Jiaming

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

*MSP430G2553 Project Creator*

*SE 423 - Dan Block*

*Spring(2019)*

*Written(by) : Steve(Keres)*

*College of Engineering Control Systems Lab*

*University of Illinois at Urbana-Champaign*

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

#include "msp430g2553.h"

#include "UART.h"

**void** print\_every**(int** rate**);**

**char** newprint **=** 0**;**

**long** NumOn **=** 0**;**

**long** NumOff **=** 0**;**

**int** statevar **=** 1**;**

**int** timecheck **=** 0**;**

**unsigned** **int** previous **=** 0**;**

**unsigned** **int** current **=** 5**;**

**unsigned** **int** time1 **=** 0**;**

**unsigned** **int** time2 **=** 0**;**

**int** edge 0**;**

**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 Port 1*

P1SEL **&=** **~**0x01**;** *// Make sure P1.0 GPIO*

P1SEL **|=** 0x40**;** *// Set P1.6 as TA0.1*

P1SEL2 **&=** **~**0x41**;** *// P1.0 GPIO P1.6 TA0.1*

P1REN **=** 0x0**;** *// No resistors enabled for Port 1*

P1DIR **|=** 0x41**;** *// Set P1.0 output P1.6 TA0.1*

P1OUT **&=** **~**0x01**;** *// Initially set P1.0 to 0*

*// TODO: Initialize Port 2.1 as TA1.CCI1A capture input pin*

P2SEL **|=** 0x02**;** *// Set P2.1 TA1.CCI1A*

P2SEL2 **&=** **~**0x02**;** *// Set P2.1 TA1.CCI1A*

P2DIR **&=** **~**0x02**;** *// Set P2.1 TA1.CCI1A*

*// Timer0 A Config So this sets Timer0 to interrupt every 1ms*

*// and generate a PWM signal on P1.6*

TA0CCTL0 **=** CCIE**;** *// Enable Periodic interrupt*

TA0CCR0 **=** 16000**;** *// period = 1ms*

TA0CCR1 **=** 8000**;** *// Start with 50% duty cycle*

TA0CCTL1 **=** OUTMOD\_7**;** *// Set/Reset*

TA0CTL **=** ID\_1**+** TASSEL\_2 **+** MC\_1**;** *// divide by 1, source SMCLK, up mode*

*// TODO : Timer A1 Config Set Timer A1 in capture mode*

*// and use capture pin TA1.CCI1A at P2.1*

TA1CCTL1 **=** CM\_3 **+** CCIS\_0 **+** CAP **+** CCIE**;** *// TA1CCR1 Capture mode; CCI1A; Both*

*// Rising and Falling Edge; interrupt enable*

TA1CTL **=** TASSEL\_2 **+** MC\_2 **+** TACLR**;** *// SMCLK, Continous Mode; Clear timer*

Init\_UART**(**115200**,**1**);** *// Initialize UART for 115200 baud serial communication*

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

**while(**1**)** **{** *// Low priority Slow computation items go inside this while loop. Very few (if anyt) items in the HWs will go inside this while loop*

*// for use if you want to use a method of receiving a string of chars over the UART see USCI0RX\_ISR below*

*// if(newmsg) {*

*// newmsg = 0;*

*// }*

*// The newprint variable is set to 1 inside the function "print\_every(rate)" at the given rate*

PP **if** **(** **(**newprint **==** 1**)** **&&** **(**senddone **==** 1**)** **)** **{** *// senddone is set to 1 after UART transmission is complete*

*// only one UART\_printf can be called every 15ms*

UART\_printf**("TA0Perid %uus,TA0OnTime %uus\n\r",(int)(((**time1**+**time2**)\***1000L**)/**16000**),(int)((**time1**\***1000L**)/**16000**));**

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.*

print\_every**(**250**);** *// units determined by the rate Timer\_A ISR is called, print every "rate" calls to this function*

