SE 423 Mechatronics Homework Assignment #6
Spring 2020, Due 5:00PM May 8th.

1. For this homework question, you are going to use the Capture functionality of the TimerA1 peripheral. Remember that the MSP430G2553 has two timers, TimerA0 and TimerA1. TimerA0 has been used in all of our previous homework assignments as our 1ms timer. We set TA0CCR0 = 16000 and that caused an interrupt every 1ms. For this homework problem, you will be given starter code that sets up TimerA0 to generate a periodic interrupt but also produce a PWM signal on pin P1.6. We will start with the period (also called the carrier frequency) of this PWM signal at 1ms, but in later steps in this assignment you will change the period to 2ms, then, 4ms and finally 8ms. The given code also changes the duty cycle of TimerA0's compare 1's output at a certain periodic rate and displays the current percent duty cycle to TeraTerm. As a first step cut and paste all of the starter code over the top of your new project's code. Compile and Run. You should see the percent duty cycle displayed to TeraTerm. In addition, if on your red board you install the jumper for the LED connected to P1.6, you should see the red LED (also on the red board) gradually get brighter as the percent duty cycle increases.
   Now that you having your PWM signal outputting on P1.6 you will program TimerA1 to use its capture functionality to measure both the period of the PWM signal and the length of time the signal is high which of course is related to the percent duty cycle. So using a paper clip as in HW5, jumper on your green board pin P1.6 to pin P2.1. Looking at the Digital I/O cheat-sheet you can find that P2.1, when TimerA1 is set in capture mode, connects to TimerA1's CCI1A line.
   To teach yourself the capture functionality, divide this task of measuring the PWM signal into two steps.
   Step one, set the capture so that the capture event only happens on the rising edge of the PWM signal. So in this step you will only be measuring the period of the PWM signal. Notice the given function __interrupt void TIMER1_A1_ISR (void). You will put your code inside the "case TA1IV_TACCR1:" case. This is the interrupt that is generated when the signal coming into the CCI1A input (P2.1) transitions from low to high (or high to low if you select falling edge). In this interrupt read the value of TA1CCR1 and store it to an "unsigned int" global variable each time so you can calculate how much time has gone by between interrupts. You will have to subtract the previous value in TA1CCR1 from the current value. (Note: you may wonder what happens when the 16 bit timer register, TAR, reaches its maximum value of 65535. It will rollover back down to 0 and continue counting up. We are going to talk about this in lecture, but if you define your global variable that stores the previous TA1CCR1 as an "unsigned int" then the subtraction to find the periodic rate of the signal works even if there is a roll over.) Then every 250ms print this period in units of microseconds to TeraTerm. You should find that the value is 1000us.
   Step two, set the capture so that the capture event happens on both the rising and falling edge of the PWM signal. This way you can write code that will allow you to both measure in microseconds the period of the PWM signal along with the time the PWM signal is high. Print both the period of the PWM signal and the length of time the signal is high to TeraTerm. To write this code you will need to know in your interrupt function whether the interrupt was caused by a rising edge or a falling edge. The trick to knowing if the interrupt was caused by a rising edge or a falling edge is to read, in the interrupt function, the current state (High or Low) of the PWM signal. The state of the PWM signal can be read with the CCI bit in the TA1CCTL1 register.
   As a final step try a few different periods. Change the period of your TimerA0 PWM signal by changing its clock divider. So change in main() the TACTL line and change ID_0 to ID_1. Now you should see your period being measure as 2000us. Change ID_1 to ID_2. Now your period should measure 4000us. Change ID_2 to ID_3. Now the period is 8000us but what does your code measure. What happened, why is it wrong?

Challenge, this will not be graded. Looking at TimerA1's interrupt function, there is a "case TA1IV_TAIFG:". This interrupt is called when Timer1's TAR register counts all the way to it maximum 16 bit value of 65535 (0xFFFF) and rolls over back to zero. Using this additional interrupt, modify your code so that the 8ms period and the length the PWM signal is high at that 8ms period can be measure correctly?

Question. What is the speed in RPM (revolutions per minute) of a wheel measured by a digital tachometer? The tachometer is made of a proximity sensor and a gear with five teeth. As a tooth of the gear comes close to the proximity sensor, the proximity sensor produces a high (3.3V) pulse. When the tooth has moved past the proximity sensor, the sensor's signal goes back low. So with five teeth on the gear, the tachometer generates 5 pulses per rotation. The tachometer gear is attached to the wheel so that one rotation of the wheel is one full rotation of the gear. If the time between tachometer pulses is 3592us, what is the speed of the wheel in RPM?

