/\*

\* File: main.c

\* Author: Robert Stephenson / Pavlos Hanna

\*

\* Sensor into Analog pin 0

\* stepper lines 1-4 into pins 10, 30, 11, 31

\* ground in pin 3

\* 5V in pin 40

\* dip switches 1&2 on HMI for mode selection

\* dip switch 4 on micro for selecting continuous or discrete button input

\*

\* MX2: MXK Skeleton V2

\*/

#include <xc.h>

#include "ProcessorConfig.h"

#include "ISR.h"

#include "MXK.h"

#include "Config.h"

#include "Functions.h"

#include "Colours.h"

#include "Console.h"

#include <stdio.h>

#include "Motor.h"

#include "LCD.h"

#include "LED.h"

#include "HMI.h"

//--------------------------------ENTER YOUR GLOBAL VARS BELOW--------------------------//

int RST = 0; //requested steps

int RSP = 0; //requested speed

int MP = 0; //current motor position

int MPR; //motor position required

//variables used to store button states and to latch them

int Up, Down, Right, Left;

int R, L;

//variavles used to store dip switch states

int dip4m;

int dip1;

int dip2;

int dist; //to store dist from sensor to object

int x; //x used for loops

int y;

int z;

int w;

int min = 0;

int line;

int move;

//arrays used to make name in different colors

int col[] = {WHITE, BLUE, RED, GREEN, CYAN, YELLOW};

char let[] = {'P', 'a', 'v', 'l', 'o', 's'};

int CCW[512];

int CW[512];

UINT16 ADCVal = 0; //16 bit storage used to store raw ADC reading

UINT16 ADCM = 0; //16 bit storage used to store raw ADC reading

int sample = 10; //sample used to change the sample rate from the ADC

int Array[20];

int data;

int max;

int heart = 0;

float ADCValue;

float Distance;

int HighDist;

int LowDist;

int travel;

int DistTravelled = 0;

int Col;

int SongNo;

int VictimNo = 1;

//----------RG added this

int Mode;

int counter;

int checkNumber;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*added for A4\*\*\*\*\*\*\*\*\*\*\*\*\*/

int metre = 50;

int drop = 500;

int wall;

int scan;

int side;

int CCWF[20];

int CWF[20];

int safety;

int Sfact = 500;

int DistTravel;

int cond;

int angle;

int AngleL;

int angle2;

int Angle2;

int angle1;

int Angle1;

int angle0;

int Angle0;

int O=0;

int Sdist = 500;

//arrays used for conversion from raw ADC into distance

int mm[] = {100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800};

float ADC[] = {800, 550, 415, 340, 290, 260, 230, 210, 195, 182, 170, 164, 160, 155, 150};

//--------------------------------FINISH YOUR GLOBAL VARS ABOVE--------------------------//

void MCW() // This function should setup the required ports and subsystems for the operation of the motor

{

if (MXK\_SwitchTo(eMXK\_Motor)) {//---------------------------------ENTER YOUR CODE BELOW--------------------------------//

TRISG = 0x00; // sets all pins of register G as outputs

PORTG = 0x00; // sets all values of pins of G to zero

RSP = 2; // sets RSP as 2

for (x = 0; x < 1; x++) // loop to allow quick change to number of times coils activated per time called

{

PORTG = 0x01; // sets coils to 0001

delay\_ms(RSP); // delays RSP ms

PORTG = 0x02; // sets coils to 0010

delay\_ms(RSP); // delays RSP ms

PORTG = 0x04; // sets coils to 0100

delay\_ms(RSP); // delays RSP ms

PORTG = 0x08; // sets coils to 1000

delay\_ms(RSP); // delays RSP ms

PORTG = 0x00; // sets coils to 0000

}

x = 0; //sets x=0 for next loop

MP++; //incriments MP

}//---------------------------------FINISH YOUR CODE ABOVE--------------------------------//

if (MXK\_Release())

MXK\_Dequeue();

}

void MCCW() // This function should setup the required ports and subsystems for the operation of the motor

{

if (MXK\_SwitchTo(eMXK\_Motor)) {//---------------------------------ENTER YOUR CODE BELOW--------------------------------//

TRISG = 0x00; // sets all pins of register G as outputs

PORTG = 0x00; // sets all values of pins of G to zero

RSP = 2; // sets RSP as 2

for (x = 0; x < 1; x++) // loop to allow quick change to number of times coils activated per time called

