logic: Uses white line sensors to turn right until black line is encountered

\* Example Call: right\_turn\_wls(); //Turns right until black line is encountered

\*/

void right\_turn\_wls(void)//For Turning Right at the Node

{

right();

\_delay\_ms(150);

while (ADC\_Conversion(2) < 200)

{

right();

velocity(200, 200);

}

stop();

\_delay\_ms(2000);

}

/\*

\*

\* Function Name: Square

\* Input: void

\* Output: void

\* Logic: Use this function to make the robot trace a square path on the arena

\* Example Call: Square();

\*/

void Square(void)

{

}

/\*

\*

\* Function Name: Task\_1\_1

\* Input: void

\* Output: void

\* Logic: Main Program from which other sub functions are called.

\* Example Call: Task\_1\_1();

\*/

void Task\_1\_1(void)//MAIN PROGRAM

{

}

typedef struct node

{

int cost;

int reqAlign;

int futAlign;

}node;

node graph[24][24];

int present;

int presAlign;

int n;

int pathLength;

void makeNode(int i, int j, int cost, int reqAlign, int futAlign)

{

graph[i][j].cost = cost;

graph[i][j].reqAlign = reqAlign;

graph[i][j].futAlign = futAlign;

}

int createGraph()

{

printf("Creating graph\n");

int i, j;

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

graph[i][j].cost = 0;

graph[i][j].reqAlign = 0;

graph[i][j].futAlign = 0;

}

}

makeNode(0, 1, 1, 0, 0);

makeNode(1, 0, 1, 180, 180);

makeNode(1, 2, 2, 0, 0);

makeNode(2, 1, 2, 180, 180);

makeNode(2, 3, 8, 0, 0);

makeNode(3, 2, 8, 180, 180);

makeNode(2, 12, 3, 30, 30);

makeNode(12, 2, 3, 240, 240);

makeNode(2, 19, 3, 30, 30);

makeNode(19, 2, 3, 240, 240);

makeNode(3, 6, 3, 90, 90);

makeNode(6, 3, 3, 270, 270);

makeNode(3, 14, 3, 270, 270);

makeNode(14, 3, 3, 90, 90);

makeNode(4, 7, 5, 180, 180);

makeNode(7, 4, 5, 0, 0);

makeNode(4, 5, 3, 270, 270);

makeNode(5, 4, 3, 90, 90);

makeNode(5, 6, 3, 270, 270);

makeNode(6, 5, 3, 90, 90);

makeNode(5, 8, 4, 180, 180);

makeNode(8, 5, 4, 0, 0);

makeNode(7, 9, 5, 180, 180);

makeNode(9, 7, 5, 0, 0);

makeNode(7, 8, 4, 315, 315);

makeNode(8, 7, 4, 135, 135);

makeNode(8, 12, 6, 180, 225);

makeNode(12, 8, 6, 45, 0);

makeNode(9, 13, 3, 180, 180);

makeNode(13, 9, 3, 0, 0);

makeNode(9, 10, 3, 270, 270);

makeNode(10, 9, 3, 90, 90);

makeNode(10, 11, 1, 270, 270);

makeNode(11, 10, 1, 90, 90);

makeNode(11, 12, 3, 0, 315);

makeNode(12, 11, 3, 135, 180);

makeNode(11, 20, 7, 180, 0);

makeNode(20, 11, 7, 180, 0);

makeNode(14, 15, 3, 270, 270);

makeNode(15, 14, 3, 90, 90);

makeNode(15, 16, 3, 270, 270);

makeNode(16, 15, 3, 90, 90);

makeNode(15, 17, 4, 180, 180);

makeNode(17, 15, 4, 0, 0);

makeNode(16, 18, 5, 180, 180);

makeNode(18, 16, 5, 0, 0);

makeNode(17, 18, 4, 225, 225);

makeNode(18, 17, 4, 45, 45);

makeNode(17, 19, 6, 180, 135);

makeNode(19, 17, 6, 315, 0);

makeNode(18, 22, 5, 180, 180);

makeNode(22, 18, 5, 0, 0);

makeNode(19, 20, 3, 225, 180);

makeNode(20, 19, 3, 0, 45);

makeNode(20, 21, 1, 270, 270);

makeNode(21, 20, 1, 90, 90);

makeNode(21, 22, 3, 270, 270);

makeNode(22, 21, 3, 90, 90);

makeNode(22, 23, 3, 180, 180);

makeNode(23, 22, 3, 0, 0);