2. For this homework problem, I would like to thank Yorihisa Yamamoto who is (was?) an employee at Mathworks that put together documentation and Simulink files for controlling a self-balancing robot made out of Legos. I will be referencing his paper that describes the equations of motion of a Segbot. All his work can be found at the link https://www.mathworks.com/matlabcentral/fileexchange/19147-nxtway-gs-self-balancing-two-wheeled-robot-controller-design. If you download his files and find the docs folder you will find his paper NXTway-GS Model-Based Design.pdf. Our goal with this homework problem is to design a balancing control law to stabilize the Lego Segbot (NXTway) described by Yorihisa. Then with this design completed, for your final project you should be able to make the small changes to the parameters of the NXTway and form the equations for your Segbot design. With these equations, you will be able to design your balancing controller. A few of the motor parameters may be a challenge to find documented, so we will have to make some educated approximations for those parameters.

   The controller I would like you to design for this homework problem is just the balancing control law. We will talk about how to control the turn of the Segbot in lecture. For the balancing control, we only need to think of the Segbot looking at its side so motor θ_right equals motor θ_left and the voltage applied to both motors will be the same value v. Looking at the below figures we are only interested in the left figure that is looking at the side view of the Segbot which means ϕ is always going to be zero. So taking equations 3.13, 3.14 from the NXTway paper (Equation 3.15 is equal to zero because ϕ is zero) and setting ϕ to zero the equations of motion for the Segbot become: (See the NXTway paper for more details)

$$\psi = body\ rotation\ angle, \theta = motor\ rotation\ angle, v = voltage\ applied\ to\ one\ motor$$

$$\alpha = \frac{nK_t}{R_m}, \qquad \beta = \frac{nK_tK_b}{R_m} + f_m$$

$$\begin{bmatrix} ML^2 + J_\psi + 2n^2J_m & MRLcos(\psi) - 2n^2J_m \\ MRLcos(\psi) - 2n^2J_m & (2m+M)R^2 + 2J_w + 2n^2J_m \end{bmatrix}\begin{bmatrix}\ddot{\psi}\\ \ddot{\theta}\end{bmatrix} = \begin{bmatrix} MgLsin(\psi) + 2\beta\dot{\theta} - 2\beta\dot{\psi} - 2\alpha v \\ MLR\dot{\psi}^2\sin(\psi) - 2\beta\dot{\theta} + 2\beta\dot{\psi} + 2\alpha v \end{bmatrix}$$

$$linearize\ with\ 2nd\ order\ Taylor\ series, \cos = 1, \sin = angle\ , and\ squared\ states = 0$$

$$\begin{bmatrix} ML^2 + J_\psi + 2n^2J_m & MRL - 2n^2J_m \\ MRL - 2n^2J_m & (2m+M)R^2 + 2J_w + 2n^2J_m \end{bmatrix}\begin{bmatrix}\ddot{\psi}\\ \ddot{\theta}\end{bmatrix} = \begin{bmatrix} MgL\psi + 2\beta\dot{\theta} - 2\beta\dot{\psi} - 2\alpha v \\ -2\beta\dot{\theta} + 2\beta\dot{\psi} + 2\alpha v \end{bmatrix}$$

$$assign\ D = \begin{bmatrix} ML^2 + J_\psi + 2n^2J_m & MRL - 2n^2J_m \\ MRL - 2n^2J_m & (2m+M)R^2 + 2J_w + 2n^2J_m \end{bmatrix}, which\ is\ a\ constant\ and\ invertable$$

$$then\ this\ equation\ can\ be\ rewritten$$

$$\begin{bmatrix}\ddot{\psi}\\ \ddot{\theta}\end{bmatrix} = D^{-1}\begin{bmatrix} MgL & -2\beta & 0 & 2\beta \\ 0 & 2\beta & 0 & -2\beta \end{bmatrix}\begin{bmatrix}\psi\\ \dot{\psi}\\ \theta\\ \dot{\theta}\end{bmatrix} + D^{-1}\begin{bmatrix}-2\alpha\\ 2\alpha\end{bmatrix}v$$

$$since\ D\ is\ a\ constant\ symmetric\ matrix, D^{-1}\ is\ also\ a\ constant\ symmetric\ matrix$$

$$define\ D^{-1} = \begin{bmatrix} DI_{11} & DI_{12} \\ DI_{12} & DI_{22} \end{bmatrix}$$