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

P1OUT **^=** 0x1**;**

**if** **(**TACCR1**<(**TACCR0**-**1000**)){**

TACCR1 **=** TACCR1**+**1000**;**

**}**

**else{**

TACCR1 **=** 4000**;**

**}**

timecheck **=** 0**;**

**}**

**}**

*// TA1\_A1 Interrupt vector*

#pragma vector = TIMER1\_A1\_VECTOR

\_\_interrupt **void** TIMER1\_A1\_ISR **(void)**

**{**

**switch(**TA1IV**)**

**{**

**case** TA1IV\_NONE**:**

*// Should not get here*

**break;**

**case** TA1IV\_TACCR1**:** *// TACCR1 CCIFG Capture interrupt*

*//TODO Read TACCR1 to know when the Capture interrupt occurred.*

*// and then additional code to perform given tasks.*

current **=** TA1CCR1**;**

**if** **(**previous **!=** 0**){**

edge **=** TA1CCTL1**;**

**if** **((**edge **&=** 0x08**)** **==** 0**){** *//distinguish rising or falling edge*

time1 **=** current **-** previous**;** *//calclualte the time period by the difference between two interrupt time*

**}**

**else{**

time2 **=** current **-** previous**;**

**}**

**}**

previous **=** current**;** *// replace previous by current, move to next step*

**break;**

**case** TA1IV\_TACCR2**:** **break;** *// TACCR2 CCIFG Not used in HW6*

**case** TA1IV\_6**:** **break;** *// Reserved CCIFG Not used*

**case** TA1IV\_8**:** **break;** *// Reserved CCIFG Not used*

**case** TA1IV\_TAIFG**:** *// TAIFG Overflow interrupt. Used in Challenge part*

**break;**

**default:** **break;**

**}**

**}**

*/\**

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

*#pragma vector=ADC10\_VECTOR*

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

*}*

*\*/*

*// 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**;**

**}**

**}**

*// This function takes care of all the timing for printing to UART*

*// Rate determined by how often the function is called in Timer ISR*

**int** print\_timecheck **=** 0**;**

**void** print\_every**(int** rate**)** **{**

**if** **(**rate **<** 15**)** **{**

rate **=** 15**;**

**}**

**if** **(**rate **>** 10000**)** **{**

rate **=** 10000**;**

**}**

print\_timecheck**++;**

**if** **(**print\_timecheck **==** rate**)** **{**

print\_timecheck **=** 0**;**

newprint **=** 1**;**

**}**

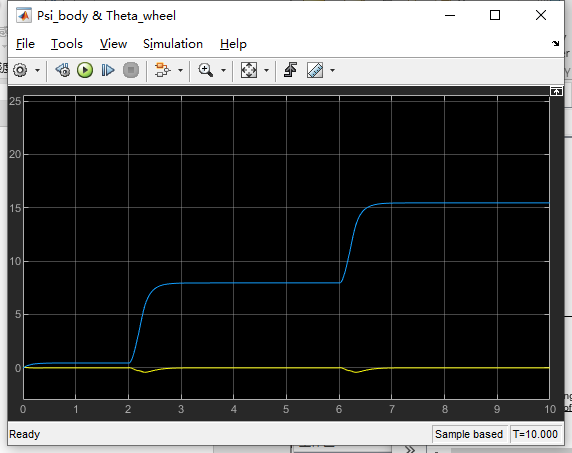
**}**

2.

(1)

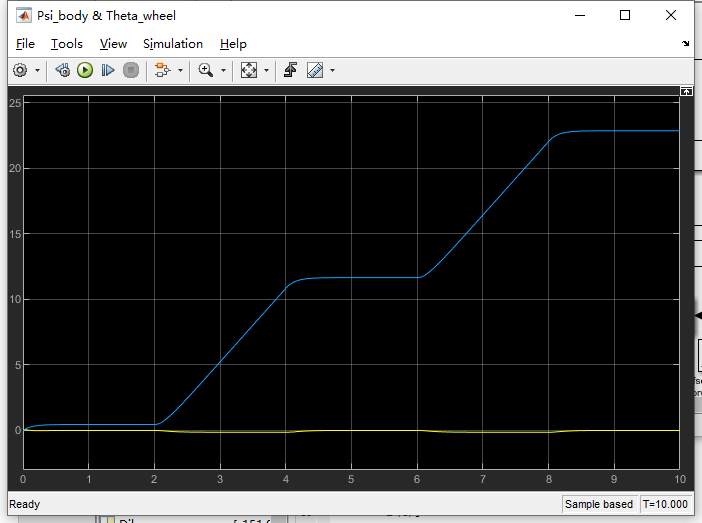
K3 = place(A3, B3, [-241.1937 -10 -6.5199]);