{

PORTG = 0x08; // sets coils to 1000

delay\_ms(RSP); // delays RSP ms

PORTG = 0x04; // sets coils to 0100

delay\_ms(RSP); // delays RSP ms

PORTG = 0x02; // sets coils to 0010

delay\_ms(RSP); // delays RSP ms

PORTG = 0x01; // sets coils to 0001

delay\_ms(RSP); // delays RSP ms

PORTG = 0x00; // sets coils to 0000

}

x = 0; //sets x=0 for next loop

MP--; //decriments MS

}//---------------------------------FINISH YOUR CODE ABOVE--------------------------------//

if (MXK\_Release())

MXK\_Dequeue();

}

// HMI is broken into two parts to allow text to be entered within

void HMIO() {//starts writing to the HMI

printf("%c", ENDOFTEXT);

printf("Select Mode:\nUp: 5MetreMode\nDown: SquareMode\n");

}

void HMIC() {//ends writing to the HMI

//-------- re-initialise the buttons

/\*if ((Up == 1) || (Down == 1) || (Left == 1) || (Right == 1)) {

Up = 0;

Down = 0;

Left = 0;

Right = 0;

} \*/

Console\_Render(); //updates the screens

if (MXK\_Release())

MXK\_Dequeue();

}

void HMIdefaultmode() {

printf("%c", ENDOFTEXT);

printf("Select Mode:\nUp: 5MetreMode\nDown: SquareMode\n");

HMIC();

}

void IR() {

//ADC//

/\* if (sample <= 0) //uses variable called sample to take a reading once every 'sample' cycles of loop

{

delay\_ms(2);

//Task 2//

ADCON0bits.GO\_DONE = 1; //- Activate the ADC

while (ADCON0bits.GO\_DONE == 1)//- Wait for the ADC to complete

{

}

ADCVal = (ADRESH << 8) | ADRESL; //- Combine the results into a single value (ADCReading) and display

sample = 10; //sets value of 'sample' to 10

}

sample--; // decriments sample

ADCM = ADCVal; //changes name that ADC reading is stored in

for (x = 0; x <= 13; x++) //loops through arrays cm & ADC and does a copmarison between ADC reading then interpolates between correct points to find correct distance

{

if (ADCM <= ADC[x] && ADCM > ADC[x + 1]) //checks which pair of consequitive ADC values the reading is between.

{

MPR = mm[x + 1] - (ADCM - ADC[x + 1]) / (ADC[x] - ADC[x + 1])\*5; //interpolates between correct readings

}

}

x = 0; //resets x to 0 for next for loop

\*/

ADCON0bits.GO\_DONE = 1; // Turns ADC On

while (ADCON0bits.GO\_DONE == 1) // Waiting for Conversion of ADC to finish getting results

{

}

ADCValue = (ADRESH << 8) | ADRESL; // Takers Upper 2 bits of ADRESH and adds lower 8bits from ADRESL

MPR = ((27617 / (ADCValue + 29)) - 12)\*10; // Worked with friend to calculate Formula.

if (MXK\_BlockSwitchTo(eMXK\_HMI)) {

HMI\_SetNumber(MPR); //sets the distance that the ADC reading is equivilant to on the seven segment display

HMI\_Render(); // updates the HMI

if (MXK\_Release())

MXK\_Dequeue();

}

}

void HeartBeat() {

//delay\_ms(8);

if (PORTCbits.RC0 == 0) {

PORTCbits.RC0 = 1;

} else {

PORTCbits.RC0 = 0;

}

}

void checkReg() {

for (x = 0; x < max; x++) {

data = Array[x];

while (TXSTAbits.TRMT == 0) {

}

TXREG1 = data;

}

}

void AStartUp() {

Array[0] = 128;

Array[1] = 132;

Array[2] = 139;

Array[3] = 8;

Array[4] = 255;

Array[5] = 128;

max = 6;

checkReg();

}

void ARequestDist() {

Array[0] = 142;

Array[1] = 19;

max = 2;

checkReg();

}

void ANormalBeep(){

Array[0] = 140;

Array[1] = 0;

Array[2] = 2;

Array[3] = 80;

Array[4] = 64;

Array[5] = 128;

Array[6] = 64;

max = 7;

checkReg();

