#include "mbed.h"

#include "nRF24L01P.h"

#include <cstdio>

BusOut display1(PC\_4, PB\_13, PB\_14, PB\_15, PB\_1, PB\_2, PB\_12, PA\_11);

BusOut display2(PC\_3, PC\_2, PB\_7, PA\_14, PC\_12, PC\_10, PD\_2, PC\_11);

nRF24L01P receiver(D11, D12, D13,D7,D8); //Initialize nRF24L01P

BusOut motorControl(D2, D3, D4, D5);

PwmOut ENA(D6);

PwmOut ENB(D9);

//Timers and Threads

Thread displayThread;

Thread navigationThread;

Timer timerController;

Timer displayUpdateTimer;

Timer navigationTimer;

//Starting and ending coordinates

int startX, startY, startDir;

int endX, endY;

//Cordinates and direction

int currentX;

int currentY;

int direction;

//Global variables

double movementTime, turnTime; //Seconds needed to complete certain actions

bool inMotion; //Tracks if robot is currently in the middle of moving

bool isTurning; //Tracks if robot is currently rotating

bool isReversing; //Tracks if robot is reversing

bool isDancing; //Tracks if the robot is dancing

bool enabledMovements; //Whether the robot is allowed to continue moving or not

bool enableTransmission; //Whether to broadcast navigation result time or not

//Return hex value to display on seven segment

int displayHexValue(int number) {

int display = 0x00;

switch (number) {

case 0: display = 0x3F; break;

case 1: display = 0x06; break;

case 2: display = 0x5B; break;

case 3: display = 0x4F; break;

case 4: display = 0x66; break;

case 5: display = 0x6D; break;

case 6: display = 0x7D; break;

case 7: display = 0x07; break;

case 8: display = 0x7F; break;

case 9: display = 0x6F; break;

default: display = 0x79; break;

}

return display;

}

//Display coordinates on both 7-segment displays

void displaySegments() {

while(true){

display1 = displayHexValue(currentY);

display2 = displayHexValue(currentX);

}

}

//Update x,y coordinates depending on direction in motion

void updateCoordinates() {

if(isReversing){ //Check if robot is moving forwards or backwards

switch(direction) {

case 1: currentY-=1; break;

case 2: currentX-=1; break;

case 3: currentY+=1; break;

case 4: currentX+=1; break;

}

} else {

switch(direction) {

case 1: currentY+=1; break;

case 2: currentX+=1; break;

case 3: currentY-=1; break;

case 4: currentX-=1; break;

}

}

}

//Stops any robot movement and changes global bool variable to be false

void robotStopMovement() {

motorControl = 0x00;

inMotion = false;

isTurning = false;

isReversing = false;

isDancing = false;

}

//Main robot movement function

void robotMotorMovements(int motorMovement) {

switch(motorMovement) {

case 1: //Stop

motorControl = 0x00;

break;

case 2: //Left

motorControl = 0x09;

if(direction == 1) {

direction = 4;

break;

}

direction-=1;

break;

case 3: //Right

motorControl = 0x06;

if(direction == 4) {

direction = 1;

break;

}

direction+=1;

break;

case 4: //Forward

motorControl = 0x0A;

break;

case 5: //Reverse

motorControl = 0x05;

isReversing = true;

break;

}

}

//Function that continously runs to operate navigation and movement of the robot around the grid

void robotNavigation(){

while(true) {

//Verify if coordinates are meant to update

if(isDancing && timerController.read() > turnTime\*8) {

robotStopMovement();

timerController.stop();

timerController.reset();

}

if(inMotion && displayUpdateTimer.read() > movementTime/2) {

updateCoordinates();

displayUpdateTimer.stop();

displayUpdateTimer.reset();

}

//Check if enough time to have moved 8 inches has passed

if(inMotion && timerController.read() > movementTime) {

robotStopMovement();

timerController.stop();

timerController.reset();

