COSC 4P78 – Lab 6

For this lab, you'll be using incremental encoders to measure how far your robot's motors have moved. You'll then use this ability to let the robot repeat actions (without relying on precise timing/speed).
As a bonus, you'll be accomplishing this by assembling a very simple 'intelligent' sensor, using a second microcontroller for the sensor-polling.

This Week's Task:
You'll be creating a simple I^2C-based sensor out of the provided wheel encoders, ATtiny85, 4.7Kohm resistors, and an electrolytic capacitor. You'll then attach them to your robot, and program your Hackduino to do the following:

1. Move some distance forward, turn 90°, move a bit further, and turn again; all at some speed
2. Repeat step #1, with the same distances, but a different speed

The idea is to use the encoders to ensure that the robot can travel a measured distance, irrespective of what speed it happens to be travelling at. Pick whatever two speeds you like (I'd suggest around 50 and 80 for your `analogWrite`'s).

DFRobot Encoders:
You've been provided with some very basic IR-based wheel encoders. Note that these are *not* quadrature encoders! If you use your H-Bridge fully, you'll have to account for the lack of 'subtraction'!
Their operation is simple: each sensor is just a basic IR break-beam, powered by the 5V and GND pins, returning a signal over the data line. Technically, they should probably be analog inputs, but we'll just be using them digitally.
The actual installation may be a bit of a nuisance. You can find the “instructions” here:
http://www.robotshop.com/media/files/pdf/installation-method-fit0029.pdf

Note that you'll need to not only physically remove the motors, but also swap out the old bolts for longer ones. When it comes time to attaching the actual sensors and tightening the nuts holding them in, it's been suggested that you should tighten the top nut first, since it's easier (and thus keeps the motor assembly more stable when you're tightening the bottom one from below.

- You'll almost certainly need to use some needle-nose pliers to reach the bottom nut

Be immensely careful with the pieces — particularly the encoder disc — as they can be fragile.
There should be some extra parts in the bag that accompanied the sensor, including some spare washers. Feel free to use them, or not, to get everything aligned.

ATtiny85:
You've been provided with a very simple microcontroller. It has many of the same features as your ATmega328p (including PWM, ADC, and general digital IO), but is mostly just… smaller.
It also doesn't have the same communications options as the ATmega, but it does have a USI (Universal Serial Interface) that can simulate much of what's needed.

The ATtiny also lacks the bootloader we normally use with Hackduino/Arduino, so you can only program it with an external device.

Your ATtiny85 has already been programmed. You don't need to do that. It's ready to go.
(You can see what it's doing on the last page)

However, you do need to actually wire it up, so let's briefly discuss that, shall we?

- If you have enough space on your motor board, you may wish to solder it onto that
  - **DO NOT POWER THE ATTINY WITH 8V!** Run a separate 5V from your Hackduino!
    - Or there might be a spare 7805 or two left, if you want to use that, for some reason
  - Add an extra electrolytic capacitor for the 5V power supply on the board (the one on your Hackduino might not be sufficient to guarantee that both will run reliably)
- Alternatively, this lab definitely does work with the ATtiny simply mounted in the mini-breadboard. It gets a bit crowded, but there's enough room, and it communicates just fine
- You'll need pullup resistors for your SDA and SCL lines on your ATtiny
  - This just means you should put a 4.7Kohm resistor between pin 5 (SDA) and +5V, and another between pin 7 (SCL) and +5V
- The left sensor's data pin should connect to the ATtiny's Pin 3 (technically the ATtiny's A2 analog-in for its ADC, but for both the ATtiny and the ATmega, you can repurpose ADCs as digital-in's, if you want)
- The right sensor's data pin connects to the ATtiny's Pin 2
- Connect the ground (refer to the Atmel cheat sheet you've been using for all of the labs), and you're ready to hook up the I^2C lines to your Hackduino

I^2C/TWI:

In the section above, we identified the SDA and SCL lines, but where are those on the ATmega?
Actually, SDA is the same connection as your A4 analog-in, and SCL is A5. Why? Because of reasons.