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

printf("%d ", graph[i][j].cost);

}

printf("\n");

}

return 1;

}

int minDistance(int dist[], int sptSet[])

{

int min = INT\_MAX, min\_index;

int v;

for (v = 0; v < n; v++)

if (sptSet[v] == 0 && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

int\* printSolution(int dist[], int parent[], int destn)

{

int src = present;

int \* path = (int\*)malloc(1 \* sizeof(int));

path[0] = destn;

int i, j, k;

for (i = 1; path[i - 1] != src; i++)

{

path = (int \*)realloc(path, (i + 1) \* sizeof(int));

path[i] = parent[path[i - 1]];

}

for (j = i - 1, k = 0; j >= i / 2; j--, k++)

{

int temp = path[k];

path[k] = path[j];

path[j] = temp;

}

for (j = 0; j < i; j++)

{

printf(" %d ", path[j]);

}

pathLength = i;

\_delay\_ms(200);

return path;

}

int\* findPathTo(int destn)

{

int src = present;

int parent[24];

int i, count;

int dist[24];

int sptSet[24];

for (i = 0; i < n; i++)

{

parent[i] = -1;

dist[i] = INT\_MAX;

sptSet[i] = 0;

}

dist[src] = 0;

for (count = 0; count < n - 1; count++)

{

int u = minDistance(dist, sptSet);

sptSet[u] = 1;

int v;

for (v = 0; v < n; v++)

if (!sptSet[v] && graph[u][v].cost && dist[u] != INT\_MAX && dist[u] + graph[u][v].cost < dist[v])

{

parent[v] = u;

dist[v] = dist[u] + graph[u][v].cost;

}

}

printf("Path made\n");

int \*path = printSolution(dist, parent, destn);

\_delay\_ms(200);

return path;

}

//PLEASE COMPLETE

int turn(int degTurn)

{

printf("turn");

//UNCOMMENT THIS

if (degTurn == 0)

{

follow\_line();

}

//turn right

printf("Turn %ds right\n", degTurn); //COMMENT THIS

return 1;

}

//PLEASE COMPLETE

int isObstacle()//how will the bot know the destination

{

if (ADC\_Conversion(FRONT\_IR\_ADC\_CHANNEL) < 100)

return 1;

else

return 0;

}

//PLEASE COMPLETE

int follow\_line()

{

unsigned char left\_sensor, centre\_sensor, right\_sensor;

while (1)

{

if (isObstacle() == 1)

{

return 0;

}

printf("Following line");

//follow line until next node encountered

left\_sensor = ADC\_Conversion(1);

centre\_sensor = ADC\_Conversion(2);

right\_sensor = ADC\_Conversion(3);

printf("\n %d %d %d ", left\_sensor, centre\_sensor, right\_sensor);

if (left\_sensor <= 100 && centre\_sensor >= 100 && right\_sensor <= 100)//FORWARD

{

forward();

velocity(50, 50);

}

if (left\_sensor <= 100 && centre\_sensor <= 100 && right\_sensor >= 100)//RIGHT

{

soft\_right();

velocity(50, 50);

}

if (left\_sensor >= 100 && centre\_sensor <= 100 && right\_sensor <= 100)//LEFT

{

soft\_left();

velocity(50, 50);

}

if (left\_sensor <= 100 && centre\_sensor <= 100 && right\_sensor <= 100)//STOP

{

stop();

\_delay\_ms(500);

}

if (left\_sensor >= 100 && centre\_sensor >= 100 && right\_sensor >= 100)

{

\_delay\_ms(350);

stop();

\_delay\_ms(3000);

//break;

}

}

\_delay\_ms(300);

return 1;

}

//turn doubt

int moveTo(int dest)

{

printf("moveto");

int reqAlign = graph[present][dest].reqAlign;

turn(reqAlign - presAlign);

int result = follow\_line();

if (result == 0)

{

graph[present][dest].cost = 10000;

return 0;

}

else

{

printf("\nReached node %d\n", dest);

presAlign = graph[present][dest].futAlign;

present = dest;

return 1;

}

}

int moveAlong(int \*path)

{

printf("movealong");

int i;

for (i = 1; i < pathLength; i++)

{

printf("Next Hop: %d\n", path[i]);

int result = moveTo(path[i]);

\_delay\_ms(200);

if (result == 0)

{

return 0;

