**SE 423: Mechatronics (Homework Project) Ayush Sinha, ayush2@, 677549575**

**user\_proj\_ver1.c**  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Demonstration of Localization and Mapping

using Minimal and Inexpensive Components

(For SE 243: Mechatronics mini project)

Author: Ayush Sinha

MS Mechanical Engineering,

UIUC, ayush7.sinha@gmail.com

Date: 15 April 2018

MSP430G2553

------------------------

/|\| XIN|-

| | |

--|RST XOUT|-

9600 ------>|P1.1/UCA0RXD P1.7/A7|------>Photo-resistor

8N1 <------|P1.2/UCA0TXD P1.6/A6|------>Sharp GP2Y0A21YK0F (Front IR)

Sharp GP2Y0A21YK0F (Right IR)------>|P1.5/A5 |

A4973SLBT (Left)<------|P2.2/TA1.1 P2.4/TA1.2|------>A4973SLBT (Right)

World

---------------------

| 15 | 14 | 13 | 12 |

---------------------

| 8 | 9 | 10 | 11 |

---------------------

| 7 | 6 | 5 | 4 |

---------------------

| 0 | 1 | 2 | 3 |

---------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MSP430G2553 Project Creator

SE 423 - Dan Block

Spring(2017)

Written(by) : Steve(Keres)

College of Engineering Control Systems Lab

University of Illinois at Urbana-Champaign

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#include** "msp430g2553.h"

**#include** "UART.h"

**#define** PWM\_FORWARD 475

**#define** PWM\_BACKWARD 325

**#define** PWM\_STOP 400

**#define** MAX\_TURN 30

**#define** PWM\_TURN\_SPEED 100

**#define** ONE\_CELL\_TIME 1200 // ms

**#define** TURNING\_TIME 900 // ms

**#define** PAUSE\_CONTROL\_TIME 1000 //ms

**#define** PAUSE\_FORWARD 1000 //ms

**#define** PHOTOR\_THRESHOLD 80

// desired traj for exploration stored in explore\_traj.c

**extern** **int** explore\_traj[7][3];

**char** newprint = 0; // flags main() while loop to print

**int** timecheck = 0; // keeps time

**int** rightIRref\_timer = 0; // timer used for stopping and calculating right IR reference value for right-wall following after each turn or when prog begins

**int** timerV = 0; // record time for each step

// Sensors

**int** sensors[8]; // array to hold ADC values

// current and 19 older values stored for all 3 sensors

**int** photoR[20] = {0}; // photo-resistor for detecting occupied cell

**int** front\_IR[20] = {0}; // IR proximity sensor in front

**int** right\_IR[20] = {0}; // IR proximity sensor on right side

**int** right\_IR\_raw = 0; // not averaged value for reference calculation

**int** front\_IR\_raw = 0; // not averaged value for wall detection

**float** right\_IR\_ref = 0; // averaged value used as reference for right-wall following

// right wall following to drive straight

**int** turn = 0; // control input for wall following (PWM units)

**float** K\_turn = 0.2; // Proportional gain Right Wall following

// flags used for driving on traj

**char** first\_cmd = 1; // flags that program just started

**char** first\_s\_cmd = 0; // flags that car moves forward for the first time after a stop

**char** dir\_change = 1; // flags that direction of motion changed (car has made a turn)

**char** car\_stop = 0; // flag to stop car (highest priority)

**char** drive\_mode = 's'; // forward, right, left, stop

**char** move\_type = 'x'; // flags state-machine

**char** start\_timer = 0; // start timerV only after right\_IR\_ref is calculated

// trajectory information

**int** cell\_cnt = 0; // # of cells to move forward in a step

**char** which\_turn = 'x'; // left or right - set by explore\_traj

**int** traj\_step\_num = 0; // tracks which step of explore traj is being implemented

**int** start\_cell = 0; // first cell of each traj step

// reference or desired location

**int** ref\_loc = 0; // car location by reference traj (explore\_traj)

**int** theta = 0; // records angle of car by counting # of turns

**int** heading = 0; // theta limited to 0-3, 0=0,1=pi/2,2=pi,3=3pi/2

// observed location (using only IR readings)

**char** row = 0;

**char** row\_old = 0;

**char** col = 0;

**char** col\_old = 0;

**int** obs\_loc = 0; // observed location

**int** obs\_loc\_new = 0;

**int** obs\_loc\_old = 0; // old observed location

**int** rIR\_old = 585; // right IR value at obs\_loc\_old

**int** fIR\_old = 800; // front IR value at obs\_loc\_old

// Filtered Location (Kalman filter to calculate expected location)

**float** loc\_predicted = 0; // location after prediction step

**float** kalman\_loc = 0; // location after update step

**float** kalman\_loc\_old = 0; // older updated locaion

**int** filter\_loc = 0; // discretized kalman\_loc

**float** step\_size = 0; // dist. traveled at each call calculated using velocity model (used for prediction)

**float** K\_kf = 0.01; // kalman gain (very low value here as IR readings are found to be unreliable)

// mapping

**char** obstacle\_found = 0; // flag that curr cell is occupied or blackened

**int** obs\_count = 0; // counts # of consecutive photo resistor readings below threshold

// sets PWM for different modes - forward, right, left etc.

