CSCE 436/836: Embedded Systems
Lab 3: Interrupts, Communication, Sensor Fusion

Instructor: Carrick Detweiler
carrick _at_ cse.unl.edu
University of Nebraska-Lincoln
Spring 2011

Started: Feb 22, 2011
Due: March 10, 2011

1 Overview

This lab will culminate in a series of competitions between each of the groups. The competition will involve driving automated sequences. You will start this lab by implementing interrupts to handle serial communication and also to read the gyro at a fixed and known rate. You will then implement I^2C communication to read the magnetometer and to communicate between the two processors on the hoverboard. With this in place, you will be able to perform sensor fusion on the 5V Atmel to maintain an accurate estimate of the pose of the hovercraft. Finally, you start using a radio to communicate with the 5V processor to enable easier remote control.

Many of the specific implementation decisions are left to you in this lab. Make sure to plan ahead as a group as to the approach you will take. In addition, some items (such as implementing master/slave I^2C communication and the protocol) are discussed in separate sections of the lab. You may want to do these together instead of going through the lab in a linear order.

The high level overview of the system you will create by the end of the lab is as follows. The 3.3V Atmel will control thrusters and will read the gyro at a fixed rate. Since the gyro gives a reading based on the rotational rate of change, the 3.3V Atmel will integrate these readings to come up with an estimate of the current angle. The 5V Atmel will use I^2C to read the estimated angle from the 3.3V processor. The 5V Atmel will also read the magnetometer over I^2C and fuse this data together with the gyro readings to compensate for gyro drift.

The 5V Atmel will receive desired angle and thrust or trajectory information over a radio link. It will use the fused angle estimate and the desired settings to send thruster commands to the 3.3V Atmel.

2 Safety

Remember and review the safety instructions from the previous labs.

3 Interrupts [15pts.]

In this section of the lab you will use interrupts to send and receive data over the serial port. In addition, you will use the timers to generate a periodic interrupt that you will use to read the gyro at a fixed rate.

3.1 Serial Interrupts

In this section of the lab, you should implement interrupts to handle serial communication. You **must** implement code to handle receiving bytes, but you do not have to implement the transmitter interrupt, although it is recommended that you do[1]. This will let your code spend less time waiting to send and receive serial data. You should certainly implement this on the 5V Atmel, however, you may not need this on the 3.3V Atmel (as must communication will be over I^{2}C). Read the datasheet to determine how to enable receive and transmit interrupts. The receive interrupt triggers when a new byte is received. For sending data, you should use the `UDREn` interrupt, which indicates when the transmit buffer is empty. You have to be careful to disable this interrupt when you have nothing left to send as otherwise you will repeatedly receive this interrupt. Similarly, you should only enable this interrupt if you have more than one byte to send[2].

You will also need to enable global interrupts before any interrupts will work. To enable interrupts, use the command `sei()` and to disable them use `cli()`. These functions are accessed by including `<avr/interrupt.h>`.

To tell the `C` compiler what code should be executed when an interrupt is triggered (the function to jump to in the vectored interrupt table), you can use the following code:

```
#include <avr/io.h>
#include <avr/interrupt.h>

/**
 * Receive interrupt handler for UART0
 **/
SIGNAL(USART0_RX_vect){
}

/**
 * Interrupt handler for when the UART0 transmit buffer is empty
 **/
SIGNAL(USART0_UDRE_vect){
}
```

While the code within the `SIGNAL` is executing, global interrupts are disabled. Similar code will let you use interrupts over `UART1`. It is worth looking at the header files that are included. These are located in `/usr/lib/avr/include/avr`. The most useful is `iom1284p.h`, which `io.h` includes. This contains all of the pin and interrupt definitions for our processor. If you search for `USART0_RX_vect`, you will see all of the other interrupts that are available.

In order to use interrupts, you will need to have some sort of space or queue to transfer data between the main program and the interrupt handlers. In the code for this lab is a file `queue.h` which implements a fairly efficient FIFO (first in, first out) queue. Note that the macros are not all interrupt safe, so you will need to disable and then reenable interrupts when calling these functions outside of interrupt handlers (where interrupts are already disabled). The idea is that you will replace your existing `uartReceiveByte(...)` code with that checks the queue to see if there is data in it and then returns it. The interrupt handler adds data to this queue. (Same idea for the `uartSendByte(...)`).
Question: What do you do if the receive queue is full?
Question: What do you do if the transmit queue is full? Answer this even if you did not implement this.
Question: Look at the assembly code in `main.lss` for the uart transmit interrupt handler. Estimate the number of clock cycles needed to execute the interrupt handler (do not forget the instructions needed to save/restore the old PC, jump to the interrupt handler, etc.). How does this compare to the number of cycles the processor would be idle if you instead just busy waited for a byte to be sent? If you did not implement the TX interrupt, use the RX interrupt code to estimate the number of cycle. Please explain how

[1]Receive interrupts are critical as without them it is easy to miss receiving serial data if your code is processing elsewhere. Transmit interrupts are nice as they significantly reduce idle time sending, but are not as necessary because no data will be lost without them.

