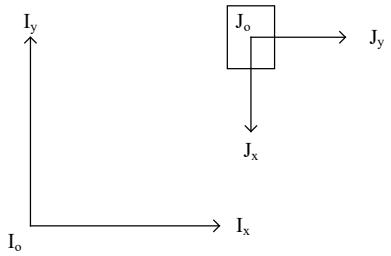
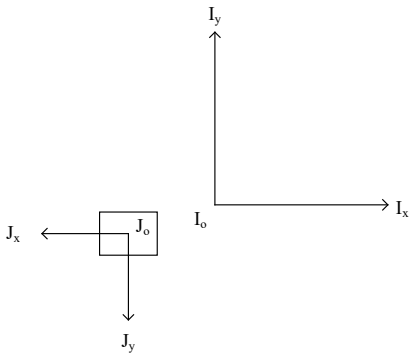
SE 423 Mechatronics Homework Assignment #4
Spring 2020, Due In Lecture April 1st. The Microcontroller Demonstration Check-Off for Questions 5 and 6 is due by 5PM Tuesday March 31st.
Most answers should be typed. Graphs, etc. can be hand drawn if you wish.

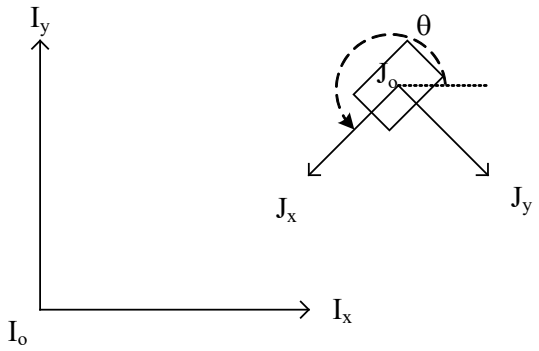
1. Read Chapter 10 in "Teach Yourself C". Read Wikipedia and other websites explaining "HSV Color Space". Read Chapter 9 in "Essentials of Mechatronics".
2. In Lab 8 we are going to be working with a color camera. Each pixel of the camera picture is described by a red, green and blue intensity value. This is called the RGB color space. Each component is stored in an 8-bit char variable with 0 representing no contribution and 255 full contribution. For the robot to identify other colors besides pure red, green, and blue it will need to look at a combination of these colors. An elegant way to do this is to convert the picture from the RGB color space to the HSV color space (Hue, Saturation, Value). In this color space, all the colors are arranged on a color wheel with a different angle indicating a different color. Do a web search for HSV color space to find more information about this color space and the equations to convert RGB to HSV. Write a function with the prototype `void RGBtoHSV(unsigned char r, unsigned char g, unsigned char b, unsigned char *h, unsigned char *s, unsigned char *v);`. This function should take the values passed in r, g and b, and convert them to the corresponding h, s and v values. h, s, and v are passed by reference so that their values can be changed by the function. Scale h so that its value from 0-360 is scaled to 0-255 (unsigned char). Scale the s and v values that range from 0 to 1 to a 0 to 255 value. Inside the function you will need to use floats to perform the conversion but return unsigned chars in the three variables passed by reference. *This code does NOT have to run on the MSP430G2553.*
3. Obtain the rotation matrix, which converts base J's coordinates into base I's coordinates (R_J^I), for the following cases:

a.



b.





4. Given the following translation vectors:
   a. $\overrightarrow{I_0J_0}$= (3.83,-1.75,0)$_I$
   b. $\overrightarrow{I_0J_0}$= (11.1,6.7,0)$_I$
   c. $\overrightarrow{I_0J_0}$ = (5.3,-1.4,0)$_I$

   and using the results on the previous a, b, and c questions respectively, obtain the homogeneous transformation matrix, which coverts frame J coordinates to frame I coordinates.

   Using the homogeneous transformation matrix found in part c. above, solve the following two questions:
   i. If a golf ball's coordinates are (2.8,-2.3,0) in the robot's (J) frame, what are the golf ball's coordinates in the world (I) frame as a function of θ? What are the golf ball's coordinates if θ equals 35°? To check that you are doing this problem correctly you should find that x is equal to 8.9129. Show your work and find this x value and also the y coordinate.
   ii. If a golf ball's coordinates are (2.54,1.45,0) in the world (I) frame, what are the golf ball's coordinates in the robot (J) frame as a function of θ? Note that the inverse of a homogeneous transformation matrix $H = \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix}$ is $H^{-1} = \begin{bmatrix} R^T & -R^T d \\ 0 & 1 \end{bmatrix}$. If θ equals 120°, is the golf ball on the right or left side of the robot? Remember that robot positive x points straight ahead of the robot and therefore positive y points to the robot's left. To check that you are doing this problem correctly you should find that x is equal to 3.8482. Show your work and find this x value and also the y coordinate and determine if the ball is on the right or left.