**void** **drive\_car**(**char** mode){

**int** left, right; // left motor PWM, right motor PWM

**switch**(mode){ // mode set by drive\_mode

**case** 's': // straight forward using right wall following

**if** (turn > MAX\_TURN) turn = MAX\_TURN; // turn saturation and

**if** (turn < -MAX\_TURN) turn = -MAX\_TURN; // avoid random IR spikes

left = PWM\_FORWARD - turn; // right wall following

right = PWM\_FORWARD + turn;

**break**;

**case** 'l': // left turn

left = PWM\_STOP - PWM\_TURN\_SPEED;

right = PWM\_STOP + PWM\_TURN\_SPEED;

**break**;

**case** 'r': // right turn

left = PWM\_STOP + PWM\_TURN\_SPEED;

right = PWM\_STOP - PWM\_TURN\_SPEED;

**break**;

**case** 'x': // pause when program begins

left = PWM\_STOP;

right = PWM\_STOP;

**break**;

**default**:

left = PWM\_STOP;

right = PWM\_STOP;

}

// saturation

**if** (left > PWM\_FORWARD) left = PWM\_FORWARD;

**if** (left < PWM\_BACKWARD) left = PWM\_BACKWARD;

**if** (right > PWM\_FORWARD) right = PWM\_FORWARD;

**if** (right < PWM\_BACKWARD) right = PWM\_BACKWARD;

// set PWMs for phase pins of H-bridges for both motors

TA1CCR1 = left;

TA1CCR2 = right;

}

// gets observed location (using IR reading)

**void** **observed\_loc**(**void**){

/\* matching current IR values to row and col IR values pre-determined

\* to get cell location. Failed as IR values are unreliable when car

\* is not horizontally or vertically oriented \*/

// switch (heading){

// case 0:

// // row->right\_IR; col->front\_IR

// if ((right\_IR[0] >= 578) && (right\_IR[0] <= 590)) row = 0;

// else if ((right\_IR[0] >= 708) && (right\_IR[0] <= 718)) row = 1;

// else if ((right\_IR[0] >= 741) && (right\_IR[0] <= 755)) row = 2;

// //else if ((right\_IR[0] >= 716) && (right\_IR[0] <= 735)) row = 3;

// else row = row\_old;

//

// //if ((front\_IR[0] >= 756) && (front\_IR[0] <= 780)) col = 0;

// if ((front\_IR[0] >= 753) && (front\_IR[0] <= 761)) col = 1;

// else if ((front\_IR[0] >= 710) && (front\_IR[0] <= 716)) col = 2;

// else if ((front\_IR[0] >= 463) && (front\_IR[0] <= 475)) col = 3;

// else col = col\_old;

//

// break;

//

// case 1:

// // row->front\_IR; col->right\_IR

// //if ((right\_IR[0] >= 723) && (right\_IR[0] <= 750)) col = 0;

// if ((right\_IR[0] >= 736) && (right\_IR[0] <= 756)) col = 1;

// else if ((right\_IR[0] >= 708) && (right\_IR[0] <= 724)) col = 2;

// else if ((right\_IR[0] >= 574) && (right\_IR[0] <= 600)) col = 3;

// else col = col\_old;

//

// //if ((front\_IR[0] >= 756) && (front\_IR[0] <= 780)) row = 0;

// if ((front\_IR[0] >= 750) && (front\_IR[0] <= 760)) row = 1;

// else if ((front\_IR[0] >= 700) && (front\_IR[0] <= 712)) row = 2;

// else if ((front\_IR[0] >= 440) && (front\_IR[0] <= 460)) row = 3;

// else row = row\_old;

//

// break;

//

// case 2:

// // row->right\_IR; col->front\_IR

// //if ((right\_IR[0] >= 570) && (right\_IR[0] <= 690)) row = 0;

// if ((right\_IR[0] >= 732) && (right\_IR[0] <= 753)) row = 1;

// else if ((right\_IR[0] >= 702) && (right\_IR[0] <= 716)) row = 2;

// else if ((right\_IR[0] >= 570) && (right\_IR[0] <= 580)) row = 3;

// else row = row\_old;

//

// if ((front\_IR[0] >= 426) && (front\_IR[0] <= 436)) col = 0;

// else if ((front\_IR[0] >= 700) && (front\_IR[0] <= 710)) col = 1;

// else if ((front\_IR[0] >= 749) && (front\_IR[0] <= 760)) col = 2;

// //else if ((front\_IR[0] >= 400) && (front\_IR[0] <= 480)) col = 3;

// else col = col\_old;

//

// break;

//

// case 3:

// // row->front\_IR; col->right\_IR

// if ((right\_IR[0] >= 575) && (right\_IR[0] <= 595)) col = 0;

// else if ((right\_IR[0] >= 706) && (right\_IR[0] <= 725)) col = 1;

// else if ((right\_IR[0] >= 737) && (right\_IR[0] <= 760)) col = 2;

// //else if ((right\_IR[0] >= 500) && (right\_IR[0] <= 600)) col = 3;