}

void AVictim1Song(){

Array[0] = 140;

Array[1] = 1;

Array[2] = 5;

Array[3] = 40;

Array[4] = 32;

Array[5] = Array[2]+10;

Array[6] = Array[3];

Array[7] = Array[4]+10;

Array[8] = Array[3];

Array[9] = Array[4];

Array[10] = Array[3];

Array[11] = Array[2];

Array[12] = Array[3];

max = 13;

checkReg();

}

void AVictim2Song(){

Array[0] = 140;

Array[1] = 2;

Array[2] = 3;

Array[3] = 60;

Array[4] = 32;

Array[5] = 80;

Array[6] = Array[3];

Array[7] = 40;

Array[8] = Array[3];

max = 9;

checkReg();

}

void ABump() {

Array[0] = 142;

Array[1] = 7;

max = 2;

checkReg();

}

void AVirtualWall(){

Array[0] = 142;

Array[1] = 13;

max = 2;

checkReg();

}

void AVictim(){

Array[0] = 142;

Array[1] = 17;

max = 2;

checkReg();

}

void ASRight() {

Array[0] = 137;

Array[1] = 1; // SPEED - 456 converted to hex split into 4 bytes 2 hi 2 low then converted individually into hi and low decimal THEN hi is put first then lo

Array[2] = 200;

Array[3] = 254; // Radius of curvature - 500 (to the right)

Array[4] = 12;

max = 5;

checkReg();

}

void ASRightS() {

Array[0] = 137;

Array[1] = 0; // SPEED - 200 converted to hex split into 4 bytes 2 hi 2 low then converted individually into hi and low decimal THEN hi is put first then lo

Array[2] = 200;

Array[3] = 254; // Radius of curvature - 500 (to the right)

Array[4] = 12;

max = 5;

checkReg();

}

void ASLeft() {

Array[0] = 137;

Array[1] = 1; // SPEED - 456 converted to hex split into 4 bytes 2 hi 2 low then converted individually into hi and low decimal THEN hi is put first then lo

Array[2] = 200;

Array[3] = 1; // Radius of curvature + 500mm (to the left)

Array[4] = 244;

max = 5;

checkReg();

}

void ASLeftS() {

Array[0] = 137;

Array[1] = 0; // SPEED - 456 converted to hex split into 4 bytes 2 hi 2 low then converted individually into hi and low decimal THEN hi is put first then lo

Array[2] = 200;

Array[3] = 1; // Radius of curvature + 500mm (to the left)

Array[4] = 244;

max = 5;

checkReg();

}

void ReadDisTravelled() {

ARequestDist();

while (RC1IF == 0) {

}

HighDist = RCREG1;

RC1IF = 0;

while (RC1IF == 0) {

}

LowDist = RCREG1;

RC1IF = 0;

travel = (HighDist << 8) | LowDist;

DistTravelled = DistTravelled + travel;

}

void AStop() {

Array[0] = 137;

Array[1] = 0;

Array[2] = 0;

Array[3] = 0;

Array[4] = 0;

max = 5;

checkReg();

}

void ABeep() {

Array[0] = 141;

Array[1] = SongNo;

max = 2;

checkReg();

}

void RL90() {

//------------- Drive settings set [137] then 1234 does speed and radius

Array[0] = 137;

Array[1] = 1;

Array[2] = 44;

Array[3] = 0;

Array[4] = 1;

//------------- Wait for Angle [157]

Array[5] = 157;

Array[6] = 0;

Array[7] = 85;

max = 8;

checkReg();

}

void collision() {

ABump();

while (RC1IF == 0) {

}

if (RCREG1 > 0) {

ABeep();

Col=1;

// DistTravelled = 6000;

}

AVirtualWall();

while (RC1IF == 0){}

if (RCREG1 == 1){

SongNo = 0;

ABeep();

RL90();

AStop();

Col=1;

// DistTravelled = 6000;

}

AVictim();

while (RC1IF == 0){}

if (RCREG1 == 254){

SongNo = VictimNo;

ABeep();

VictimNo++;

// DistTravelled = 6000;

}

/\* ACliffL();

while (RC1IF == 0) {

}

if (RCREG1 > 1) {

ABeep();

Col=1;

DistTravelled = 6000;