}

//Check if enough time to turn 90 degrees has passed

if(isTurning && timerController.read() > turnTime) {

robotStopMovement();

timerController.stop();

timerController.reset();

}

//If robot is not in motion or is not turning, check switch statement for next set of instructions

if (!inMotion && !isTurning && enabledMovements) {

if(currentX == endX && currentY == endY){ //If robot has reached its destination, dance and stop

navigationTimer.stop();

enabledMovements = false;

enableTransmission = true;

isDancing = true;

timerController.start();

robotMotorMovements(2);

//Dance

startX = -1;

startY = -1;

endX = -1;

endY = -1;

} else { //Robot has not yet reached its destination

if( direction%2 == 1) { //Direction is either facing north or south (Faster to move by the y-axis)

int currentOffsetY = endY - currentY; //Find out distance needed across y-axis to reach destination

if(currentOffsetY > 0 ) { //If y distance is positive, robot must travel north

if(direction == 1) { //If already facing north, move forward

displayUpdateTimer.start();

timerController.start();

robotMotorMovements(4);

inMotion = true;

} else { //If facing south, just move reverse

displayUpdateTimer.start();

timerController.start();

robotMotorMovements(5);

inMotion = true;

}

} else if (currentOffsetY < 0) { //If y distance is negative, robot must travel south

if(direction == 1) { //If facing north, move reverse

displayUpdateTimer.start();

timerController.start();

robotMotorMovements(5);

inMotion = true;

} else { //If facing south, just move forward

displayUpdateTimer.start();

timerController.start();

robotMotorMovements(4);

inMotion = true;

}

} else { //This else statement assumes y distance is 0, thus robot must turn to face correct x axis

int currentOffsetX = endX - currentOffsetX; //Track which direction to turn

if (currentOffsetX > 0) { //X destination is towards the east

if(direction == 1) { //If facing north, turn right towards east

timerController.start();

robotMotorMovements(3);

isTurning = true;

} else { //If south, turn left to face east

timerController.start();

robotMotorMovements(2);

isTurning = true;

}

} else { //X is towards the west

if(direction == 1) { //If facing north, turn left towards west

timerController.start();

robotMotorMovements(2);

isTurning = true;

} else { //If south, turn right towards west

timerController.start();

robotMotorMovements(3);

isTurning = true;

}

}

}

} else { //Direction is either facing west or east (Faster to move by the x-axis)

int currentOffsetX = endX - currentX; //Find out distance needed across x-axis to reach destination

if(currentOffsetX > 0 ) { //If x distance is positive, robot must travel east

if(direction == 2) { //If already facing east, move forward

displayUpdateTimer.start();

timerController.start();

robotMotorMovements(4);

inMotion = true;

} else { //If facing west, just move reverse

displayUpdateTimer.start();

timerController.start();

robotMotorMovements(5);

inMotion = true;

}

} else if (currentOffsetX < 0) { //If x distance is negative, robot must travel west

if(direction == 2) { //If facing east, move reverse

displayUpdateTimer.start();

timerController.start();

robotMotorMovements(5);

inMotion = true;

} else { //If facing east, just move forward

displayUpdateTimer.start();

timerController.start();

robotMotorMovements(4);

inMotion = true;

}

} else { //This else statement assumes x distance is 0, thus robot must turn to face correct y axis

int currentOffsetY = endY - currentOffsetY; //Track which direction to turn

if (currentOffsetY > 0) { //Y destination is towards the north

if(direction == 2) { //If facing east, turn left towards north

timerController.start();

robotMotorMovements(2);

isTurning = true;

} else { //If west, turn right to face north

timerController.start();

robotMotorMovements(3);

isTurning = true;

}

} else { //Y destination is towards the south

if(direction == 2) { //If facing east, turn right towards south

timerController.start();

robotMotorMovements(3);

isTurning = true;

} else { //If west, turn left towards south

timerController.start();

robotMotorMovements(2);

isTurning = true;