// else col = col\_old;

//

// if ((front\_IR[0] >= 440) && (front\_IR[0] <= 464)) row = 0;

// else if ((front\_IR[0] >= 700) && (front\_IR[0] <= 711)) row = 1;

// else if ((front\_IR[0] >= 750) && (front\_IR[0] <= 760)) row = 2;

// //else if ((front\_IR[0] >= 400) && (front\_IR[0] <= 500)) row = 3;

// else row = row\_old;

//

// break;

//

// default:

// row = row\_old;

// col = col\_old;

// }

// Past State Aware Observation Model

/\* Assume that car cannot change rows or cols by more than 1

\* between 2 consecutive calls of observed\_loc() (i.e 10 ms)\*/

**switch** (heading){

**case** 0: // car heading is 0 degrees

// row->right\_IR

**if** (row\_old == 0){

**if** (right\_IR[0] >= 590) {

row = 1;

rIR\_old = right\_IR[0];

}

}

**else** **if** (row\_old == 1){

**if** (right\_IR[0] <= 708) {

row = 0;

rIR\_old = right\_IR[0];

}

**else** **if** (right\_IR[0] >= 718) {

row = 2;

rIR\_old = right\_IR[0];

}

}

**else** **if** (row\_old == 2){

**if** (right\_IR[0] <= 741) {

row = 1;

rIR\_old = right\_IR[0];

}

**else** **if** (right\_IR[0] >= 718) {

row = 3;

rIR\_old = right\_IR[0];

}

}

**else** **if** (row\_old == 3){

**if** ((right\_IR[0] - rIR\_old >= 10) || (rIR\_old - right\_IR[0] >= 10)) {

row = 2;

rIR\_old = right\_IR[0];

}

}

**else** row = row\_old;

// col->front\_IR

**if** (col\_old == 0){

**if** ((front\_IR[0] - fIR\_old >= 20) || (fIR\_old - front\_IR[0] >= 20)) col = 1;

}

**else** **if** (col\_old == 1){

**if** (front\_IR[0] <= 753) {

col = 2;

fIR\_old = front\_IR[0];

}

}

**else** **if** (col\_old == 2){

**if** (front\_IR[0] <= 710) {

col = 3;

fIR\_old = front\_IR[0];

}

}

**else** col = col\_old;

**break**;

**case** 1: // car heading is 90 degrees

// col->right\_IR

**if** (col\_old == 0){

**if** ((right\_IR[0] - rIR\_old >= 20) || (rIR\_old - right\_IR[0] >= 20)) {

col = 1;

rIR\_old = right\_IR[0];

}

}

**else** **if** (col\_old == 1){

**if** (right\_IR[0] <= 736) {

col = 2;

rIR\_old = right\_IR[0];

}

**else** **if** (right\_IR[0] >= 756) {

col = 0;

rIR\_old = right\_IR[0];

}

}

**else** **if** (col\_old == 2){

**if** (right\_IR[0] <= 708) {

col = 3;

rIR\_old = right\_IR[0];

}

**else** **if** (right\_IR[0] >= 724) {

col = 1;

rIR\_old = right\_IR[0];

}

}

**else** **if** (col\_old == 3){

**if** (right\_IR[0] >= 600) {

col = 2;

rIR\_old = right\_IR[0];

}

}

**else** col = col\_old;

// row->front\_IR

**if** (row\_old == 0){

**if** ((front\_IR[0] - fIR\_old >= 10) || (fIR\_old - front\_IR[0] >= 10)) {

row = 1;

fIR\_old = front\_IR[0];

}

}

**else** **if** (row\_old == 1){

**if** (front\_IR[0] <= 750) {

row = 2;

fIR\_old = front\_IR[0];

}

}

**else** **if** (row\_old == 2){

**if** (front\_IR[0] <= 700) {

row = 3;

fIR\_old = front\_IR[0];

}

}

**else** row = row\_old;

**break**;

**case** 2: // car heading is 180 degrees

// row->right\_IR

**if** (row\_old == 0){

**if** ((right\_IR[0] - rIR\_old >= 10) || (rIR\_old - right\_IR[0] >= 10)) {

row = 1;

rIR\_old = right\_IR[0];

}

}

**else** **if** (row\_old == 1){

**if** (right\_IR[0] <= 732) {

row = 2;

rIR\_old = right\_IR[0];

}

**else** **if** (right\_IR[0] >= 753) {

row = 0;

rIR\_old = right\_IR[0];

}

}

**else** **if** (row\_old == 2){

**if** (right\_IR[0] <= 702) {

row = 3;

rIR\_old = right\_IR[0];

}

**else** **if** (right\_IR[0] >= 716) {

row = 1;

rIR\_old = right\_IR[0];

}

}

**else** **if** (row\_old == 3){

**if** (right\_IR[0] >= 580) {

row = 2;

rIR\_old = right\_IR[0];

}

}

**else** row = row\_old;

// col->front\_IR

**if** (col\_old == 0){

**if** (front\_IR[0] >= 436) {

col = 1;

fIR\_old = front\_IR[0];

}

}

**else** **if** (col\_old == 1){

**if** (front\_IR[0] <= 700) {

col = 0;

fIR\_old = front\_IR[0];