Amplitude=2.5, period=4s, pulse width=5%, phase delay=2s



(2)

Amplitude=0.4, period=4s, pulse width=50%, phase delay=2s



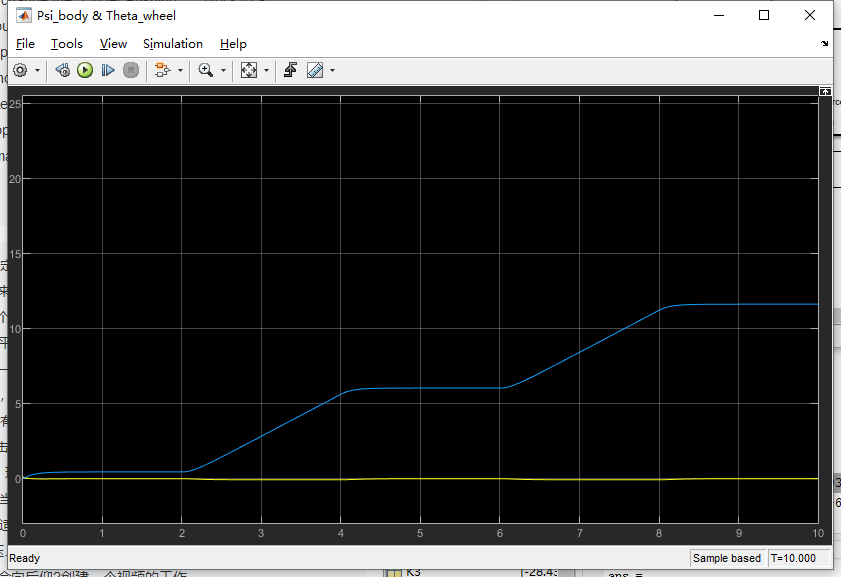
This time the Segway robot move step by step, in each step it move forward and stop.

The finger pushing force is placed at the top of the body, if there is no controller, the robot will fall forward. However, to keep the robot not fall down, the controller needs to drive the motor to drive the robot forward.

Whether the robot lean forward or backward can be determined by the gain of the controller. When moving the robot, the body needs to keep leaning back, because if it lean forward, when the extend force disappears, the wheel need to slow down the boy will fall forward to the ground because of the inertia. So to keep the robot more bearable, it needs to lean back.

(3)

Amplitude=0.2, period=4s, pulse width=50%, phase delay=2s



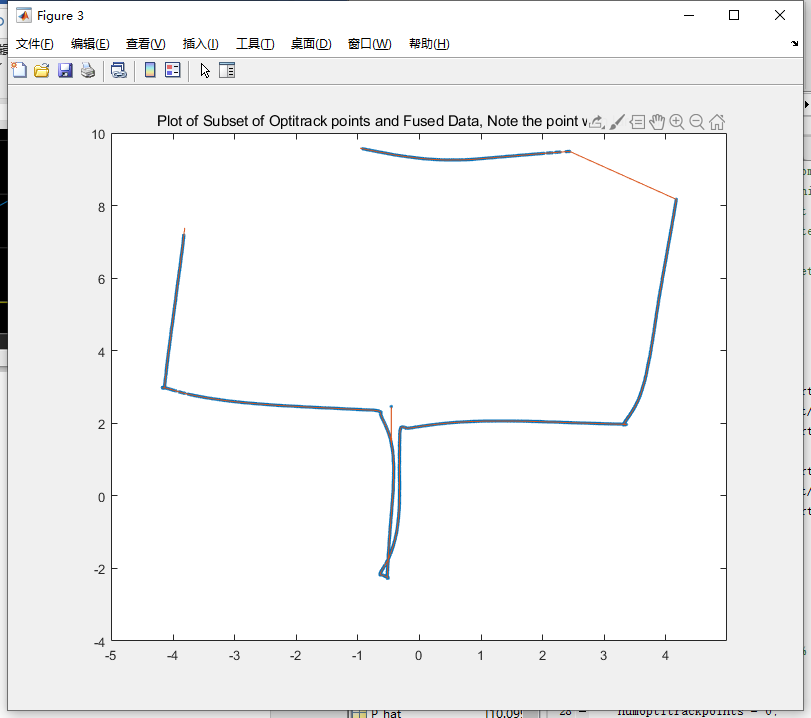
The robot can move forward and stop as we want.

