Colorado School of Mines
Department of Mechanical Engineering
MEGN 540: Mechatronics

Laboratory 3: Sensor Interfacing

Deliverable: Report out encoder readings and battery voltages. Setup a safety-net to inform the host (R-PI) if the batteries need to be charged.
Outcomes:

- Use external interrupts to detect digital-pin signal changes
- Apply quadrature logic to implement encoder readings
- Setup and use the A/D interface to monitor a voltage
- Use repeated monitoring to send health-status information based on physical readings
- Become more proficient with the state-machine programming flow and event handling.

Equipment:
· Lab Kit Mouse Keyboard Monitor Internet Connection

Background:
The atmega32u4 can trigger interrupts based on external pin changes. This is particularly useful for catching encoder ticks to track wheel rotations. In Addition, the atmega32u4 can monitor analog voltages though its A/D system. The Zumo-Car has a battery voltage line attached to the A/D, so the car can monitor its motor-battery charge level and issue warnings when the batteries need to be recharged. In this lab you'll be focused on interfacing with these two sensing modalities (counting ticks and reading voltages). You will also begin using the provided python3 based serial monitor plotting to visualize how these signals are changing in time.

Primary Functions to add: (note the file can be found in MEGN540/c_lib):
void Encoders_Init();
  Encoder.h/c
  Function Encoders_Init initializes the encoders, sets up the pin change interrupts, and zeros the initial encoder counts.
int32_t Counts_Left();
  Encoder.h/c
  Function Counts_Left returns the number of counts from the left encoder.
int32_t Counts_Right();
  Encoder.h/c
  Function Counts_Left returns the number of counts from the right encoder.
float Rad_Left();
  Encoder.h/c
  Function Rad_Left returns the number of radians for the left encoder.

float Rad_Right();

> Encoder.h/c
>
> Function Rad_Left returns the number of radians for the right encoder.

void Battery_Monitor_Init();

> Battery_Monitor.h/c
>
> Function Battery_Monitor_Init initializes the Battery Monitor to record the current battery voltages.

float Battery_Voltage();

> Battery_Monitor.h/c
>
> Function Battery_Voltage initiates the A/D measurement and returns the result for the battery voltage.

Assignment:

You will be implementing sensor interfaces for this assignment.

1. Review the provided functions, understand what they are doing, and how their called. Enable the functionality defined for the new functions.
2. Read the External Interrupt documentation by atmel32u4. Look up which pins/interrupts are connected to which encoder channels. Begin by filling in the helper-functions at the top of Encoder.h/c to read the A and B state of the pins. I suggest you begin by getting the INT6 side working first, then attempt the PCINT4 side.

   Note: A 8-bit microcontroller only can process 8 bits at a time. So, a 32 bit integer takes 8 clock cycles to write or read. Since interrupts are asynchronous, a simple write can be interrupted in the middle to process the interrupt. Thus, your data can get corrupted, if for example a bit-rollover occurred due to the summation. To handle, you need to make the data interfacing operation in Counts_Right effectively atomic by disabling interrupts, copying the current value, re-enabling interrupts, and then returning the solution. There is a good discussion on this in the Timer 1 documentation (14.2 Accessing 16-bit Registers). Take a look and use some the provided code snippets as inspiration.
3. Enable the message handling for the new ‘e’ and ‘E’ messages to report on encoder counts
    a. Plot the results using the provided plotting capability.
    b. What is the ratio between counts and radians? Is it what you’d expect from the Zumo documentation?
4. Once the Encoders are working, get the analog reading of battery voltage to work. Start by reading the Zumo Polo power discussion, then the ADC conversion section in the atmega documentation. Consult the wiring schematic to make a good choice for the settings as outlined in table 24.3. Despite all the documentation surrounding it, ADC reads are only a few lines of code, enable settings, initiate reading, wait until complete, record result. This is the one function where a *while(bit-not-set);* loop makes sense, since it only takes a few ADC clock cycles to complete a reading. Be careful about setting the prescalar for the ADC system, make sure you pay attention to 24.4 and do the math.
5. Enable the message handling for the new ‘b’ and ‘E’ messages to report on battery voltage.

   a. Plot the results when you turn on and off the switch, can you tell by the voltage if the switch is in the off location?
   b. Look up Ni-MH batteries, what is the minimum voltage (for 4) that is acceptable before damaging the battery? If your below that minimum with the switch on, send the following message to the serial to inform the user to charge the batteries:

```
struct __attribute__((__packed__)) {char let[7]; float volt;} msg =
{
        .let = {'B','A','T',' ','L','O','W'},
        .volt = bat_volt
};
// Send Warning to Serial that batteries need to be charged
usb_send_msg("c7sf",'!', &msg, sizeof(msg));
```

6. Using your filter code from class, measure the battery voltage every 2ms and create a digital low-pass filter to reduce the noise on the signal. Choose a cutoff frequency between 5 and 20 hz. Use this filtered signal to send to the host when queried and for determining if the batteries need to be charged.