}

}

**else** **if** (col\_old == 2){

**if** (front\_IR[0] <= 749) {

col = 1;

fIR\_old = front\_IR[0];

}

}

**else** **if** (col\_old == 3){

**if** ((front\_IR[0] - fIR\_old >= 20) || (fIR\_old - front\_IR[0] >= 20)) {

col = 2;

fIR\_old = front\_IR[0];

}

}

**else** col = col\_old;

**break**;

**case** 3: // car heading is 270 degrees

row = row\_old;

// col->right\_IR

**if** (col\_old == 0){

**if** (right\_IR[0] >= 595) {

col = 1;

rIR\_old = right\_IR[0];

}

}

**else** **if** (col\_old == 1){

**if** (right\_IR[0] <= 706) {

col = 0;

rIR\_old = right\_IR[0];

}

**else** **if** (right\_IR[0] >= 725) {

col = 2;

rIR\_old = right\_IR[0];

}

}

**else** **if** (col\_old == 2){

**if** (right\_IR[0] <= 737) {

col = 1;

rIR\_old = right\_IR[0];

}

**else** **if** (right\_IR[0] >= 760) {

col = 3;

rIR\_old = right\_IR[0];

}

}

**else** **if** (col\_old == 3){

**if** ((right\_IR[0] - rIR\_old >= 20) || (rIR\_old - right\_IR[0] >= 20)) {

col = 2;

rIR\_old = right\_IR[0];

}

}

**else** col = col\_old;

**break**;

**default**:

row = row\_old;

col = col\_old;

}

// calculating loc based on row and col

**if** (row == 0) obs\_loc\_new = col;

**else** **if** (row == 1) obs\_loc\_new = 7 - col;

**else** **if** (row == 2) obs\_loc\_new = col + 8;

**else** obs\_loc\_new = 15 - col;

obs\_loc = obs\_loc\_new;

// updating old vals

row\_old = row;

col\_old = col;

obs\_loc\_old = obs\_loc;

}

// gets filtered location

**void** **kalman\_filter**(**void**){

// State transmission model (calculating step size for each 10 ms)

**if** (heading == 0){ // 0 degrees, left to right

**if** ((filter\_loc == 3) || (filter\_loc == 4) || (filter\_loc == 11) || (filter\_loc == 12)) step\_size = 0; // right edge of world grid

**else** step\_size = 10.0/((**float**)ONE\_CELL\_TIME);

}

**else** **if** (heading == 2){ // 180 degees, right to left

**if** ((filter\_loc == 0) || (filter\_loc == 7) || (filter\_loc == 8) || (filter\_loc == 15)) step\_size = 0; // left edge of world grid

**else** step\_size = -10.0/((**float**)ONE\_CELL\_TIME);

}

**else** **if** (heading == 1){ // 90 degrees, bottom to top

**if** ((filter\_loc >= 0) && (filter\_loc <= 3))

step\_size = (7.0 - (2.0\*(**float**)filter\_loc))\*10.0/((**float**)ONE\_CELL\_TIME);

**else** **if** ((filter\_loc >= 4) && (filter\_loc <= 7))

step\_size = (15.0 - (2.0\*(**float**)filter\_loc))\*10.0/((**float**)ONE\_CELL\_TIME);

**else** **if** ((filter\_loc >= 8) && (filter\_loc <= 11))

step\_size = (23.0 - (2.0\*(**float**)filter\_loc))\*10.0/((**float**)ONE\_CELL\_TIME);

**else** step\_size = 0;

}

**else** **if** (heading == 3){ // 270 degrees, top to bottom

**if** ((filter\_loc >= 0) && (filter\_loc <= 3)) step\_size = 0;

**else** **if** ((filter\_loc >= 4) && (filter\_loc <= 7))

step\_size = (7.0 - (2.0\*(**float**)filter\_loc))\*10.0/((**float**)ONE\_CELL\_TIME);

**else** **if** ((filter\_loc >= 8) && (filter\_loc <= 11))

step\_size = (15.0 - (2.0\*(**float**)filter\_loc))\*10.0/((**float**)ONE\_CELL\_TIME);

**else**

step\_size = (23.0 - (2.0\*(**float**)filter\_loc))\*10.0/((**float**)ONE\_CELL\_TIME);

}

// Prediction

loc\_predicted = kalman\_loc\_old + step\_size;

// Updating

kalman\_loc = loc\_predicted + K\_kf\*((**float**)obs\_loc - loc\_predicted);

// Discrete Values

**if** ((kalman\_loc - (**int**)kalman\_loc) < 0.5) filter\_loc = (**int**)kalman\_loc;

**else** filter\_loc = (**int**)kalman\_loc + 1;

// Store old value

kalman\_loc\_old = kalman\_loc;

}

**void** **main**(**void**) {

WDTCTL = WDTPW + WDTHOLD; // Stop WDT

**if** (CALBC1\_16MHZ ==0xFF || CALDCO\_16MHZ == 0xFF) **while**(1);

DCOCTL = CALDCO\_16MHZ; // Set uC to run at approximately 16 Mhz

BCSCTL1 = CALBC1\_16MHZ;

// Initialize ADC10

// photoR at A7, front\_IR at A6, right\_IR at A5

ADC10CTL1 = INCH\_7 + ADC10SSEL\_3 + CONSEQ\_1; // Enable A7 first, Use SMCLK, Sequence of Channels

ADC10CTL0 = ADC10ON + MSC + ADC10IE; // Turn on ADC, Put in Multiple Sample and Conversion mode, Enable Interrupt

ADC10DTC1 = 8; // Eight conversions.

