


Physics 124: Lecture 3

Motors: Servo; DC; Stepper

Messing with PWM (and 2-way serial)

The Motor Shield

Three Types (for us)

- Servo motor
  - PWM sets position, used for R/C planes, cars, etc.
  - 180° range limit, typically
  - 5 V supply
- Stepper motor
  - For precise angular control or speed control
  - Can rotate indefinitely
  - Lots of holding torque
- DC motor
  - simplest technology; give up on precise control
  - good when you just need something to SPIN!

When any old PWM won't do

- The function `analogWrite()` gives you easy control over the duty cycle of PWM output
  - but no control at all over frequency
- Consider the Hitec servo motors we'll be using:

Pulse Data
All Hitec servos require 3-5V peak to peak square wave pulse. Pulse duration is from 0.9mS to 2.1mS with 1.5mS as center. The pulse refreshes at 50Hz (20mS).
Voltage Range
All Hitec Servos can be operated within a 4.8V-6V. range.
Only the HS-50 operates exclusively with 4 Nicad cells (4.8 volt).
Wire Color Meanings
On all Hitec servos the Black wire is 'ground', the Red wire (center) is 'power' and the third wire is 'signal'.

- Wants a 50 Hz pulse rate, and a duty cycle from 4.5% to 10.5% (11/255 to 27/255) to drive full range

What frequency is Arduino PWM?

- Depends on which output is used
- Pins 5 and 6: default ~977 Hz
  - 16 MHz clock rate divided by 2^{14} = 16384
- Pins 3, 9, 10, 11: default 488 Hz
  - 16 MHz / 2^{15}
- Neither is at all like the 50 Hz we need for the servo motor

What choice do we have?

- We can change the clock divider on any of three counters internal to the ATMega328
  - timer/counter 0, 1, and 2
  - consider this snippet from the register map:

(0xB2)	TCNT2	Timer/Counter2 (8-bit)								164
(0xB1)	TCCR2B	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	163
(0xB0)	TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	160

  - note in particular the lowest 3 bits in TCCR2B
  - setting these according to the following rubric scales speed

Table 18-9. Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(No prescaling)
0	1	0	clk_{T2S}/8 (From prescaler)
0	1	1	clk_{T2S}/32 (From prescaler)
1	0	0	clk_{T2S}/64 (From prescaler)
1	0	1	clk_{T2S}/128 (From prescaler)
1	1	0	clk_{T2S}/256 (From prescaler)
1	1	1	clk_{T2S}/1024 (From prescaler)

Valid Divider Options

PWM pins	Register	scaler values	frequencies (Hz)
5, 6	TCCR0B	1, 2, 3, 4, 5	62500, 7812, 977, 244, 61.0
9, 10	TCCR1B	1, 2, 3, 4, 5	31250, 3906, 488, 122, 30.5
3, 11	TCCR2B	1, 2, 3, 4, 5, 6, 7	31250, 3906, 977, 488, 244, 122, 30.5

- Defaults are shown in red
- Obviously, choices are limited, and we can't precisely hit our 50 Hz target
- Closest is to use timer 0 with divider option 5 (61 Hz)
- 0.9 to 2.1 ms pulses correspond to 14/255 to 33/255
  - only 20 possible steps by this scheme

How to set divider and change PWM freq.

- It's actually not that hard
  - can do in setup or in main loop

```
TCCR0B = TCCR0B & 0b11111000 | 0x05;
```

- Broken Down:
  - modifying TCCR0B associated with pins 5 & 6
  - & is bitwise AND operator
  - 0b11111000 is binary mask, saying "keep first five as-is"
  - while zeroing final three bits (because 0 AND anything is 0)
  - | is bitwise OR operator, effectively combining two pieces
  - 0x05 is hex for 5, which will select 61.0 Hz on Timer0
  - if TCCR0B started as vwxyzabc, it ends up as vwxyz101