3

1. Kalman output data follows the Optitrack data too closely

ProcUncert = 10;

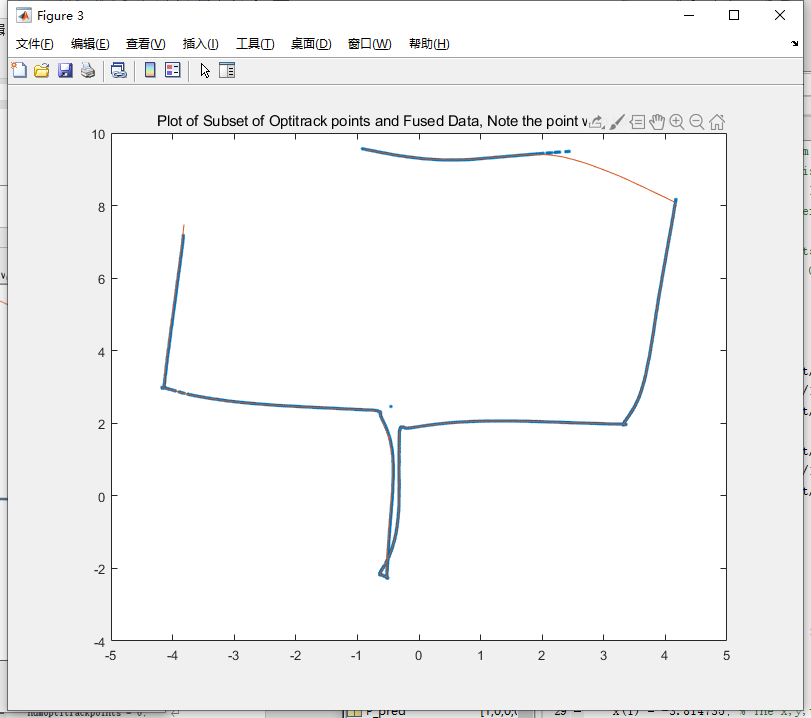
MeasUncert = 0.1;



1. Kalman output is filtered too much and converges slowly to the Optitrack data.

ProcUncert = 0.1;

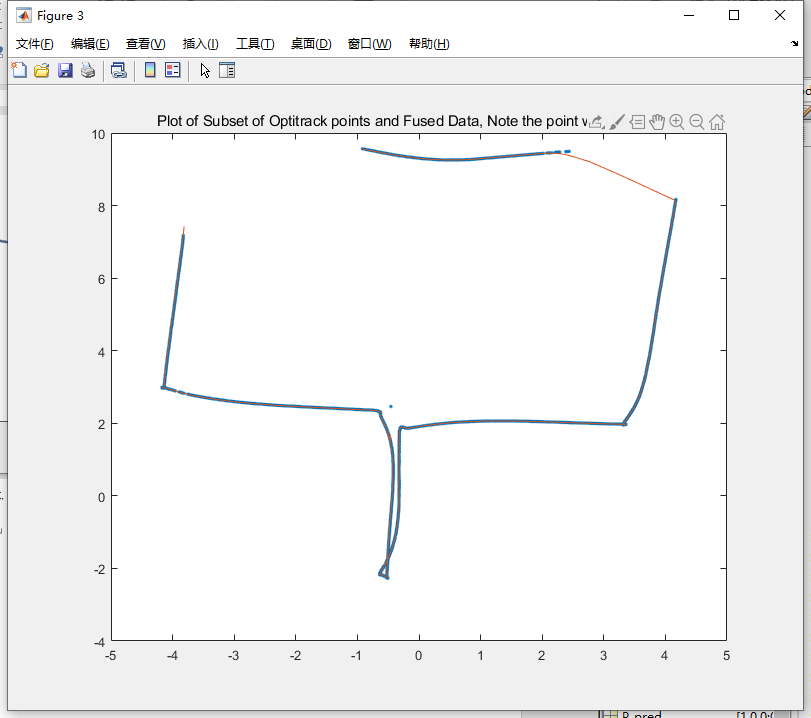
MeasUncert = 10;



1. Kalman output is just right, converges pretty quickly but does not jump imidiately to every Optitrack data point.

ProcUncert = 0.15;

MeasUncert = 3;



%You must have the data from "data\_forKalman\_Filter\_HWProblem.m" loaded in Matlab's

%memory before executing this M-file.