ADC10SA = (**short**)&sensors[0]; // ADC10 data transfer starting address. Hence, array is filled backwards (i.e A7 in ADC[0] to A0 in ADC[7])

// Initialize Port 2

P2SEL |= 0x14; // set P2.2 and P2.4 as

P2SEL2 &= ~0x14; // TA 1.1 and TA 1.2 for

P2DIR |= 0x14; // sending PWM to motors

// Timer A Config

TACCTL0 = CCIE; // Enable Periodic interrupt

TACCR0 = 16000; // period = 1ms

TACTL = TASSEL\_2 + MC\_1; // source SMCLK, up mode

// Timer A1 Config

TA1CTL = TASSEL\_2 + MC\_1; // SMCLK, up mode

TA1CCTL0 = 0; // corresponds to TA1CCR0

TA1CCTL1 = OUTMOD\_7; // Reset/set mode for TA1.1 PWM

TA1CCTL2 = OUTMOD\_7; // Reset/set mode for TA1.2 PWM

TA1CCR0 = 800; // carrier freq of 20 kHz

TA1CCR1 = PWM\_STOP; // initially robot at rest

TA1CCR2 = PWM\_STOP;

exploration(); // get exploration trajectory

first\_cmd = 1; // first cmd after prog starts

dir\_change = 1; // set as 1 to calculate right\_IR\_ref in beginning

**Init\_UART**(9600,1); // Initialize UART for 9600 baud serial communication

\_BIS\_SR(GIE); // Enable global interrupt

**while**(1) {

**if**(newmsg) {

newmsg = 0;

}

**if** (newprint) {

**UART\_printf**("%d?%d?%d?%d\n\r",ref\_loc,obs\_loc,filter\_loc,obstacle\_found);

newprint = 0;

}

}

}

// Timer A0 interrupt service routine

**#pragma** vector=TIMER0\_A0\_VECTOR

**\_\_interrupt** **void** **Timer\_A** (**void**)

{

timecheck++; // Keep track of time for main while loop.

**if** (timecheck % 10 == 0){ // every 10 ms

ADC10CTL0 |= ENC + ADC10SC; // Enable Sampling and start ADC conversion

}

**if** (timecheck == 500) {

timecheck = 0;

**if** (start\_timer) newprint = 1; // print only when car is moving

}

}

// ADC 10 ISR - Called when a sequence of conversions (A7-A0) have completed

**#pragma** vector=ADC10\_VECTOR

**\_\_interrupt** **void** **ADC10\_ISR**(**void**) {

**if** (start\_timer == 1){ // if right\_IR\_ref is calculated

timerV++; // time for each move or step (mutiples of 10 ms)

**switch** (move\_type){

**case** 'x': // pause after first time right\_IR\_ref is calculated

car\_stop = 1;

**if** ((timerV \* 10) >= PAUSE\_FORWARD){ // forward after pause time elapsed

car\_stop = 0;

move\_type = 's';

timerV = 0; // reset timer

first\_s\_cmd = 1; // flag its first forward move next

}

**break**;

**case** 's': // forward

**if** ((timerV \* 10) >= (cell\_cnt \* ONE\_CELL\_TIME)){ // move forward until time for cell\_cnt # of cells has elapsed

move\_type = which\_turn; // next move is left or right turn - set by explore\_traj

**if** (which\_turn == 'l') theta++; // tracking car angle

**else** **if** (which\_turn == 'r') theta--;

timerV = 0; // reset timer

}

**break**;

**case** 'l':

**if** ((timerV \* 10) >= TURNING\_TIME){

dir\_change = 1;

timerV = 0;

}

**break**;

**case** 'r':

**if** ((timerV \* 10) >= TURNING\_TIME){

dir\_change = 1;

timerV = 0;

}

**break**;

**case** 'e': // trajectory completed

car\_stop = 1;

move\_type = 'e';

**break**;

**default**:

car\_stop = 1;

}

}

// get current sensor values from adc

photoR[0] = sensors[0];

front\_IR[0] = sensors[1];

right\_IR[0] = sensors[2];

// Reversing IR values to get high value for larger distance

front\_IR[0] = 1023 - front\_IR[0];

right\_IR[0] = 1023 - right\_IR[0];

// Not averaged raw value for right\_IR\_ref calculations

right\_IR\_raw = right\_IR[0];

front\_IR\_raw = front\_IR[0];

// average filtering sensor data

**int** i = 19; // 20 old vals used in filtering

**for** (i = 19; i > 0; i--){

photoR[0] += photoR[i];

front\_IR[0] += front\_IR[i];

right\_IR[0] += right\_IR[i];