Code to interactively explore PWM frequencies

- Will use serial communications in both directions

```
const int LED = 5;  // or any PWM pin (3,5,6,9,10,11)
char ch;            // holds character for serial command

void setup()
{
  pinMode(LED,OUTPUT);     // need to config for output
  Serial.begin(9600);
}
```

  - continued on next slide…

…continued

```
void loop()
{
  analogWrite(LED,128);    // 50% makes freq. meas. easier
  if (Serial.available()){ // check if incoming (to chip)
    ch = Serial.read();    // read single character
    if (ch >='0' && ch <='7'){    // valid range
      if (LED == 3 || LED == 11){ // will use timer2
        TCCR2B = TCCR2B & 0b11111000 | int(ch - '0');
        Serial.print("Switching pin ");
        Serial.print(LED);
        Serial.print(" to setting ");
        Serial.println(ch);
      }
    }
    if (ch >='0' && ch <='5'){    // valid for other timers
      if (LED == 5 || LED == 6){ // will use timer0
        TCCR0B = TCCR0B & 0b11111000 | int(ch – '0');
        Serial.print(same stuff as before…);
      }
      if (LED == 9 || LED == 10){ // uses timer1
        TCCR1B etc.
      } } } }          // would indent more cleanly if space
```


Using the interactive program

- Use serial monitor (Tools: Serial Monitor)
  - make sure baud rate in lower right is same as in `setup()`
  - can send characters too
  - in this case, type single digit and return (or press send)
  - get back message like:
    - `Switching pin 11 to setting 6`
  - and should see frequency change accordingly

Rigging a Servo to sort-of work

- Original motivation was getting a 50 Hz servo to work

```
const int SERVO = 5;
char ch;                // for interactive serial control
int level = 23;         // 23 is 1.5 ms; 14 is 0.9; 33 is 2.1

void setup()
{
  pinMode(SERVO, OUTPUT);  // set servo pin for output
  Serial.begin(9600);
  TCCR0B = TCCR0B & 0b11111000 | 0x05;  // for 61 Hz
  analogWrite(SERVO, level);            // start centered
}
```

  - continued next slide…

Continuation: main loop

```
void loop()
{
  if (Serial.available()){ // check if incoming serial data
    ch = Serial.read();    // read single character
    if (ch >='0' && ch <='9'){    // use 10 step range for demo
      level = map(ch-'0',0,9,14,33);    // map 0-9 onto 14-33
      analogWrite(SERVO, level);        // send to servo
      Serial.print("Setting servo level to: ");
      Serial.println(level);
    }
  }
  delay(20);              // interactive program, so slow
}
```

- Being lazy and only accepting single-character commands, limited to ten values, mapping onto 20
  - the `map()` function is useful here
  - the `ch - '0'` does "ASCII subtraction"

A better (and easier!) way

- The previous approach was a poor fit
  - poor match to frequency, and not much resolution
- Arduino has a library specifically for this: Servo.h
- Various libraries come with the Arduino distribution
  - in /Applications/Arduino.app/Contents/Resources/Java/libraries on my Mac

```
EEPROM/         Firmata/        SD/             Servo/          Stepper/
Ethernet/       LiquidCrystal/  SPI/            SoftwareSerial/ Wire/
```

  - Handles stepper and servo motors, LCDs, memory storage in either EEPROM (on-board) or SD card; several common communication protocols (ethernet—for use with shield, SPI, 2-wire, and emulated serial)
  - can look at code as much as you want

Example using Servo library

- Watch how easy: one degree resolution

```
// servo_test . . . . slew servo back and forth thru 180 deg
#include <Servo.h>

Servo hitec;            // instantiate a servo
int deg;                // where is servo (in degrees)

void setup(){
  hitec.attach(9,620,2280); // servo physically hooked to pin 9
  // 620, 2280 are min, max pulse duration in microseconds
  // default is 544, 2400; here tuned to give 0 deg and 180 deg
}
void loop(){
  for(deg = 0; deg <= 180; deg++){  // visit full range
    hitec.write(deg);               // send servo to deg
    delay(20);
  }
  for(deg = 180; deg >= 0; deg--){  // return trip
    hitec.write(deg);               // send servo to deg
    delay(20);
  }
}
```