% type at the Matlab prompt >> data\_forKalman\_Filter\_HWPrblem

% then type edit KalmanFilterHWProblem.m if you have not done so already

x\_pred = [0;0;0]; %[x,y,theta]

z = [0;0;0]; %[Optitrack x, Optitrack y, Optitrack Theta

ytilde = [0;0;0];

K = eye(3);

ProcUncert = 0.15;

Q = [ProcUncert 0 ProcUncert/10;

0 ProcUncert ProcUncert/10;

ProcUncert/10 ProcUncert/10 ProcUncert];

MeasUncert = 3;

R = [MeasUncert 0 MeasUncert/10;

0 MeasUncert MeasUncert/10;

MeasUncert/10 MeasUncert/10 MeasUncert];

S = eye(3);

P\_pred = eye(3);

x = zeros(200000,1);

y = zeros(200000,1);

theta = zeros(200000,1);

opti\_x = zeros(17593,1); % in this data set there happens to be 17593 Optitrack data points

opti\_y = zeros(17593,1);

numoptitrackpoints = 0;

x(1) = -3.814735; % The x,y,theta location where I happened to start the robot for this run

y(1) = 6.565212;

theta(1) = -1.588956;

x\_pred(1,1) = x(1);

x\_pred(2,1) = y(1);

x\_pred(3,1) = theta(1);

F = eye(3);

H = eye(3);

P = eye(3);

U = data(:, 2:3);

Z = data(:, 4:6);

for i=1:199999 %Loop through all the data. Each data point is 0.001s.

%Prediction steps that happen every loop.

%Helping out here because B\*u is a bit confusing because is uses a previous step back theta

uk = [data(i,2);data(i,3)]; %[Average wheel velocity, gyro reading]

B = [cos(theta(i))\*0.001 0;sin(theta(i))\*0.001 0;0 0.001];

u = U(i,:)';

%Calcuate x\_pred (xhatK+1|K) = F\*(xhatK|K) + B\*uK

x\_hat = F\*x\_pred + B\*u;

y\_hat = H\*x\_hat;

%Calculate P\_pred (PK+1|K) = F\*(PK|K)\*F' + Q

P\_hat = F\*P\*F'+Q;

% Correction Steps

% In the data set I collected when there was no new Optitrack data I

% assigned the Optitrack data equal to 0.0. So if the data point is

% not equal to zero it is a new Optitrack point

if data(i,4) ~= 0.0

numoptitrackpoints = numoptitrackpoints + 1;

%log for plotting

opti\_x(numoptitrackpoints) = data(i,4);

opti\_y(numoptitrackpoints) = data(i,5);

%z is current Optitrack measurement

z(1,1) = data(i,4);

z(2,1) = data(i,5);

if cos(theta(i)) < -0.99 %Optitrack has difficulty with this angle

z(3,1) = theta(i);

else

z(3,1) = data(i,6);

end

% ytilde = zK+1 - H\*(xhatK+1|K)

% S = H\*(PK+1|K)\*H' + R

% K = (PK+1|K)\*H'\*S^-1

% (xhatK+1|K+1) = (xhatK+1|K) + K\*ytilde

% (PK+1|K+1) = (I - K\*H)\*(PK+1|K)

error = z - H\*(x\_pred);

S = H\*(P\_hat)\*H' + R;

K = (P\_hat)\*H'\*S^-1;

x\_pred = x\_hat+K\*error;

P = (eye(3) - K\*H)\*P\_hat;

end

% Log for plotting

x(i+1) = x\_pred(1,1);

y(i+1) = x\_pred(2,1);

theta(i+1) = x\_pred(3,1);

end

figure(1)

plot(opti\_x,opti\_y,'.')

title('Plot of all the Optitrack x,y points');

figure(2)

plot(x,y)

title('Plot of Fused Dead Reckoned Data and Optitrack Data');

figure(3)

plot(opti\_x(5200:8690),opti\_y(5200:8690),'.',x(60000:100000),y(60000:100000))

title('Plot of Subset of Optitrack points and Fused Data, Note the point with large error');