[2]There is also an interrupt to indicate when the transmit is complete (TXCn). This indicates when the data has truly been sent (not just that the TX buffer is empty). This is used for non-duplex protocols like RS485 that need to know when the data is actually off the TX wire.

you counted the number of instructions needed for the interrupt handler. At what baud rate would it make more sense to busy wait than to use a transmit interrupt[3]?
Question: Make sure to discuss the details of your implementation and any challenges you had to overcome.

3.2 Reading Gyro with Interrupts

In the last lab you read the gyro with the A2D ports on the 3.3V Atmel. You also configured the timers to output PWM signals. In this section of the lab, you should implement reading the A2D gyro pins at a fixed rate determined by the timers. To do this, configure one of your timers to generate an interrupt when the counter resets[4]. Every time you receive this interrupt you should read the gyro[5]. Reading at a constant rate will make it easier to fuse the gyro data in Section 6.1.
Question: Which timer did you use to generate interrupts and what interrupt on that timer did you use?
Question: What is the frequency that you receive the interrupt to perform measurements?
Question: How long does reading the gyro take? Is it possible that you will lose I^2C data while you are reading the A2D pins? If so, how could you avoid losing data (there are multiple ways to do this)? You may need to implement this.

4 I^2C [15pts.]

In this section you will configure the 5V Atmel as the I^2C master. The 5V Atmel will read the magnetometer and also communicate with the 3.3V Atmel over I^2C. This will require configuring the 3.3V Atmel as an I^2C slave.

In the lab code there are a number of files, including `i2c.c` and `i2c.h`. This is an I^2C implementation from an opensource project for the Atmel. There are additional header files that are also needed: `avrlibdefs.h`, `avrlibtypes.h`, and `global.h`. You should look over all of these files, in particular `i2c.c` and `i2c.h`.
Question: Look through this code. Based on what you see, does this code support multi-master I^2C operation? If so, how?

4.1 Master

The 5V Atmel will be the only I^2C master on the hoverboard (although it is possible to have both processors be masters). As you look through `i2c.h`, you will see that there are interrupt and non-interrupt based master functions. You should probably use the non-interrupt based versions as these are easier to use and as the serial port uses interrupts, we don't need to worry about loosing serial or other data.
Question: Describe what you needed to do to configure the 5V Atmel as an I^2C master.

4.2 Reading Magnetometer

You can read the magnetometer from the 5V Atmel over the I^2C bus. The magnetometer (HMC5843) has a number of internal registers that control the operation of the device. Using I^2C you can read and write these registers. After reading the magnetometer datasheet, you should write a function that reads all of the magnetometer's registers and prints them out over the serial port. This will help verify the configuration of the device. Note that this will also be your first real test of the I^2C master code you wrote.
Question: What is the address of the magnetometer?

[3]It almost always makes sense to use an RX interrupt as you do not know when you might receive bytes. Often processors will have a hardware TX and RX queue, which means there need to be fewer interrupts. Some processors also let you use DMA so you don't even need an interrupt to transfer the data.

[4]This should be at approximately 500Hz.

[5]An alternative approach is to have the interrupt increment a global counter. The main loop can then be read whenever the counter changes. This has the advantage that very little time is spent in the interrupt handler, which is generally a good thing. It has the disadvantage that interrupts (or other code in the main loop) could slow down the reading of the gyro. Of course, with the global counter you can detect large delays in the reading.

Question: Verify that the default values listed in the datasheet for the magnetometer configuration registers match those that you read. What are they and what mode does the magnetometer start out in?

After making sure that you can read registers, you are nearly done as writing the registers is just about the same, except only some of the registers can be written. There are two different modes for obtaining measurements, continuous- and single-conversion mode. You should configure it to continuous-conversion mode so that you do not need to continuously set the single conversion flag.

Question: The magnetometer can update it's measurements at different rates. What rate did you choose? Why?

Complete code to obtain measurements periodically. You should configure one of the timers to generate a periodic interrupt to read data from the magnetometer.

Question: What timer did you use and what data reading rate does this produce?

Question: What happens if you read from the magnetometer faster than it updates?

Question: How long does it take to actually read all of the measured values from the magnetometer?

The values that you read from the magnetometer are the raw magnetic field values in the X, Y, and Z direction. You need to convert this into a compass heading to be useful. Honeywell (the manufacturer of our magnetometer) has an application note describing how to do this conversion. It is called *AN203 Compass Heading Using Magnetometers*, find this document and use it to determine how to convert the raw readings into rotational degrees.