}

ACliffFL();

while (RC1IF == 0) {

}

if (RCREG1 > 1) {

ABeep();

Col=1;

DistTravelled = 6000;

}

ACliffFR();

while (RC1IF == 0) {

}

if (RCREG1 == 1) {

ABeep();

Col=1;

DistTravelled = 6000;

}

ACliffR();

while (RC1IF == 0) {

}

if (RCREG1 == 1) {

ABeep();

Col=1;

DistTravelled = 6000;

}

\*/

}

void ASForward() {

Array[0] = 137;

Array[1] = 1; // SPEED - 300 converted to hex split into 4 bytes 2 hi 2 low then converted individually into hi and low decimal THEN hi is put first then lo

Array[2] = 44;

Array[3] = 128; // Radius of curvature - 8000 is the hex value for straight 80 is turned into 128 while 00 is turned into 0

Array[4] = 0;

max = 5;

checkReg();

/\* Array[0] = 156;

Array[1] = 19;

Array[2] = 136;

max = 3;

checkReg();\*/

}

void ASBack() {

Array[0] = 137;

Array[1] = 255;

Array[2] = 56;

Array[3] = 1;

Array[4] = 244;

//Array[5]=128;

max = 5;

checkReg();

}

void MetreMode() {

ASForward();

DistTravel = DistTravelled;

while (DistTravelled - DistTravel < metre) {

ReadDisTravelled();

collision();

}

AStop();

}

void QRight() {

ASRightS();

DistTravel = DistTravelled;

while (DistTravelled - DistTravel < 780) {

ReadDisTravelled();

}

AStop();

// MetreMode();

}

void QLeft() {

ASLeftS();

DistTravel = DistTravelled;

while (DistTravelled - DistTravel < 780) {

ReadDisTravelled();

}

AStop();

// MetreMode();

}

void HMI5MetreMode() {

ASForward();

while (DistTravelled < 5000) {

ReadDisTravelled();

if (MXK\_BlockSwitchTo(eMXK\_HMI)) {//Write data to LCD

printf("Distance = %d\r", DistTravelled);

HMIC();

}

collision();

}

DistTravelled = 0;

AStop();

SongNo = 0;

ABeep(); //code to beep

HMIC();

}

void HMISquareMode() {

//x=0;

// for(x=0; x<=3; x++){

ASForward();

while (DistTravelled < 1000) {

ReadDisTravelled();

if (MXK\_BlockSwitchTo(eMXK\_HMI)) {//Write data to LCD

printf("Distance = %d\r", DistTravelled);

HMIC();

}

}

AStop();

RL90();

DistTravelled = 0;

ASForward();

while (DistTravelled < 1000) {

ReadDisTravelled();

if (MXK\_BlockSwitchTo(eMXK\_HMI)) {//Write data to LCD

printf("Distance = %d\r", DistTravelled);

HMIC();

}

}

AStop();

RL90();

DistTravelled = 0;

ASForward();

while (DistTravelled < 1000) {

ReadDisTravelled();

if (MXK\_BlockSwitchTo(eMXK\_HMI)) {//Write data to LCD

printf("Distance = %d\r", DistTravelled);

HMIC();

}

}

AStop();

RL90();

DistTravelled = 0;

ASForward();

while (DistTravelled < 1000) {

ReadDisTravelled();

if (MXK\_BlockSwitchTo(eMXK\_HMI)) {//Write data to LCD

printf("Distance = %d\r", DistTravelled);

HMIC();

}

}

AStop();

RL90();

DistTravelled = 0;

// }

AStop();

ABeep(); //code to beep

HMIC();

}

void Lfollow(){

IR();

if (MPR < 500) //if robot is moving towards wall

{

if (safety>500){

ASRight(); //code to turn robot to the right(away from wall) slightly

}

if (safety<=500){

ASRightS();

}

}

if (MPR > 500) //if robot is moving away from wall

{

if (MPR < 500 + drop)

{

if (safety>500){

ASLeft(); //code to turn robot to the right(away from wall) slightly

}

if (safety<=500){

ASLeftS();

}

}

if (MPR > 500 + drop)

{

wall=0;

}

}

}

void Rfollow(){

IR();

if (MPR < 500) //if robot is moving towards wall

{

ASLeft(); //code to turn robot to the left(away from wall) slightly

}

if (MPR > 500) //if robot is moving away from wall

{

if (MPR < 500 + drop)

