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
```