5. For this exercise you are going to take advantage of the hardware interrupt capability of the MSP430G2553's Digital I/O lines. You will use the same push buttons you have already soldered to pins P2.6 and P2.7. Now when you press (or tap) these switches individual hardware interrupts will be generated in the G2553. Read again chapter 8 of the MSP430 users guide: http://coecsl.ece.uiuc.edu/ge423/datasheets/MSP430Ref_Guides/MSP430x2xx_usersguide.pdf.
   http://coecsl.ece.illinois.edu/ge423/datasheets/MSP430Ref_Guides/Cexamples/MSP430G2xx3%20Code%20Examples/C/msp430g2xx3_P1_04_SE423.c is also a good example to look at to get started on this exercise. In your source code, just as in HW #1, setup P2.6 and P2.7 as inputs, enable the resistor and make it a pull-up resistor. This will cause the state of the switch to be logic high when it is not pressed and logic low when pressed. Starting with a new project created in project creator and using the example given above as a guide, modify the source code to enable P2.6 and P2.7 as hardware interrupts. When a hardware interrupt is generated on line P2.6, increment an integer (say int1) by one. When an interrupt is generated on P2.7 increment a second integer (say int2). Modify what prints to Tera Term to be these two interrupt count values. *Note: P2.6 and P2.7 both cause the same interrupt "PORT2_VECTOR" to be called. Inside the interrupt function you will have to decide which interrupt triggered the function call by checking the respective bit in the P2IFG register. Also don't forget to clear the P2IFG bit before exiting the function.* Compile and run your program. What happens? You should probably see every once in a while that when you press a switch you receive 2, 3 or more interrupts with each press. This is because the processor is seeing multiple jumps from high to low at the point of switch contact. This is termed "bouncing". We need to "debounce" this input line. There are a number of ways to accomplish this and we will talk about a number

of these in lecture. Here we will solve the problem by adding a delay. When an interrupt from a pin occurs, change your ISR code to still increment the integer and clear the P2IFG flag but now also disable that interrupt source (P2IE register). Set a flag variable indicating that P2.6 or P2.7 has been disabled. Then inside the Timer_A (void) interrupt function, wait for 10ms to go by and then re-enable the P2.6 or P2.7's interrupt. Now run your code and see if the extra interrupts have disappeared. If not increase the 10ms delay until you find good results. A good way to test if you have successfully "debounced" your switch interrupt is to tap the switch closed 10 times in a row and see if you received more than ten interrupts. Hand in your code and demonstrate it working to your TA. This exercise does not benefit much from using a state machine so you are not required to use one here.

6. For this exercise you will learn how to wire/use both transistors and relays to allow the G2553's digital outputs to turn on and off high current devices that the digital outputs alone would not be able to drive and if attempted to drive these loads would damage the digital outputs. The source code for this exercise is very simple. All you need to do is toggle on and off P2.0 and P2.1 outputs inside P2.6 and P2.7's interrupt service routines. So when P2.6 switch is pressed, P2.0 should toggle from "off to on" or "on to off". When P2.7 switch is pressed, P2.1 should toggle. Toggle the I/O pin by keeping track of the "On" or "Off" state with a global state variable for both interrupt sources. The rest of this question explains and illustrates how to wire P2.0 and P2.1 outputs to the given NPN transistors and then in turn use the high current output of the transistors to drive an ultra-bright LED and a relay.
If you are unfamiliar with transistors, refer to any introductory electronic circuits text or check the web; a good transistor switch tutorial is given at http://electronicsclub.info/transistorcircuits.htm. In a transistor, the (possibly large) collector-emitter current is controlled by the small base current through the relationship $I_C = h_{FE} I_B$. Since we are interested in using the transistors as switches, we will be operating in only the "on" (saturation) and "off" (cutoff) modes of the transistor. We can "turn off" the load circuit by setting $I_B = 0$. Likewise, if we know the desired current I_C and the gain h_{FE} (from datasheet), we can choose I_B so that the transistor is fully "on" (saturated). When the transistor is saturated, the voltage drop between collector and emitter is small and the collector-emitter junction can be modeled as a short circuit.