Deliverables:

1. Demo: Make a 1-2-Minute Video that talks through your code and how you completed the lab. Make sure to show me how you handled the functions added to interface with the encoders and the battery voltage. Also show me how you are handling messaging and sending the battery warning.
2. Demo: Upload a video of you interacting with the car demoing each operation as well as error states (not sending a math operation).
   a. Make sure you show the car printing encoders, show me the real-time plot of them changing.
   b. Make sure you show the car sending battery voltages, show me the real-time plot of it changing and responding to a change in switch state.
   c. Make sure to demo at least one of the Lab 1 and Lab 2 commands.
3. Lab Report: Prepare a summary of what you have completed for this assignment. Follow a typical lab report format. Make sure that you:
   a. Include the information that you found important and critical while completing this lab and would be useful if you wanted to replicate it in the future.
   b. Indicate which pins on the electrical schematic were used in this project, and how you can tell which registers you needed to initialize and set. (What pins are connected to the interrupts, how can you tell the XOR vs the side A values? What prescalar did you choose and why? What settings/configurations did you choose and why?)
   c. Discuss your filter design for the battery. What type you picked, what order, and how you calculated the coefficients. Discuss how well it removes noise from the signal. Why don’t we need to filter the encoder readings?

Host -> Device Communication Messing Definition

	CMD char	CMD Hex	# Bytes	Format	Resulting Action
Lab 1	~	0x7e	1	c	Reset the program (return 0)
	*	0x2a	9	cff	Multiply the two floats and return the product.
	/	0x2f	9	cff	Divide the first float by the second and return the result.
	+	0x2b	9	cff	Sum the floats and return the result.
	-	0x2d	9	cff	Subtract the second from the first and return the result.
	UNDEF				If the command char is unrecognized, flush the input buffer, and respond with a '?' (0x3f) character
Lab 2	t	0x54	2	cc	Return the time it requested followed by the time to complete the action specified by the second imput char. 0x00 -> Time Now 0x01 -> Time to send a float 0x02 -> Time to complete a full loop iteration Others as you define
	T	0x74	6	ccf	Return the time it requested followed by the time to complete the action specified by the second imput char and returns the time every X milliseconds. If the time is zero or negative it canceles the request.
Lab 3	e	0x65	1	c	Return the encoder counts for the left and right motors
	E	0X45	5	cf	Return the left and right encoder values every X milliseconds specified by float sent. If the float sent is less-than-or-equal-to zero, this cancels the send request.
	b	0x62	1	c	Return the current battery voltage level
	B	0x42	5	cf	Returns the current battery voltage leve every X seconds as sepcified by the float sent. If the float is less-than-or-equal-to zero 0, the request is canceled.

Device -> Host Communication Messaging Definition [MSG Length] [Format C-Str][Host Initiating CMD Char][DATA] **Response Format**	**Function Call**
0x8 'c' 'f' 0x0 '*' [float]	*usb_send_msg*("cf", '*', &data, *sizeof*(data))
0x8 'c' 'f' 0x0 '/' [float]	*usb_send_msg*("cf", '/', &data, *sizeof*(data))
0x8 'c' 'f' 0x0 '+' [float]	*usb_send_msg*("cf", '+', &data, *sizeof*(data))
0x8 'c' 'f' 0x0 '-' [float]	*usb_send_msg*("cf", '-', &data, *sizeof*(data))
0x5 'c' 'c' 0x0 '?' [char]	*usb_send_msg*("cc", '?', &data, *sizeof*(data))
0xA 'c' 'H' 'f' 0x0 't' [uint8] [float]	*usb_send_msg*("cHf", 't', &data, *sizeof*(data))
0xA 'c' 'H' 'f' 0x0 'T' [uint8] [float]	*usb_send_msg*("cHf", 'T', &data, *sizeof*(data))
0xD 'c' 'f' 'f' 0x0 'e' [float] [float]	*usb_send_msg*("cff", 'e', &data, *sizeof*(data))
0xD 'c' 'f' 'f' 0x0 'E' [float] [float]	*usb_send_msg*("cff", 'E', &data, *sizeof*(data))
0x8 'c' 'f' 0x0 'b' [float]	*usb_send_msg*("cff", 'e', &data, *sizeof*(data))
0x8 'c' 'f' 0x0 'b' [float]	*usb_send_msg*("cff", 'B', &data, *sizeof*(data))

Every Second (if battery power is low), the board should send:

0xC 'c' '7' 's' 0x0 '!' 'B' 'A' 'T' ' ' 'L' 'O' 'W'	struct {char let[7]; float volts; } data = { .let = {'B','A','T',' ','L','O', 'W'}, .volt = [float] }; usb_send_msg("c7sf", '!', &data, sizeof(data));