{

ASRight(); //code to turn robot to the right (towards wall) slightly

}

if (MPR > 500 + drop)

{

wall=0;

}

}

}

void Oscan(){

if (O == 1)

{

for (y=0; y<=scan; y++)

{

MCW();

IR();

CW[y] = MPR;

}

y=0;

for (y=0; y<scan; y++)

{

if (CW[y+1]<= CW[min])

{

min = y+1;

}

}

y=0;

min = scan + 1 - min;

for (y=0; y<= min; y++)

{

MCCW();

}

y=0;

O =0;

}

else if (O == 0)

{

for (y = 0; y <= scan; y++)//code to rotate through 360 degrees

{

MCCW(); //turns motor CCW

IR(); //reads IR sensor and updates value on SSD

CCW[y] = MPR; //stores IR value (which is in MPR) in CCW array

}

y = 0;

for (y = 0; y < scan; y++) //goes through stored values and finds min

{

if (CCW[y + 1] <= CCW[min]) //if new min is found

{

min = y + 1; //store location of min

}

}

y = 0;

min = scan + 1 - min; //calculates how far from min position IR sensor is

for (y = 0; y <= min; y++) // repeats number of times requires to point to minimum

{

MCW(); // turns motor CW

}

y = 0;

O =1;

}

}

void FCCW(){

for (y=0; y<=scan; y++)

{

MCCW();

}

y=0;

IR();

min = 0;

CCWF[min] = MPR;

safety = CCWF[min]-Sfact;

if (safety <= 0){

RL90();

AStop();

}

for (y=0; y<=scan; y++)

{

MCW();

}

y=0;

}

void FCW(){

for (y=0; y<=scan; y++)

{

MCW();

}

y=0;

IR();

min = 0;

CWF[min] = MPR;

safety = CWF[min]-Sfact;

if (safety <= 0){

RL90();

AStop();

}

for (y=0; y<=scan; y++)

{

MCCW();

}

y=0;

}

/\*void FCW(){

for (y=0; y<=scan-10; y++)

{

MCW();

}

for (y=0; y<=20; y++)

{

MCW();

IR();

CWF[y] = MPR;

}

y=0;

for (y=0; y<20; y++)

{

if (CWF[y+1]<= CWF[min])

{

min = y+1;

}

}

y=0;

safety = CWF[min]-Sfact;

if (safety <= 0){

RL90();

AStop();

}

// min = scan + 1 - min;

for (y=0; y<= scan + 10; y++)

{

MCCW();

}

y=0;

// ABeep();

}\*/

/\*void FCCW(){

for (y=0; y<=scan-10; y++)

{

MCCW();

}

for (y=0; y<=20; y++)

{

MCCW();

IR();

CCWF[y] = MPR;

}

y=0;

for (y=0; y<20; y++)

{

if (CCWF[y+1]<= CCWF[min])

{

min = y+1;

}

}

y=0;

safety = CCWF[min]-Sfact;

if (safety <= 0){

RL90();

AStop();

}

//min = scan + 1 - min;

for (y=0; y<= scan + 10; y++)

{

MCW();

}

y=0;

//ABeep();

}\*/

void F(){

wall = 1;

if (side == 1){

FCW();

// condition to exit if distance to front wall is lower that clearance limit use distance traveled since last called

// while (Col != 1) {

// while (wall != 0){

if (Col != 1){

cond = 1\*wall\*safety;

}

if (Col == 1){

cond = 0;

Col = 0;

}

while (cond >0){

ReadDisTravelled();

safety = safety - travel;

Lfollow();

collision();

if (Col != 1){

cond = 1\*wall\*safety;

}

if (Col == 1){

cond = 0;

safety = 0;

// Col=0;

}

} AStop();

// }

// }

if (wall == 0){

QLeft();

safety = 0;

wall = 1;}

}

if (side == 0){

FCCW();

// condition to exit if distance to front wall is lowwer that clearance limit

// while (Col != 1) {

if (Col != 1){

cond = 1\*wall\*safety;

}

if (Col == 1){

cond = 0;

Col = 0;

}

// while (wall != 0){

// safety = 1000;

while (cond >0){

ReadDisTravelled();

safety =safety - travel;

Rfollow();

collision();

if (Col != 1){

cond = 1\*wall\*safety;

}

if (Col == 1){

cond = 0;

safety = 0;

// Col =0;

}

} AStop();

// }

//}

if (wall == 0){

QRight();

safety = 0;

wall = 1;}

}

if (Col == 1){

RL90();

// RL90();

AStop();

Col = 0;

}

}

///////////////////////////////////////////////////////////////////////////////////////////////////////

/////////////////////////////////END FUNCTIONS, START MAIN PROGRAM/////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////

void main() {

//Configures the initial conditions required for safe and correct operation of the MXK.