Available Servo Methods

- `attach`(*pin*)
  - Attaches a servo motor to an i/o pin.
- `attach`(*pin*, *min*, *max*)
  - Attaches to a pin setting min and max values in microseconds; default min is 544, max is 2400
- `write`(*deg*)
  - Sets the servo angle in degrees. (invalid angle that is valid as pulse in microseconds is treated as microseconds)
- `writeMicroseconds`(*us*)
  - Sets the servo pulse width in microseconds (gives very high resolution)
- `read()`
  - Gets the last written servo pulse width as an angle between 0 and 180.
- `readMicroseconds()`
  - Gets the last written servo pulse width in microseconds
- `attached()`
  - Returns true if there is a servo attached.
- `detach()`
  - Stops an attached servo from pulsing its i/o pin.

Libraries: Documentation

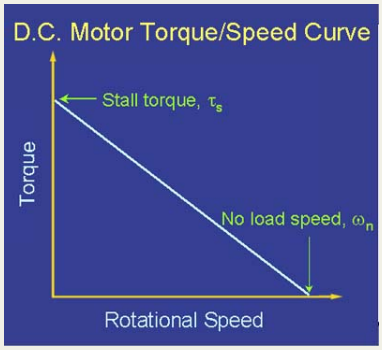
- Learn how to use standard libraries at:
  - http://arduino.cc/en/Reference/Libraries
- But also a number of contributed libraries
- Upside: work and deep understanding already done
- Downside: will you learn anything by picking up pre-made sophisticated pieces?

DC Motor

- Coil to produce magnetic field, on rotating shaft
- Permanent magnet or fixed electromagnet
- Commutator to switch polarity of rotating magnet as it revolves
  - the "carrot" is always out front (and will also get push from behind if switchover is timed right)

DC Torque-speed Curve

- See http://lancet.mit.edu/motors/motors3.html

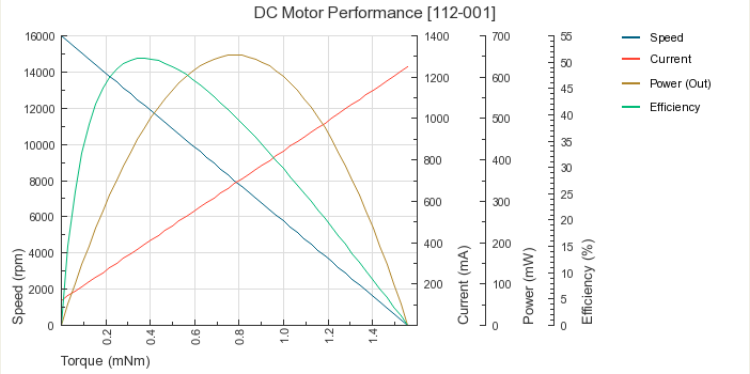


- Stalls at τ_s; no load at ω_n
- Output mechanical power is $\tau\omega$
  - area of rectangle touching curve
  - max power is then $P_{max} = ¼\ \tau_s\omega_n$

Electrical Expectations

- Winding has resistance, R, typically in the 10 Ω range
- If provided a constant voltage, V
  - winding eats power $P_w = V^2/R$
  - motor delivers $P_m = \tau\omega$
  - current required is $I_{tot} = (P_w + P_m)/V$
- At max power output ($P_m = ¼\ \tau_s\omega_n$)
  - turns out winding loss is comparable, for ~50% efficiency

Example 2.4 V motor



- Random online spec for 2.4 V motor (beware flipped axes)
  - note at power max 0.0008 Nm; 0.7 A; 8000 RPM (837 rad/s)
    - total consumption 2.4×0.7 = 1.68 W
    - output mechanical power 0.0008×837 = 0.67 W; efficiency 40%
    - at constant V = 2.4, total power consumption rises → 3 W toward stall
    - 1.25 A at stall implies winding $R = V/I$ = 1.9 Ω