}

}

}

}

}

}

}

}

// main() runs in its own thread in the OS

int main()

{

//Setting up reciever

#define TRANSFER\_SIZE 24

char txData[TRANSFER\_SIZE], rxData[TRANSFER\_SIZE];

int txDataCnt = 0;

int rxDataCnt = 0;

receiver.powerUp();

receiver.setRfOutputPower(-6);

receiver.setTxAddress((0x1D21372D90),DEFAULT\_NRF24L01P\_ADDRESS\_WIDTH);

receiver.setRxAddress((0x1D21372D90),DEFAULT\_NRF24L01P\_ADDRESS\_WIDTH);

receiver.setAirDataRate(2000);

// Display the (default) setup of the nRF24L01+ chip

printf( "nRF24L01+ Frequency : %d MHz\r\n", receiver.getRfFrequency() );

printf( "nRF24L01+ Output power : %d dBm\r\n", receiver.getRfOutputPower() );

printf( "nRF24L01+ Data Rate : %d kbps\r\n", receiver.getAirDataRate() );

printf( "nRF24L01+ TX Address : 0x%010llX\r\n", receiver.getTxAddress() );

printf( "nRF24L01+ RX Address : 0x%010llX\r\n", receiver.getRxAddress() );

receiver.setTransferSize( TRANSFER\_SIZE );

receiver.setReceiveMode();

receiver.enable();

//PWM for motors

ENA.period(0.05);

ENA=0.5;

ENB.period(0.05);

ENB=0.5;

//Movement time at 50% duty cycle for 8 inches + 90 degree-ish turn

movementTime = 0.9;

turnTime = 0.35;

//Initialize coords

startX = -1, startY = -1;

endX = -1, endY = -1;

currentX = -1, currentY = -1;

direction = -1;

//Initialize global variables

inMotion = false;

isTurning = false;

isReversing = false;

enableTransmission = false;

enabledMovements = false;

//Start threads

displayThread.start(displaySegments);

navigationThread.start(robotNavigation);

//Initialize bool to end while loop

bool endOfProgram = false;

timerController.start();

//While loop

while (true) {

if(receiver.readable()) {

rxDataCnt = receiver.read(NRF24L01P\_PIPE\_P0, rxData, sizeof(rxData));

printf("%s\n", rxData);

//Commands SC, EC, PA, ST, CT

if( rxData[0] == 'C' && rxData[1] == 'T'){ //Cancel transmission of results

enableTransmission = false;

}

if( rxData[0] == 'P' && rxData[1] == 'A'){ //Pause robot navigation

enabledMovements = false;

}

if( rxData[0] == 'E' && rxData[1] == 'C'){ //End coordinates for robot

if(endX != (rxData[2] - '0') && endY != (rxData[3] - '0') && enabledMovements == false) {

endX = rxData[2] - '0';

endY = rxData[3] - '0';

}

}

if( rxData[0] == 'S' && rxData[1] == 'C'){ //Starting coordinates + direction for robot

if(startX != (rxData[2] - '0') && startY != (rxData[3] - '0') && enabledMovements == false) {

startX = rxData[2] - '0';

startY = rxData[3] - '0';

direction = rxData[4] - '0';

currentX = startX;

currentY = startY;

}

}

if( rxData[0] == 'S' && rxData[1] == 'T'){ //Start command to start robot navigation

if(startX != -1 && endX != -1){

enabledMovements = true;

navigationTimer.start();

}

}

}

if(enableTransmission) { //Transmit navigation time to PC mbed board

printf("%i/100 seconds\n", int(navigationTimer.read()\*100));

snprintf(txData, 24, "%i/100", int(navigationTimer.read()\*100)); //Convert float to char array

receiver.write( NRF24L01P\_PIPE\_P0, txData, sizeof txData); //Broadcast array

enableTransmission = !enableTransmission;

navigationTimer.reset();

}

}

}