MXK\_Init(); // initiates the MXK

ISR\_Enable(); // enables the ISR

FunctInitADC(); // initiates the ADC

//Init stepper

Motor Stepper;

if (MXK\_BlockSwitchTo(eMXK\_Motor)) {

TRISG = 0x00; // sets all pins of register G as outputs

PORTG = 0x00; // sets all values of pins of G to zero

if (MXK\_Release())

MXK\_Dequeue();

}

//Init ADC//

//Task 1//

TRISA = 0xFF; //sets port A as inputs

WDTCONbits.ADSHR = 1; //sets the ADSHR to 1

ANCON0 = 0x00; //sets ANCON0 to 0

ANCON1 = 0x00; //sets ANCON1 to 0

WDTCONbits.ADSHR = 0; //sets the ADSHR to 0

ADIE = 0; //- Disable ADC related interrupts (we don't want these stopping our program)

ADCON1 = 0xBF; //- Setup to be right justified- Disable calibration- Set acquisition time to 20 TADSet conversion clock rate to FRC

ADSHR = 1;

//- Set relevant input channels to analog

ADSHR = 0;

//- Set voltage references to Vdd and Vss (5V and 0V respectively)

//- Make sure the ADC is idle

//- Clear ADC interrupt flag

//Setup channels

//- Switch the ADC on

ADCON0 = 0x01;

ADCON0bits.ADON = 1;

if (MXK\_BlockSwitchTo(eMXK\_HMI)) {

LCD\_Init(); //calls LCD\_Init

Console\_Init(); //calls Console\_Init

HMI\_Init(); //calls HMI\_Init

if (MXK\_Release())

MXK\_Dequeue();

}

INTCONbits.GIE = 0; //sets the global interupt from the intcon register to 0

TRISDbits.RD3 = 1; // sets bit 3 of register D as an input for dip4 on microcontroler

TRISEbits.RE2 = 1; // sets bit 2 of register E as an input for dip2 on HMI

TRISEbits.RE3 = 1; // sets bit 3 of register E as an input for dip1 on HMI

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*above is old code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Set correct pin directions (refer to schematic) Hint: TX=OUT and RX=IN

TRISCbits.RC0 = 0;

TRISCbits.RC7 = 1;

TRISCbits.RC6 = 0;

TXSTA1bits.TX9 = 0; //Set correct number of transmission bits (refer to iRobot user manual)

TXSTA1bits.TXEN = 1; //Enable transmission

TXSTA1bits.SYNC = 0; //Decide on transmission mode

TXSTA1bits.SENDB = 0; //Disable sync break characters

RCSTA1bits.SPEN = 1; //Enable the serial port

RCSTA1bits.RX9 = 0; //Set correct number of reception bits (same as transmission)

RCSTA1bits.CREN = 1; //Enable continuous reception

//Disable any framing or overrun errors

RCSTA1bits.FERR = 0;

RCSTA1bits.OERR = 0;

BAUDCON1bits.DTRXP = 0; //Disable data inversion

BAUDCON1bits.SCKP = 0; //Idle state is high

BAUDCON1bits.WUE = 0; //Disable monitoring of the RX pin for wake up

BAUDCON1bits.ABDEN = 0; //Disable automatic baud detection

//Task 2 - Set the baud rate (bits per second)

//

//Check the iRobot user guide for the required baud rate and note it down

//Use the formulas in the datasheet (table 20-1) to calculate the required value for SPBRG

//Ensure that BRGH and BRG16 are set according to the formula you chose to use

//Calculate the baud rate error using the formulas in the datasheet (example 20-1)

SPBRG1 = 12;

TXSTA1bits.BRGH = 0;

BAUDCON1bits.BRG16 = 0;

AStartUp();

/\*writing song for beep\*/

ANormalBeep();