$$set\ the\ states\ of\ our\ linearized\ system\ to \begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\end{bmatrix} = \begin{bmatrix}\psi\\ \dot{\psi}\\ \theta\\ \dot{\theta}\end{bmatrix}$$

$$then\ our\ four\ state\ space\ system\ is$$

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ DI_{11}(MgL) & DI_{11}(-2\beta) + DI_{12}(2\beta) & 0 & DI_{11}(2\beta) + DI_{12}(-2\beta) \\ 0 & 0 & 0 & 1 \\ DI_{12}(MgL) & DI_{12}(-2\beta) + DI_{22}(2\beta) & 0 & DI_{12}(2\beta) + DI_{22}(-2\beta) \end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\end{bmatrix} + \begin{bmatrix} 0 \\ DI_{11}(-2\alpha) + DI_{12}(2\alpha) \\ 0 \\ DI_{12}(-2\alpha) + DI_{22}(2\alpha) \end{bmatrix}v$$

3.1 Two-Wheeled Inverted Pendulum Model

NXTway-GS can be considered as a two wheeled inverted pendulum model shown in Figure 3-1.

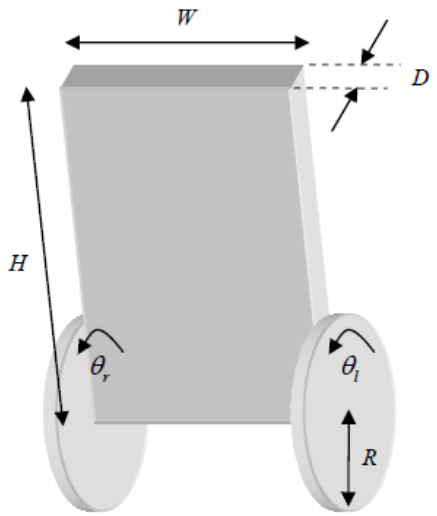
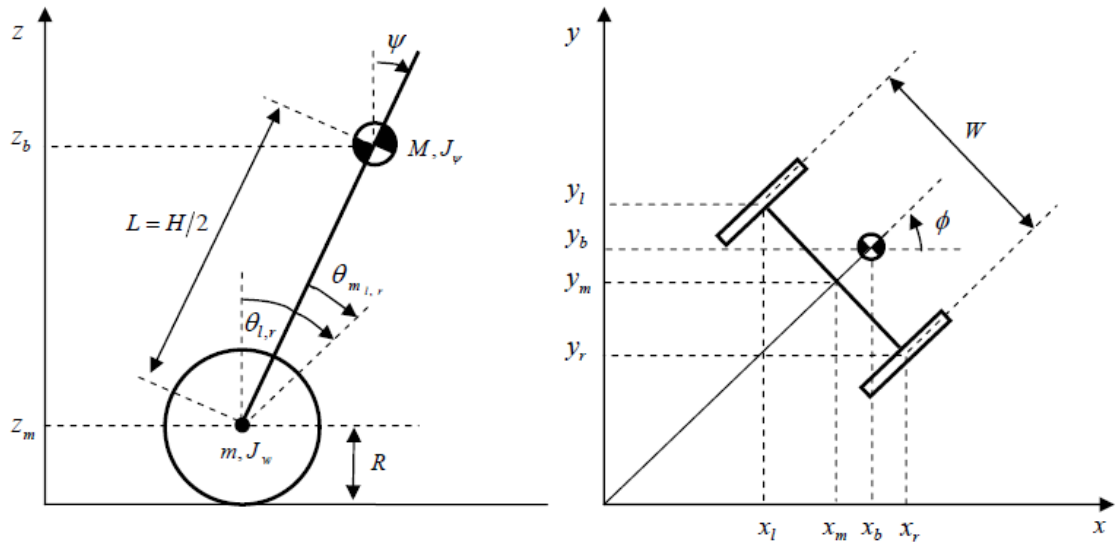


Figure 3-1 Two-wheeled inverted pendulum

Figure 3-2 shows side view and plane view of the two wheeled inverted pendulum. The coordinate system used in 3.2 Motion Equations of Two-Wheeled Inverted Pendulum is described in Figure 3-2.



ψ : body pitch angle $\theta_{l,r}$: wheel angle (l, r indicates left and right) $\theta_{m_{l,r}}$: DC motor angle

Figure 3-2 Side view and plane view of two-wheeled inverted pendulum

These Figures came from the NXTway paper from Mathworks.