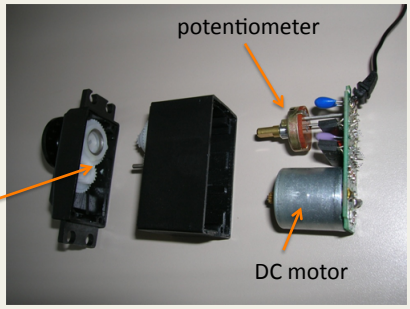
Another random example

	VOLTAGE	NO LOAD		AT MAXIMUM EFFICIENCY					STALL		
MODEL	NOMINAL	SPEED	CURRENT	SPEED	CURRENT	TORQUE		OUTPUT	TORGUE		CURRENT
	V	r/min	A	r/min	A	g.cm	mN.m	W	g.cm	mN.m	A
RX-RF370CH-15370	12	5500	0.026	4840	0.17	25.3	2.48	1.25	187	18.3	1.06

- Note provision of stall torque and no-load speed
  - suggests max output power of ¼×2π(5500)/60×0.0183 = 2.6 W
  - about half this at max efficiency point
  - 2π(4840)/60×0.00248 = 1.25 W
  - at max efficiency, 0.17×12 = 2.04 W, suggesting 61% eff.
  - implied coil resistance 12/1.06 ≈ 11 Ω (judged at stall)
- Lesson: for DC motors, electrical current depends on loading condition
  - current is maximum when motor straining against stall

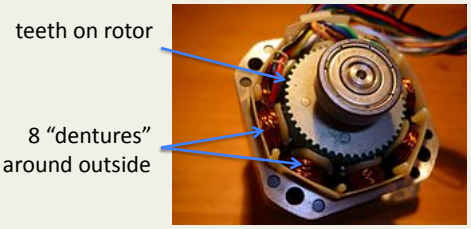
Servo Internals

- A Servo motor is just a seriously gear-reduced DC motor with a feedback mechanism (e.g, potentiometer) to shut it off when it is satisfied with its position
  - and drive motor faster or slower depending on how far off target



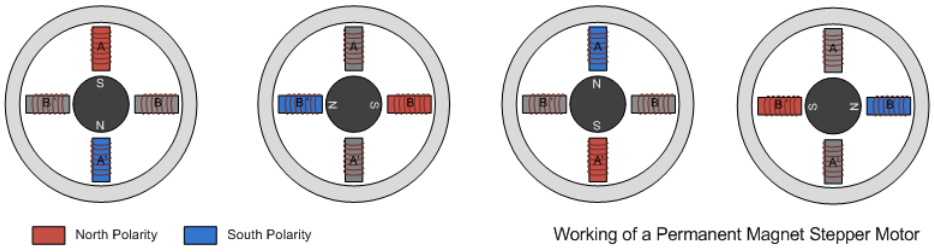
Clever Steppers

- Stepper motors work in baby steps
- In simplest version, there are two DC windings
  - typically arranged in numerous loops around casing
  - depending on direction of current flow, field is reversible
- Rotor has permanent magnets periodically arranged
  - but a differing number from the external coils



A Carefully Choreographed Sequence

- Four different combinations can be presented to the two coils (A & B; each bi-directional)
  - each combination attracts the rotor to a (usu. slightly) different position/phase
  - stepping through these combinations in sequence walks the rotor by the hand to the next step