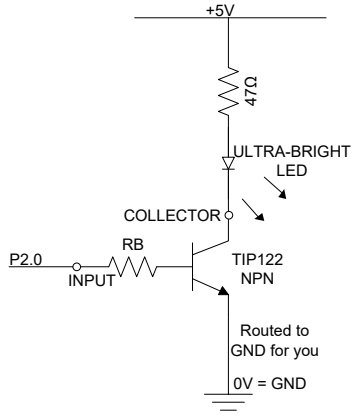


Figure 1

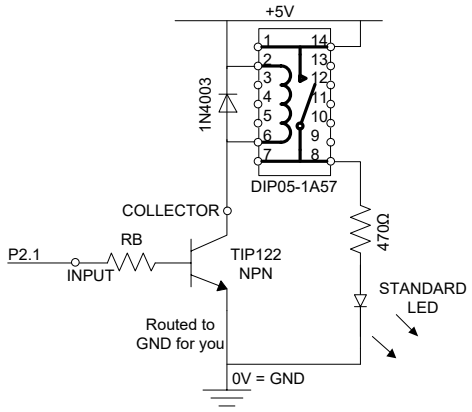
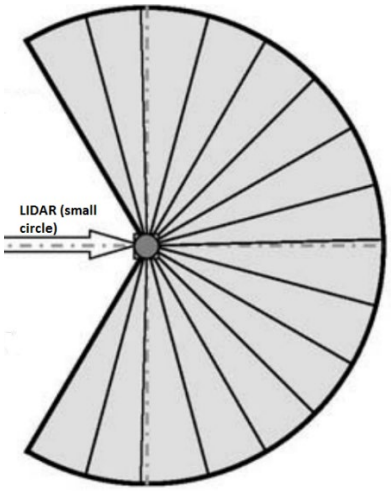


Figure 2

Refer to Figure 1 for the following example. If we want to use 5V to drive an ultra-bright LED in series with a 47Ω resistor, we can calculate $I_B \geq \frac{I_C}{h_{FE}} = \frac{(5-0.7)V / 47\Omega}{h_{FE}}$. For the TIP122 NPN Darlington Transistors you will use, $h_{FE} \geq 1000$. Therefore, $I_B \geq 0.09mA$. To realize this current with our microcontroller's 3.3V output, we need to determine an appropriate value for the resistor R_B. When $V_B > 1.4V$, the base-emitter junction of a Darlington transistor behaves like a diode with a 1.4V drop. Therefore, the drop across R_B is V_R = 3.3V-1.4V = 1.9V. Now, $R_B \leq \frac{V_R}{I_B} = \frac{1.9V}{0.09mA} \approx 20k\Omega$. Therefore, any resistor with a value less than 20kΩ would suffice.

For the sake of universality, we will use a resistor with a much smaller value (470Ω) so that we are able to drive larger loads if the need arises (say in your final HW project). Solder the two TIP122 chips onto the breakout board. Solder two 470Ω resistors in the places labeled "R_B". Also solder the LED circuit to the "COLLECT" terminal of one of the transistors as shown in Fig. 1. Note that 5V is available on your board from the pad labeled "5V 3A" when power is applied to the battery jack. Solder the relay circuit shown in Fig. 2 to the "COLLECT" terminal of the second transistor. (Note the current rating of this small relay's switch is 0.5 Amps so obviously you could drive a larger current load than a standard LED.) Solder P2.0 to the INPUT of the ultra-bright LED circuit and P2.1 to the INPUT of the relay circuit. Hand in your code and demonstrate it working to your TA. As a final part of this exercise, make hand drawings (2 drawings) of the transistor and relay circuits you soldered. You are being asked to do this to force you to look at the schematics given is Figure 1 and Figure 2 because many of you just blindly copy the soldering of the demo board. Your hand drawings will be very similar to Figure 1 and Figure 2, but make sure to look at the datasheet for the TIP122 and label the pins (1,2,3 and which is the Base, Emitter and Collector) of the TIP122.

7. For this assignment I would like you to expand on the state machine and pseudocode of HW #3 problem 6. Now instead of just having two distance readings from the LIDAR sensor, (one front distance and one front-right distance) you have access to the 228 distance readings from the LIDAR. The robot's LIDAR calculates 228 distance readings starting at -120 degrees on its right and sweeping to 120 degrees on its left. So the LIDAR produces a distance reading every 1.05 degrees starting at -120 and ending at 120. The LIDAR readings are given to you in an array of 228 elements. Element 113 is the distance reading straight in front of the robot. The below picture gives you an idea of the sweep of measurements, but note this picture is only showing 17 measurements.



Given these new measurements from the LIDAR, modify your obstacle avoidance section of your state machine with decisions and needed states to have the robot either left wall follow or right wall follow when it recognizes an obstacle with the LIDAR. Make sure to think about what you should do if you find an obstacle on your left, on your right and pretty much straight in front of you. As a final task, think about how you could recognize human legs and what would you have your robot do?