If you have not yet, download this homework's segbotsimfiles.zip file. Unzip it to a folder on your PC. Then launch Matlab and change Matlab's working directory to the directory where you unzip these files. Run the M-file "equations.m" This file defines all the parameters of the NTXway and finds the A and B matrices of the linear approximation of the Segbot when it is close to its upright balancing position. You should find that it calculates the linear system to be

$$\dot{x} = Ax + Bv$$

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 269.6273 & -78.1496 & 0 & 78.1496\\ 0 & 0 & 0 & 1\\ -409.7184 & 162.1273 & 0 & -162.1273\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\end{bmatrix} + \begin{bmatrix}0\\ -151.9152\\ 0\\ 315.1597\end{bmatrix}v$$

You can play with this system if you would like, and I give a Simulink file that uses this four state system "segbot.slx." But this is not the system you are required to use for this homework. Why? Because one of the states, $x_3 = \theta$, is the angle of the motor. So if you design a controller, $v = -kx$, you are designing a controller that wants to drive all its states to zero so that means the controller will always want to try to keep the motor angle at 0 radians. That is ok when you are just balancing, but what if you want the Segbot to move forward and backward. Also, it is not reasonable to keep motor theta to zero on a real Segbot due to wheel slippage. It causes a number of problems when trying to control an actual Segbot. So instead, we are going to use a three state system to define and design controllers for the Segbot. So our final linear system that approximates the Segbot when it is close to it upright balancing position is:

$$\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} = \begin{bmatrix}\psi\\ \dot{\psi}\\ \dot{\theta}\end{bmatrix} and$$

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{bmatrix} = \begin{bmatrix}0 & 1 & 0\\ 269.6273 & -78.1496 & 78.1496\\ -409.7184 & 162.1273 & -162.1273\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} + \begin{bmatrix}0\\ -151.9152\\ 315.1597\end{bmatrix}v$$

The file "equations.m" create these matrices for you and calls them A3 and B3. Now given the system, your job is the use the "place" function in Matlab to design your controller v=-K*x. You need to figure out what pole locations will give you controller gains that control the Segbot sufficiently. You will run the command in Matlab, K3 = place(A3,B3,mypoles). A3 and B3 are given so you just need to find three "good enough" stable poles that allow the Segbot to balance. For this homework our definition of "good enough" is that your controller must be able to keep the Segbot balanced even when an external tap force of 2.5 newtons is applied and the corresponding response has minimal oscillations. An external tap force is applied by setting the amplitude of the block labeled "Finger Pushing At Top of Body". Use the defaults of Period 4secs, Pulse Width 5% and Phase delay 2 secs. In the equations.m file, I start you off with a set of gains, K3, that balance the Segbot but are too oscillatory and can also not handle the 2.5N tap. The default tap is only 0.5N. Make sure everything is working on your PC by loading the Simulink file segbot3state.slx and run the equations.m file if you have not already. Run the Simulink simulation and notice that the response is oscillatory but it does keep the Segbot balanced even with the 0.5N taps. Change the tap to 2.5N is see that the Segbot does not stay balanced when the taps are applied at 2 secs and then at 6 seconds. Now choose different poles and find other K3 gains to see if you can improve the performance. A good place to start is looking at the open-loop poles of the system (eig(A3)). Of course, you will need to make all the poles stable, but if a pole is already stable you may not have to move it. Make sure to indicate in your report what pole locations worked for you. Also submit plots of your system response and make a video of your animation working.

Now that you have a good controller, try a few things. First off, change the "Finger Pushing …" block to a 50% pulse width and change the amplitude to 0.4. This is simulating you pushing the Segbot for 2 seconds. Does the response of the Segbot make sense? Why does it lean back when you are pushing it? Create a video of this working.

Second, zero the amplitude of the "Finger Pushing …" block and change the amplitude of the "Psi Offset …" block to 0.2 radians. This is changing the operating point of the linearized system from 0 for Psi to 0.2. This is a method for telling the Segbot to go forward and backward. The offset is applied for 2 secs and then set back to zero. Does the Segbot stop as you would expect? Produce a video of this animation working.

3. The goal of this problem is to give you experience with the Kalman filter and understanding how it is helping with the robot's navigation. Given a set of actual data stored during a run of the robot car, your goal is to find a Q and a R matrix that makes the Kalman filter merge the dead reckoning data from the wheel encoders and rate gyro with the optitrack data. The "ideal" merge of this data is not to track exactly the optitrack data. It should definitely be close to the optitrack data but filter the optitrack data so large errors are not followed exactly. The optitrack's resolution is +\- 2cm and it is possible (but not very often) for the optitrack to send incorrect measurements. I have purposely added a 1 tile error in one of the optitrack data points so you can see how your Kalman filter reacts to that kind of large error.