**if** (i > 1){ // updating old values

photoR[i] = photoR[i-1];

front\_IR[i] = front\_IR[i-1];

right\_IR[i] = right\_IR[i-1];

}

}

// taking average

photoR[0] = ((**float**)photoR[0])/20.0;

front\_IR[0] = ((**float**)front\_IR[0])/20.0;

right\_IR[0] = ((**float**)right\_IR[0])/20.0;

// updating immediate old values

photoR[1] = photoR[0];

front\_IR[1] = front\_IR[0];

right\_IR[1] = right\_IR[0];

// find reference right\_IR when dir is changed

**if** (dir\_change){

car\_stop = 1; // stop car when ref is calculated

start\_timer = 0; // stop 'move\_type' state-machine at top

**if** (rightIRref\_timer == 0) right\_IR\_ref = 0; // reset ref value

rightIRref\_timer++; // track time passed in ref calculation

right\_IR\_ref += (**float**)right\_IR\_raw;

**if** (rightIRref\_timer >= 200){ // 2000 ms passed

right\_IR\_ref = right\_IR\_ref/200.0; // take average

rightIRref\_timer = 0; // reset timer

move\_type = 's'; // drive forward after this

timerV = 0; // step timer reset to 0

start\_timer = 1; // start timerV, enter state\_machine

car\_stop = 0; // car can move now

dir\_change = 0; // reset flag

// getting traj information

**if** (traj\_step\_num < 7){ // 7 steps in explore\_traj

cell\_cnt = explore\_traj[traj\_step\_num][0]; // no. of cells to be traveled

**if** (explore\_traj[traj\_step\_num][1] == 0) which\_turn = 'l';

**else** which\_turn = 'r';

}

**else** { // traj completed

cell\_cnt = 0;

move\_type = 'e';

car\_stop = 1;

}

**if** (first\_cmd){ // if first command after prog begins

move\_type = 'x'; // pause before forward move

car\_stop = 1;

first\_cmd = 0; // reset flag

}

traj\_step\_num++; // next step of trajectory

}

}

// wall detection in front

**if** ((front\_IR\_raw < 400) && (start\_timer == 1)){

move\_type = which\_turn;

timerV = 0;

}

// move forward

**if** (move\_type == 's'){

// turn calculated as control input for right wall following

turn = (**int**)(K\_turn\*(right\_IR\_ref - right\_IR[0]));

// Motor switching ON causes weird IR vals

// so open loop for some time after motors start

**if** ((timerV\*10 < PAUSE\_CONTROL\_TIME) && (first\_s\_cmd)) {

turn = 0;

**if** (timerV\*10 >= PAUSE\_CONTROL\_TIME - 10) first\_s\_cmd = 0;

}

drive\_mode = 's'; // straight driving

}

**if** (move\_type == 'l'){

drive\_mode = 'l'; // left turn

}

**if** (move\_type == 'r'){

drive\_mode = 'r'; // right turn

}

// stop car: highest preference so last statement before sending command

**if** (car\_stop){

drive\_mode = 'x';

}

// send robot drive commands

drive\_car(drive\_mode); // modes as input: forward, 90 deg turn etc.

// car location according to velocity model (reference traj)

**if** (traj\_step\_num == 0) start\_cell = 0;

**else** start\_cell = explore\_traj[traj\_step\_num - 1][2];

**if** (move\_type == 'x'){

ref\_loc = 0; // enter x only when code starts

}

**else** **if** ((move\_type == 'l') || (move\_type == 'r')){

ref\_loc = start\_cell + cell\_cnt; // end of a traj step

}

**else** **if** (move\_type == 's'){

ref\_loc = start\_cell + ((timerV\*10)/ONE\_CELL\_TIME);

**if** (ref\_loc > (start\_cell + cell\_cnt)) ref\_loc = start\_cell + cell\_cnt;

}

**else** **if** (move\_type == 'e'){

ref\_loc = 15;

}

// localization

**if** (theta < 0) heading = (-theta) % 4;

**else** heading = theta % 4;

**if** ((move\_type == 'l') || (move\_type == 'r') || (move\_type == 'e') || (move\_type == 'x')){ // IR values unreliable during turns

obs\_loc = obs\_loc\_old;

kalman\_loc = kalman\_loc\_old;

}

**else**{ // when straight forward driving

observed\_loc(); // get observed location

kalman\_filter(); // get filtered location

}

// mapping

**if** (start\_timer){ // if moving

**if** (photoR[0] < PHOTOR\_THRESHOLD) obs\_count++; // count consecutive calls when photo resistor reads occupied cell

**else**{

obs\_count = 0;

obstacle\_found = 0;

}

**if** (obs\_count >= 20) obstacle\_found = 1; // flag when occupied cell read for consecutive 200 ms

}

// for next call of ADC ISR

ADC10CTL0 &= ~ADC10IFG; // clear interrupt flag

ADC10SA = (**short**)&sensors[0]; // ADC10 data transfer starting address

}

// USCI Transmit ISR - Called when TXBUF is empty (ready to accept another character)