Question: Describe how you convert from the raw magnetometer readings to a compass heading on the Atmel.

Question: Perform some experiments rotating your hoverboard and recording the magnetic field. Plot a number of these trials (perhaps at different locations or on different surfaces) and include the raw values as well as your degree measurements. You should include some known angles (perhaps every 45 degrees). Interpret the results.

4.3 Slave

Configure the 3.3V Atmel to be a slave on the I^{2}C bus. Again, you will use the `i2c.c` and `i2c.h` code to do this. The way the slave receiver works is that you register callback functions for the I^{2}C code to call when data is received or data is requested. These are called from an interrupt generated from the I^{2}C hardware, so make sure you have interrupts enabled for this code to work. In Section 5.1 you will create a protocol to communicate between the 3.3V and 5V Atmels.

Question: Describe how you setup the I^{2}C slave code.

5 Communication [15pts.]

In this section you will create a protocol to communicate with over the I^{2}C bus between the two processors and also use a radio link to be able to communicate remotely with the 5V Atmel.

5.1 I^{2}C

For this section of the lab, you should create a protocol to communicate between the 5V and 3.3V processors over the I^{2}C bus. The I^{2}C protocol has a read (receive) and write (send) flag built in. To read data from the slave, you first need to write to the slave to tell it where you want to read from. One approach is to always have the first byte of the write command indicate what should happen next. If the master wishes to read data, then this byte would indicate where the next read should get data from. If the master wants to write, then the first byte would indicate what the remaining bytes in the packet should do. For this lab, you should implement commands to read the current gyro angle estimate and perhaps the most recent raw gyro output. You will also need commands to tell the 3.3V Atmel what the thruster setting should be. There may be others commands you want to implement as well.

Question: Describe the protocol you developed to communicate between the two processors.

Question: Does receiving/sending I^{2}C messages on the 3.3V Atmel interfere with the timer that you are using to read the gyro values? Hint: you can look at the timer interrupt flags. If it does, what impact does this have and how could you mitigate it?

5.2 Radio

For this lab, you have a pair of radios ZigBee radios. One connects to the 5V Atmel on UART1 (at 19200 baud rate) and the other connects via a USB-to-serial converter to your computer. This should be a completely transparent link that acts like the direct cable connection. This is because each pair of radios is on a different radio channel. Thus, you can use it to basically replace the cabled connection (and use `screen` to talk directly to the board).

It is possible, however, that other people in the building may be using the same radios on the same channel. So if you see interference, this may be the cause. In general, whenever using a radio, you should implement a protocol that has some checks to verify the received data is correct and not corrupted. However, you don't need to do this for this lab.

Note that you can connect both the cable connector and radio at the same time. If you plug in the cable first, it will be located at `/dev/ttyUSB0` and the radio will be located on `/dev/ttyUSB1`. This way you can still reprogram the boards with the program scripts and communicate over the radio by opening `screen` on `/dev/ttyUSB1`.
Question: Describe your experience with the radios.

6 Sensor Fusion [15pts.]

In this section, you will fuse the gyro and magnetometer readings together to compensate for the gyro drift. Note that you can use floating point operations on the Atmel, but since the processor does not have a floating point unit, these operations will require many clock cycles to complete. So you should try to minimize the number of floating point operations you perform or just use integer or fixed point arithmetic. Regardless, you should be aware the additional processing time that floating point operations take.

6.1 Gyro

The first thing you should do is to integrate the gyro angular rate readings on the 3.3V Atmel to produce an absolute angle. Recall from the previous lab that there is some initial offset that indicates no rotation on the gyro. You will probably want to implement some calibration routine that runs on startup (or on some I^{2}C command) that reads the gyro for a little bit to determine the nominal value. Using the gyro datasheet you can determine the rotational rate based on the number of mV offset from the nominal value. Since you are reading the gyro at a fixed (and known) rate, you can use this information to integrate these readings to come up with an absolute angle estimate (assuming you start at angle zero initially). Implement this code. You should also consider the fact that the gyro has a 1x and a 4x output. Additionally, you may want to average or filter the raw gyro measurements if you found a significant amount of noise in the readings.
Question: Describe your implementation of your gyro integration code. How do you use the 1x and 4x outputs?
Question: If you do not move the board, how fast does the angular error accumulate?
Question: Rotate the hoverboard 360 degrees by hand. How accurate is the gyro? Do this several times. What happens to the error if you rotate using the thrusters?