   What are you given:

   A. The data file data_forKalman_Filter_HWProblem.mat found at http://coecsl.ece.illinois.edu/ge423/hw/HW6.htm. It has 6 columns: index, average wheel velocity, gyro reading in rad/s, x optitrack position, y optitrack position, theta optitrack. After you load the file you will see a lot of zeros in the optitrack columns. The optitrack only samples at 100 Hz where the dead reckoning data (wheel velocity, gyro) is updated at 1000 Hz. Your code will not use the optitrack data when it is 0.0. Load the "data" variable by loading the data_forKalman_Filter_HWProblem.mat file. This can be done using the "load" command or double clicking the file in the "current directory" window of the Matlab window.

   B. A Matlab script file that gives you most of what you need for the code for this problem, KalmanFilterHWProblem.m located http://coecsl.ece.illinois.edu/ge423/hw/HW6.htm. You will need to make two modifications to this file: (1) adding the Kalman equations to the large for loop, and (2) assigning values to the Q and R matrices used in the filter.

      In the m-file's for loop, there are comments to guide you through adding the code. Within the for loop you need to implement the prediction and correction steps. Referring to the given equations at the end of this homework might be useful during this coding.

      After adding the Kalman filter code to the for loop, you need to choose values for the Q and R matrices. The following form works well for the robot car.

$$U_P = Process\ (Dead\ Reckoning)\ Uncertainity, Q = \begin{bmatrix} U_p & 0 & \frac{U_p}{10} \\ 0 & U_p & \frac{U_p}{10} \\ \frac{U_p}{10} & \frac{U_p}{10} & U_p \end{bmatrix}$$

$$U_M = Measurement\ (Optitrack)\ Uncertainity, R = \begin{bmatrix} U_M & 0 & \frac{U_M}{10} \\ 0 & U_p & \frac{U_M}{10} \\ \frac{U_M}{10} & \frac{U_M}{10} & U_M \end{bmatrix}$$

      Start out with U_P and U_M at 1.0 and run the M-file. (U_P and U_m are named "MeasUncert" and "ProcUncert" in the code.) Zoom in on the third plot that plots both data sets on top of each other. Does it look like the Kalman filtered data is following the Optitrack data too closely and jumping around? Optitrack data is the dots. Also make sure to see what happens at the one error point that is off by one tile in the y direction. Then tune U_P and U_M values by changing them by a positive or negative power of 10 relative to each other. Come up with 3 Q and R matrices pairs that show three cases:

1. Kalman output data follows the Optitrack data too closely.
2. Kalman output is filtered too much and converges slowly to the Optitrack data.
3. Kalman output is just right, converges pretty quickly but does not jump imidiately to every Optitrack data point.

Make sure to produce a plot for each of these cases showing how the Kalman filter is working. In your plots, be sure to zoom in on the behavior of your fused data around the discontinuity in the optitrack data. Be sure to show how quickly the fused data converges to the optitrack data.

What to turn in:

1. A print out of your edited M-file
2. Three plots for the three Q and R pairs that you chose and somewhere by the plot the value of the Q and R matrices.

Kalman Equations

For the robot car

$$F = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} \cos(\theta_k)\Delta t & 0 \\ \sin(\theta_k)\Delta t & 0 \\ 0 & \Delta t \end{bmatrix}, \text{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \text{and start P as identity P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

K is 3 x 3 and S is 3 x 3

$$\begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \hat{x}_3 \end{bmatrix} = \begin{bmatrix} x\ position\ of\ robot \\ y\ position\ of\ robot \\ \theta\ of\ robot \end{bmatrix}, \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} average\ velocity\ of\ wheels \\ rate\ gyro\ measurement \end{bmatrix}, \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} optitrack\ x\ position\ of\ robot \\ optitrack\ y\ position\ of\ robot \\ optitrack\ \theta\ of\ robot \end{bmatrix}$$

Q and R are 3 x 3 matrixes that you need to tune.

Prediction Steps

$$\hat{x}_{k+1|k} = F\hat{x}_{k|k} + Bu_k$$

$$P_{k+1|k} = FP_{k|k}F^T + Q_k$$

Correction Steps

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K\tilde{y} \qquad \text{where} \quad \tilde{y} = z_{k+1} - H\hat{x}_{k+1|k}$$

$$P_{k+1|k+1} = (I - KH)P_{k+1|k} \qquad S = HP_{k+1|k}H^T + R_{k+1}$$

$$K = P_{k+1|k}H^T(S)^{-1}$$