**#pragma** vector=USCIAB0TX\_VECTOR

**\_\_interrupt** **void** **USCI0TX\_ISR**(**void**) {

**if**(IFG2&UCA0TXIFG) { // USCI\_A0 requested TX interrupt

**if**(printf\_flag) {

**if** (currentindex == txcount) {

senddone = 1;

printf\_flag = 0;

IFG2 &= ~UCA0TXIFG;

} **else** {

UCA0TXBUF = printbuff[currentindex];

currentindex++;

}

} **else** **if**(UART\_flag) {

**if**(!donesending) {

UCA0TXBUF = txbuff[txindex];

**if**(txbuff[txindex] == 255) {

donesending = 1;

txindex = 0;

}

**else** txindex++;

}

} **else** { // interrupt after sendchar call so just set senddone flag since only one char is sent

senddone = 1;

}

IFG2 &= ~UCA0TXIFG;

}

**if**(IFG2&UCB0TXIFG) { // USCI\_B0 requested TX interrupt (UCB0TXBUF is empty)

IFG2 &= ~UCB0TXIFG; // clear IFG

}

}

// USCI Receive ISR - Called when shift register has been transferred to RXBUF

// Indicates completion of TX/RX operation

**#pragma** vector=USCIAB0RX\_VECTOR

**\_\_interrupt** **void** **USCI0RX\_ISR**(**void**) {

**if**(IFG2&UCB0RXIFG) { // USCI\_B0 requested RX interrupt (UCB0RXBUF is full)

IFG2 &= ~UCB0RXIFG; // clear IFG

}

**if**(IFG2&UCA0RXIFG) { // USCI\_A0 requested RX interrupt (UCA0RXBUF is full)

// Uncomment this block of code if you would like to use this COM protocol that uses 253 as STARTCHAR and 255 as STOPCHAR

/\* if(!started) { // Haven't started a message yet

if(UCA0RXBUF == 253) {

started = 1;

newmsg = 0;

}

}

else { // In process of receiving a message

if((UCA0RXBUF != 255) && (msgindex < (MAX\_NUM\_FLOATS\*5))) {

rxbuff[msgindex] = UCA0RXBUF;

msgindex++;

} else { // Stop char received or too much data received

if(UCA0RXBUF == 255) { // Message completed

newmsg = 1;

rxbuff[msgindex] = 255; // "Null"-terminate the array

}

started = 0;

msgindex = 0;

}

}

\*/

IFG2 &= ~UCA0RXIFG;

}

}

**explore\_traj.c**

/\* Demonstration of Localization and Mapping

\* using Minimal and Inexpensive Components

\* (For SE 243: Mechatronics mini project)

\*

\* explore\_traj.c

\* Trajectory for exploration of world

\* Created on: 08-Apr-2018

\* Author: Ayush Sinha

\*

\* World

\* ---------------------

\* | 15 | 14 | 13 | 12 |

\* ---------------------

\* | 8 | 9 | 10 | 11 |

\* ---------------------

\* | 7 | 6 | 5 | 4 |

\* ---------------------

\* | 0 | 1 | 2 | 3 |

\* ---------------------

\*/

**int** explore\_traj[7][3] = {0}; // 7 straight line trajectories reqd to explore world

/\* row = # of trajectory

\* col 0 = # of cells to travel forward = cell\_cnt

\* col 1 = Left(0) or Right(1) turn

\* col 2 = start cell number \*/

**void** **exploration**(**void**){

explore\_traj[0][0] = 3;

explore\_traj[0][1] = 0;

explore\_traj[0][2] = 0;

explore\_traj[1][0] = 1;

explore\_traj[1][1] = 0;

explore\_traj[1][2] = 3;

explore\_traj[2][0] = 3;

explore\_traj[2][1] = 1;

explore\_traj[2][2] = 4;

explore\_traj[3][0] = 1;

explore\_traj[3][1] = 1;

explore\_traj[3][2] = 7;

explore\_traj[4][0] = 3;

explore\_traj[4][1] = 0;

explore\_traj[4][2] = 8;

explore\_traj[5][0] = 1;

explore\_traj[5][1] = 0;

explore\_traj[5][2] = 11;

explore\_traj[6][0] = 3;

explore\_traj[6][1] = 0;

explore\_traj[6][2] = 12;

}

**MATLAB Code**function [ ] = PlotCarLocation( )

% Demonstartion of Localization and Mapping

% using Minimal and Inexpensive Components

%

% Ayush Sinha

% ayush7.sinha@gmail.com

% Date: 15 April 2018

% Summary:

% Plots Robot car's location and creates a map

% as the car explores the world.