}

}

return 1;

}

//PLEASE COMPLETE

int pickUpFrom(int destn)

{

int \*path = findPathTo(destn);

printf("\nPath Finalized\n");

\_delay\_ms(200);

while (moveAlong(path) == 0)

{

free(path);

right\_turn\_wls();//180

presAlign = (presAlign + 180 ) % 360;

path = findPathTo(destn);

}

pick();

return 1;

}

//PLEASE COMPLETE

int placeAt(int destn)

{

int \* path = findPathTo(destn);

\_delay\_ms(200);

while (moveAlong(path) != 1)

{

free(path);

right\_turn\_wls();//180

presAlign = (presAlign + 180) % 360;

path = findPathTo(destn);

}

place();

return 1;

}

void Task\_1\_2(void)

{

n = 24;

present = 0;

presAlign = 0;

createGraph();

while (1)

{

forward();

velocity(10, 10);

if (ADC\_Conversion(1) == 255 && ADC\_Conversion(2) == 255 && ADC\_Conversion(3) == 255)

{

\_delay\_ms(2000);

stop();

\_delay\_ms(2000);

break;

}

}

\_delay\_ms(200);

moveAlong(findPathTo(3));

nutsearch();

}

void check\_color(void)

{

unsigned int red\_pulse\_count, green\_pulse\_count, blue\_pulse\_count, filter\_clear\_pulse\_count;

filter\_red();

red\_pulse\_count = color\_sensor\_pulse\_count;

\_delay\_ms(200);

filter\_green();

green\_pulse\_count = color\_sensor\_pulse\_count;

\_delay\_ms(200);

filter\_blue();

blue\_pulse\_count = color\_sensor\_pulse\_count;

filter\_clear();

filter\_clear\_pulse\_count = color\_sensor\_pulse\_count;

printf("\n%d %d %d %d", red\_pulse\_count, green\_pulse\_count, blue\_pulse\_count, filter\_clear\_pulse\_count);

\_delay\_ms(1000);

int i = 0, j = 0;

if (red\_pulse\_count >= 2000 && green\_pulse\_count <= 2000 && blue\_pulse\_count <= 2000)

{

printf("Color is RED");

red\_pick(i);

i++;

}

if (red\_pulse\_count <= 2000 && green\_pulse\_count >= 2000 && blue\_pulse\_count <= 2000)

{

printf("Color is GREEN");

green\_pick(j);

j++;

}

if (red\_pulse\_count <= 2000 && green\_pulse\_count <= 2000 && blue\_pulse\_count >= 2000)

{

printf("Color is BLUE");

}

}

void red\_pick(int i)

{

pick();

//function for node

if (i == 0)

{

moveAlong(findPathTo(13)); //move to 13 change it to 6

place();

}

if (i == 1)

{

//move to 10 change it to 7

place();

}

}

void green\_pick(int i)

{

pick();

if (i == 0)

{

moveAlong(findPathTo(21)); //move to 21 change it to 8

place();

}

if (i == 1)

{

//move to 23 change it to 9

place();

}

}

void scan() //360 scan

{

for (int i = 4; i < 7; i++) //for loop through array

{

right\_turn\_wls();

check\_color();

}

}

void nutsearch()

{

int a[6] = { 4,5,6,14,15,16 };//array which includes nodes 4 5 6 14 15 16 change it to 0 to 5

for (int i = 0; i < 6; i++) //for loop through array

{

moveAlong(findPathTo(a[i])); //move to i

\_delay\_ms(150);

scan(); //360 scan //turn and check\_color at every loop

}

//13 10 21 23

//4 5 6 14 15 16

}

#include "NS\_Task\_1\_Predef.h"

extern unsigned int color\_sensor\_pulse\_count;

/\*

\* Function Name: forward\_wls

\* Input: node

\* Output: void

\* Logic: Uses white line sensors to go forward by the nodes specified

\* Example Call: forward\_wls(2) //Goes forward by two nodes

\*

\*/

void forward\_wls(unsigned char node);

void left\_turn\_wls(void);

void right\_turn\_wls(void);

void Square(void);

void Task\_1\_1(void);

void Task\_1\_2(void);

void red\_pick(int i);

void green\_pick(int j);

void check\_color(void);

void nutsearch();

int follow\_line();

/\*

int placeAt(int destn);

int pickUpFrom(int destn);

int moveAlong(int \*path);

int moveTo(int dest);

\*/