So long as you remembered to include the pull-ups, all you need to do is connect SDA to SDA and SCL to SCL.

At the end of this lab is the sample program that's already loaded onto the ATtiny.
It's pretty simple:

- The main loop repeatedly polls the current state of the two sensors
  - Since the beam is either broken or not, the input is HIGH or LOW
- Whenever the state changes, it adds a tick to a tally for that sensor
  - Technically, one byte is shared for the tally; the most significant nybble is for the left sensor, and the least significant nybble is for the right
- If a byte is requested by the I^2C master, then two things happen:
  - The tally byte is yielded to the master
  - The tally is reset to zero
    - So each request effectively asks “how many ticks occurred since we last talked?”

To actually get the readings from each sensor, just split up the two nybbles.

Note: In a proper (good) I^2C device of this nature, you'd normally have two-way communication. The master would first specify which *register* in the slave was being queried (for example, just for the left sensor), followed by the slave returning only the requested information. However, this version is adequate to our needs.

Tips:

- You can write your own ATtiny code, but it isn't advisable
  - You'd need the TinyCore and TinyWire libraries, and would probably use ArduinoISP to upload
- Note that a nybble is only 4 bits, so you need to make sure you read from the sensor before 16 pulses have elapsed, or else you'll get an overflow
- If your robot actually does end up exactly where it started, I'll be immensely surprised
  - Drift isn't just fine; it's actually expected
- If you're soldering in the ATtiny, I'd strongly recommend soldering in a socket instead
  - But the socket doesn't work well in a breadboard, so skip it for that

ATtiny's I2C Slave Device Code:

```
/**
 Props to 'rambo' on github (code largely shamelessly stolen from his example)
 */
#define I2C_SLAVE_ADDRESS 0x7 // the 7-bit address (remember to change this when adapting this example)
// Get this from https://github.com/rambo/TinyWire
#include <TinyWireS.h>
// Don't think this is still necessary, but far too lazy to find out
#ifndef TWI_RX_BUFFER_SIZE
#define TWI_RX_BUFFER_SIZE ( 16 )
#endif

//For more complicated devices, you'd have some form of register array
//You'd send a value (or more) to the device to tell it the address it should use for your next request

volatile uint8_t tallies;
int lastLeft,lastRight,left,right;

//Only send one byte at a time
//(Refer to rambo's examples for how to get around this limitation)
void requestEvent()
{
    uint8_t toSend=tallies;
    tallies=0;
    TinyWireS.send(toSend);
}

void setup()
{
    //Slight delay to allow hackduino to boot up? Maybe?
    //A2 (pin 3) will be for left sensor
    //A3 (pin 2) will be for right sensor
    //Used in digital read mode, but already INPUT because they're also ADC lines
    //Don't forget to pull SDA (pin 5) and SCL (pin 7) up with 4.7k's

    tallies=0;
    lastLeft=digitalRead(A2);
    lastRight=digitalRead(A3);

    TinyWireS.begin(I2C_SLAVE_ADDRESS);
    TinyWireS.onRequest(requestEvent);
}

void loop()
{
  left=digitalRead(A2);right=digitalRead(A3);
  if (left!=lastLeft)
    tallies=((tallies+16)&0xF0)|(tallies&0x0F);
  if (right!=lastRight)
    tallies=((tallies+1)&0x0F)|(tallies&0xF0);
  lastLeft=left;lastRight=right;
  TinyWireS_stop_check();
}
```

Sample I2C Master Device (Hackduino) Code:

```
#include <Wire.h>

void setup() {
  Wire.begin();
  Serial.begin(9600);
}

void loop() {
  int dummy;
  while (Serial.available()) {
    Wire.requestFrom(0x7, 1);  // request 1 byte from slave device #7
    dummy=Serial.read();
    while (Wire.available()) { //make sure you got something
      byte c = Wire.read();
      Serial.print(c);         // print it as a number; not a character code
      Serial.println();
    }
  }
  delay(500);
}
```

Note that this just takes a reading whenever you send a single character (just type a character into the *serial monitor* and hit *enter*).