% Left map plots desired (or reference) car location and creates

% the map acoordingly, Center corresponds to observed location

% (through IR readings) and Right map uses location through Kalman Filter

%

% World

% ---------------------

% | 16 | 15 | 14 | 13 |

% ---------------------

% | 9 | 10 | 11 | 12 |

% ---------------------

% | 8 | 7 | 6 | 5 |

% ---------------------

% | 1 | 2 | 3 | 4 |

% ---------------------

%% Make Grids

[X,Y]=meshgrid(1:5);

figure; hold on;

set(gcf, 'Position', get(0, 'Screensize'));

subplot(1,3,1);

plot(X,Y,'k'); hold on

plot(Y,X,'k'); axis square; axis off

title('Trajectory through Velocity Model');

subplot(1,3,2);

plot(X,Y,'k'); hold on

plot(Y,X,'k'); axis square; axis off

title('Trajectory through IR Measurements');

subplot(1,3,3);

plot(X,Y,'k'); hold on

plot(Y,X,'k'); axis square; axis off

title('Trajectory through Kalman Filter');

pause (1e-9); % to see plot while code's running

%% Assign Cell numbers on Grid (World)

cell\_loc = zeros(16,2);

% row = cell number in MATLAB

% col 1 = x\_loc

% col 2 = y\_loc

for i = 1:4

cell\_loc(i,:) = [i + 0.5, 1.5];

end

for i = 5:8

cell\_loc(i,:) = [9.5 - i, 2.5];

end

for i = 9:12

cell\_loc(i,:) = [i - 7.5, 3.5];

end

for i = 13:16

cell\_loc(i,:) = [17.5 - i, 4.5];

end

%% Plotting

%start serial

s = serial('COM3', 'BaudRate', 9600);

fopen(s);

% initialise old\_loc to zero-th cell

old\_ref\_loc = 1;

old\_obs\_loc = 1;

old\_fil\_loc = 1;

subplot(1,3,1);

loc = plot(cell\_loc(old\_ref\_loc,1),cell\_loc(old\_ref\_loc,2),'-bs','MarkerSize',25,'MarkerFaceColor','b');

subplot(1,3,2);

oloc = plot(cell\_loc(old\_obs\_loc,1),cell\_loc(old\_obs\_loc,2),'-rs','MarkerSize',25,'MarkerFaceColor','r');

subplot(1,3,3);

floc = plot(cell\_loc(old\_fil\_loc,1),cell\_loc(old\_fil\_loc,2),'-gs','MarkerSize',25,'MarkerFaceColor','g');

pause (1e-9);

driving = 1; % flags when car is driving

% plot new locations

while driving

% get new location from serial

C=fscanf(s);

c\_str = regexp(C, '?', 'split');

new\_ref\_loc = str2double(c\_str(1));

new\_obs\_loc = str2double(c\_str(2));

new\_fil\_loc = str2double(c\_str(3));

obstacle\_found = round(str2double(c\_str(3)));

% MATLAB array indices start from 1, hence +1

new\_ref\_loc = round(new\_ref\_loc) + 1;

new\_obs\_loc = round(new\_obs\_loc) + 1;

new\_fil\_loc = round(new\_fil\_loc) + 1;

% plot robot reference traj

subplot(1,3,1);

if (obstacle\_found == 1) % blacken cell if obstacle found

plot(cell\_loc(new\_ref\_loc,1),cell\_loc(new\_ref\_loc,2),'-ks','MarkerSize',70,'MarkerFaceColor','k');

end

plot([cell\_loc(old\_ref\_loc,1) cell\_loc(new\_ref\_loc,1)],[cell\_loc(old\_ref\_loc,2) cell\_loc(new\_ref\_loc,2)],'-b','LineWidth', 7);

delete(loc)

loc = plot(cell\_loc(new\_ref\_loc,1),cell\_loc(new\_ref\_loc,2),'-bs','MarkerSize',25,'MarkerFaceColor','b');

% plot robot observed traj

subplot(1,3,2);

if (obstacle\_found == 1) % blacken cell if obstacle found

plot(cell\_loc(new\_obs\_loc,1),cell\_loc(new\_obs\_loc,2),'-ks','MarkerSize',70,'MarkerFaceColor','k');

end

plot([cell\_loc(old\_obs\_loc,1) cell\_loc(new\_obs\_loc,1)],[cell\_loc(old\_obs\_loc,2) cell\_loc(new\_obs\_loc,2)],'-r','LineWidth', 7);

delete(oloc)

oloc = plot(cell\_loc(new\_obs\_loc,1),cell\_loc(new\_obs\_loc,2),'-rs','MarkerSize',25,'MarkerFaceColor','r');

% plot robot kalman-fltered traj

subplot(1,3,3);

if (obstacle\_found == 1) % blacken cell if obstacle found

plot(cell\_loc(new\_fil\_loc,1),cell\_loc(new\_fil\_loc,2),'-ks','MarkerSize',70,'MarkerFaceColor','k');

end

plot([cell\_loc(old\_fil\_loc,1) cell\_loc(new\_fil\_loc,1)],[cell\_loc(old\_fil\_loc,2) cell\_loc(new\_fil\_loc,2)],'-g','LineWidth', 7);

delete(floc)

floc = plot(cell\_loc(old\_fil\_loc,1),cell\_loc(old\_fil\_loc,2),'-gs','MarkerSize',25,'MarkerFaceColor','g');

pause (1e-9); % to see figure while plotting

% update old\_loc

old\_ref\_loc = new\_ref\_loc;

old\_obs\_loc = new\_obs\_loc;

old\_fil\_loc = new\_fil\_loc;

% end plotting when traj finished

if new\_ref\_loc == 16

driving = 0;

end

end

fclose(s);

hold off

end