AVictim1Song();

AVictim2Song();

/////////////////////////////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////// MAIN LOOP ///////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////////////////////////////

loop() {

dip2 = PORTEbits.RE2; // gets the state of dip2 and stores in dip2 by the value of bit 2 of register E

dip1 = PORTEbits.RE3; // similar to above

dip4m = PORTDbits.RD3; // similar to above except from different register

IR();

if (heart == 0) {

HeartBeat();

heart = 150;

}

heart--;

if (MXK\_BlockSwitchTo(eMXK\_HMI)) //Read button states

{

HMI\_Poll();

Down = HMIBoard.mDown.mGetState();

Up = HMIBoard.mUp.mGetState();

Left = HMIBoard.mLeft.mGetState();

Right = HMIBoard.mRight.mGetState();

if (dip4m == 1) // if descrete mode is pressed it gets stuck in a do while loop until button is released, hence only incrimenting once.

{

if (Down == 1) {

do {

HMIBoard.mDown.mGetState();

} while (Down == HMIBoard.mDown.mGetState());

}

if (Up == 1) {

do {

HMIBoard.mUp.mGetState();

} while (Up == HMIBoard.mUp.mGetState());

}

if (Left == 1) {

do {

HMIBoard.mLeft.mGetState();

} while (Left == HMIBoard.mLeft.mGetState());

}

if (Right == 1) {

do {

HMIBoard.mRight.mGetState();

} while (Right == HMIBoard.mRight.mGetState());

}

}

if (MXK\_Release())

MXK\_Dequeue();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*NO operation\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (dip1 == 0 && dip2 == 0) //if no dipswitch is selected then go to NO opperation mode

{

if (MXK\_BlockSwitchTo(eMXK\_HMI)) {//Write data to LCD

//HMIO();

//printf("Press button");

//HMIC();

switch (Mode) {

case 1:

printf("%c", ENDOFTEXT);

printf("EXTERMINTATE. FORWARD. DIE. \nBZZT BZZT\nEXTERMINTATE. FORWARD.");

break;

case 2:

printf("%c", ENDOFTEXT);

printf("SEARCHING \nBZZT BZZT\nFOR ENEMIES");

break;

case 3:

case 4:

default:

HMIdefaultmode();

break;

}

//CheckThatNumber();

//printf("d1 = %d d2 = %d d4 = %d", dip1, dip2, dip4m);

//printf("use for calibration\n");

//printf("NumCheck = %d\n", checkNumber); //outputs the ADC raw reading for calibration

HMIC();

}

/\* if (Up == 1) // drive 5m task

{

{

counter++; // lowers delay in motor

if (counter == 40) {

if (Mode != 1) {

Mode = 1;

delay\_ms(10);

HMI5MetreMode();

}

counter = 0;

Mode = 0;

}

}

}

if (Left == 1) // scan CCW task

{

for (y = 0; y <= 511; y++)//code to rotate through 360 degrees

{

MCCW(); //turns motor CCW

IR(); //reads IR sensor and updates value on SSD

CCW[y] = MPR; //stores IR value (which is in MPR) in CCW array

}

y = 0;

for (y = 0; y <= 510; y++) //goes through stored values and finds min

{

if (CCW[y + 1] <= CCW[min]) //if new min is found

{

min = y + 1; //store location of min

}

}

y = 0;

min = 512 - min; //calculates how far from min position IR sensor is

if (MXK\_BlockSwitchTo(eMXK\_HMI)) //Write data to LCD

{

HMIO();

printf("d1 = %d d2 = %d d4 = %d", dip1, dip2, dip4m);

printf("Mode is not selected\n");

printf("use for calibration\n");

printf("min dist = %d\n", CCW[min]); //outputs the ADC raw reading for calibration

HMIC();

}

for (y = 0; y <= min; y++) // repeats number of times requires to point to minimum

{

MCW(); // turns motor CW

}

y = 0;

min = 128;

angle= min/512 \*360;

if (angle>360)

{angle = angle -360;}

if (angle<0)

{angle = angle +360;}

angle2 = (angle%256);

Angle2 = angle/256;

angle1 = (angle2%16);

Angle1 = angle2/16;

angle0 = (angle1%1);

Angle0 = angle1/1;

AngleL=16\*Angle1 + Angle0;

Array[0] = 137;