6.2 Gyro and Magnetometer

We will use the magnetometer to compensate for the gyro drift. There are a number of data fusion algorithms that you can use to do this[6]. The main idea of all of them is that since the gyro is much faster than the magnetometer (and not susceptible to interference from large chunks of metal or other environmental factors), you should mostly rely on the gyro reading and just use the magnetometer to help account for drift. One way to do this is to maintain an offset value between the "real" heading angle, A_r, and that reported by the gyro integration, A_g. Call this offset O and the magnetometer angle A_m. Then:

$$A_r = A_g + O \tag{1}$$

You can then update O using:

$$O = O * k + (A_r - A_m) * (1 - k) \tag{2}$$

Where k is between zero to one. If it is closer to one, the offset will change slowly. You should adjust k so that it is the largest value that gives a relatively stable output. Feel free to use other data fusion techniques as this is certainly not the only, nor optimal approach.
Question: Describe how you fuse the gyro and magnetometer data.
Question: What happens to the magnetometer readings when you turn on the thrusters but hold the angle constant? If it changes significantly, you may want to weight the magnetometer less when the thrusters are active.

7 Competition [20pts.]

On the due date of this lab, we will hold a competition between the groups in performing a variety of automated control tasks. In this section, we will first discuss some methods for controlling the hovercraft and then we will describe the competition tasks.

7.1 Control

The goal is to be able to control the angle and trajectory of the hovercraft as well as possible. To control the angle of the hovercraft, you will use the gyro, magnetometer, and the fused data. One approach is to have the 5V processor issue raw thrust commands to the 3.3V processor based on the fused angle estimate. This may work, however, you may want to have finer control over the angle. An alternative approach is to have the 3.3V processor try to maintain a target angle based just on the gyro and an offset angle that is periodically updated by the 5V processor. Exactly how you implement the control is up to you. In general, you will want to be able to set a target angle and have the hovercraft rotate to that angle as quickly as possible, without oscillating or loosing angular accuracy. Often a PID (proportional, integral, derivative) controller is used for this, although probably a P or PD controller will work well. Refer to the wikipedia article on PID controllers or ask the instructor for more details.

You will also want to compensate for most of the translational motions that rotating introduces. This will involve manual calibration of your system for different rotational speeds.
Question: Describe the details of how you implemented rotational control.

You will also want to be able to control the direction and distance that your hovercraft travels. You should be able to translate in any direction without rotation. Based on your translational thruster layout, come up with a mapping that allows translation in any direction. You may need to adjust this based on experimental results to compensate for errors in actual thruster placement. You should also determine the amount of thrust and time to travel a fixed distance.
Question: Describe the details of how you implemented translational control.

[6]One popular method is to implement a Kalman filter. If you are familiar with Kalman filters, you are welcome to implement one.

7.2 Competition Tasks

At the end of this lab, the class will compete on a number of tasks. Part of your grade for the lab will be based on the results of the competition. The tasks will be:

- Spinning from a start angle to a particular angle from -360 to +360 degrees. You should translate as little as possible.
- Translate along a particular vector between 0 and 360 degrees without rotating. You must travel at least 0.5 meters, but you may travel more. The angle between the start point and the final rest position will determine the angle.
- Translate to a particular point between 0.5 and 2.5 meters from the starting point.
- Drive a square trajectory with sides between 0.5 and 1.5 meters.

The speed in which your robot completes this task will also be taken into account. You should configure your hovercraft to perform these tasks autonomously. You will be given a short period of time before each task to update your code, but ideally you would be able to set the goal using serial commands.
Question: Describe how you approached each of these problems. Include data from trial runs that characterize how well you can perform these tasks. **Note,** that your final report is due before the competition. Make sure to do trials beforehand to characterize your hovercraft in performing these tasks and include this in the report.

8 To Hand In

You should designate one person from your group as the point person for this lab (each person needs to do this at least once over the semester). This person is responsible for organizing and handing in the report, but everyone must contribute to writing the text. You should list all group members and indicate who was the point person on this lab. Your lab should be submitted by email before the start of class on the due date. A pdf formatted document is preferred.

Your lab report should have an introduction and conclusion and address the various questions (highlighted as ***Question:***) throughout the lab in detail. It should be well written and have a logical flow. Including pictures, charts, and graphs may be useful in explaining the results. There is no set page limit, but you should make sure to answer questions in detail and explain how you arrived at your decisions. You are also welcome to add additional insights and material to the lab beyond answering the required questions. The clarity, organization, grammar, and completeness of the report is worth **20 points** of your lab report grade.
Question: Please include your code with the lab report. Note that you will receive deductions if your code is not reasonably well commented. You should comment the code as you write it, do not leave writing comments until the end. I recommend structuring your code such that for every section of this lab, a different function is called from the main loop. Then you will be able to comment out just one function to change between sections in the lab and you won't have to delete working code from previous parts of the lab.
Question: For everyone in your group how many hours did each person spend on this part and the lab in total? Did you divide the work, if so how? Work on everything together?
Question: Please discuss and highlight any areas of this lab that you found unclear or difficult.