Array[1] = 1;

Array[2] = 44;

Array[3] = 0;

Array[4] = 1;

//------------- Wait for Angle [157]

Array[5] = 157;

Array[6] = Angle2;

Array[7] = AngleL;

max = 8;

checkReg();

AStop();

for (y = 0; y <= 511; y++)//code to rotate through 360 degrees

{

MCCW(); //turns motor CCW

IR(); //reads IR sensor and updates value on SSD

CCW[y] = MPR; //stores IR value (which is in MPR) in CCW array

}

y = 0;

for (y = 0; y <= 510; y++) //goes through stored values and finds min

{

if (CCW[y + 1] <= CCW[min]) //if new min is found

{

min = y + 1; //store location of min

}

}

y = 0;

min = 512 - min; //calculates how far from min position IR sensor is

if (MXK\_BlockSwitchTo(eMXK\_HMI)) //Write data to LCD

{

HMIO();

printf("d1 = %d d2 = %d d4 = %d", dip1, dip2, dip4m);

printf("Mode is not selected\n");

printf("use for calibration\n");

printf("min dist = %d\n", CCW[min]); //outputs the ADC raw reading for calibration

HMIC();

}

for (y = 0; y <= min; y++) // repeats number of times requires to point to minimum

{

MCW(); // turns motor CW

}

y = 0;

//code to turn 90 deg right

//code to turn sensor to face wall

//code to make robot move forward

while (Col != 1) {

IR();

collision();

if (MPR < CCW[min]) //if robot is moving towards wall

{

ASRight(); //code to turn robot to the right(away from wall) slightly

if (MXK\_BlockSwitchTo(eMXK\_HMI)) //Write data to LCD

{

HMIO();

printf("d1 = %d d2 = %d d4 = %d", dip1, dip2, dip4m);

printf("Mode is not selected\n");

printf("use for calibration\n");

printf("turn right"); //outputs the ADC raw reading for calibration

HMIC();

}

}

if (MPR > CCW[min]) //if robot is moving away from wall

{

ASLeft(); //code to turn robot to the left (towards wall) slightly

if (MXK\_BlockSwitchTo(eMXK\_HMI)) //Write data to LCD

{

HMIO();

printf("d1 = %d d2 = %d d4 = %d", dip1, dip2, dip4m);

printf("Mode is not selected\n");

printf("use for calibration\n");

printf("turn left"); //outputs the ADC raw reading for calibration

HMIC();

}

}

}

Col = 0;

AStop();

}

if (Right == 1) // scan CW task simmilar comments to if (Left ==1) just with CCW & CW vise versa

{

for (y=0; y<=511; y++)

{

MCW();

IR();

CW[y] = MPR;

if (MXK\_BlockSwitchTo(eMXK\_HMI)) //Write data to LCD

{

HMIO();

printf("d1 = %d d2 = %d d4 = %d", dip1, dip2, dip4m);

printf("Mode is not selected\n");

printf("use for calibration\n");

printf("ADCVal = %d\n", ADCVal); //outputs the ADC raw reading for calibration

printf("x= %d\n", y);

HMIC();

}

}

y=0;

if (MXK\_BlockSwitchTo(eMXK\_HMI)) //Write data to LCD

{

HMIO();

printf("d1 = %d d2 = %d d4 = %d", dip1, dip2, dip4m);

printf("Mode is not selected\n");

printf("use for calibration\n");

printf("ADCVal = %d\n", ADCVal); //outputs the ADC raw reading for calibration

printf("done\n");

HMIC();

}

for (y=0; y<=510; y++)

{

if (CW[y+1]<= CW[min])

{

min = y+1;

}

}

y=0;

min = 512 - min;

for (y=0; y<= min; y++)

{

MCCW();

}

y=0;

ABeep();

}

}

else {

printf("Mode is not selected\n");

Mode = 0;

}\*/

if (Up == 1){

scan = 511;

Oscan();

y=0;

for (y=0; y<=0; y++)

{

MCCW();

}

y=0;

}

if (Right == 1){

MCW();

}

}

if (Left == 1){

MCCW();

}

if (Down ==1){

// scan = 511;

// Oscan();

scan = 128;

side = 0;

F();

while (safety<=0) {

F();

}

while(1);

{

RL90();

MetreMode();

}

Col = 0;

AStop();

}

}

}