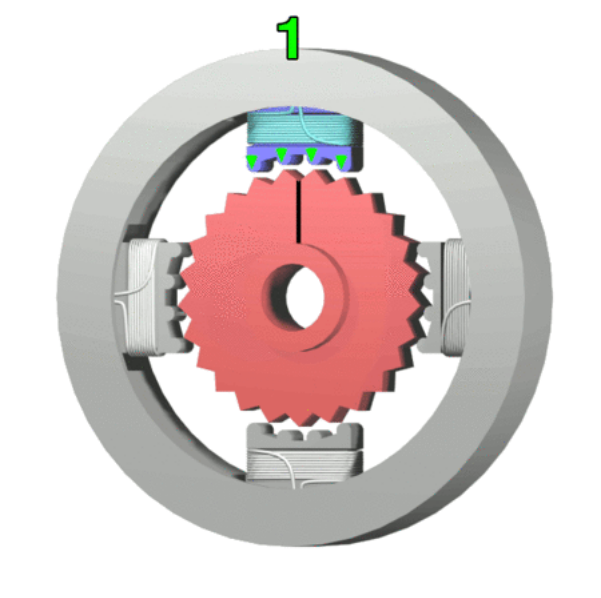


Working of a Permanent Magnet Stepper Motor

In practice, rotor has many poles around (in teeth, often), so each step is much finer.

Toothed Animation

- Note teeth are not phased with "dentures" all the way around
  - each is 90° from neighbor
- This sequence is typical of center-tap steppers
  - can activate one side of coil at a time
- Note usually have more than four "dentures" around outside



Stepping Schemes

- Can go in full steps, half steps, or even microstep
  - full step is where one coil is on and has full attention of rotor
  - if two adjacent coils are on, they "split" position of rotor
  - so half-stepping allows finer control, but higher current draw
    - every other step doubles nominal current
  - instead of coils being all on or all off, can apply differing currents (or PWM) to each; called microstepping
    - so can select a continuous range of positions between full steps
- Obviously, controlling a stepper motor is more complicated than our other options
  - must manage states of coils, and step through sequence sensibly

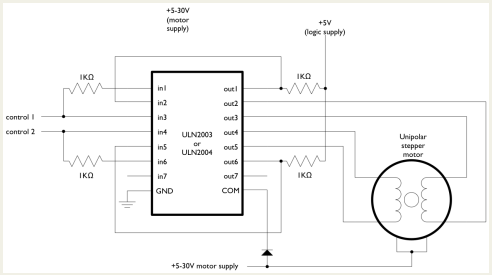
The Stepper Library

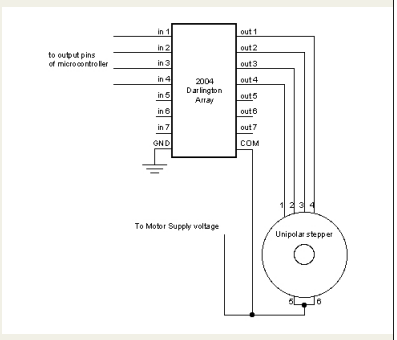
- Part of the Arduino Standard Library set
- Available commands:
  - Stepper(steps, pin1, pin2)
  - Stepper(steps, pin1, pin2, pin3, pin4)
  - setSpeed(rpm)
  - step(steps)
- But Arduino cannot drive stepper directly
  - can't handle current
  - need transistors to control current flow
  - arrangement called H-bridge ideally suited

Example stripped code

```
#include <Stepper.h>
#define STEPS 100             // change for your stepper

Stepper stepper(STEPS, 8, 9, 10, 11);

int previous = 0;

void setup(){
  stepper.setSpeed(30);           // 30 RPM
}

void loop(){
  int val = analogRead(0);    // get the sensor value

  // move a number of steps equal to the change in the
  // sensor reading
  stepper.step(val - previous);

  // remember the previous value of the sensor
  previous = val;
}
```

A Unipolar Stepper Motor: Center Tap

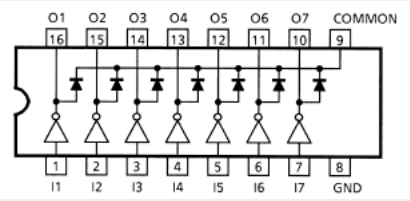


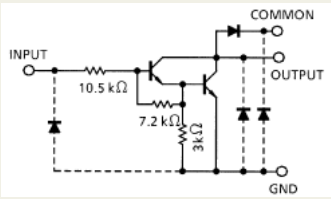


- A unipolar stepper has a center tap for each coil
  - half of coil can be activated at a time
  - can drive with two Arduino pins (left arrangement)
  - or four pins (right)
  - both use ULN2004 Darlington Array

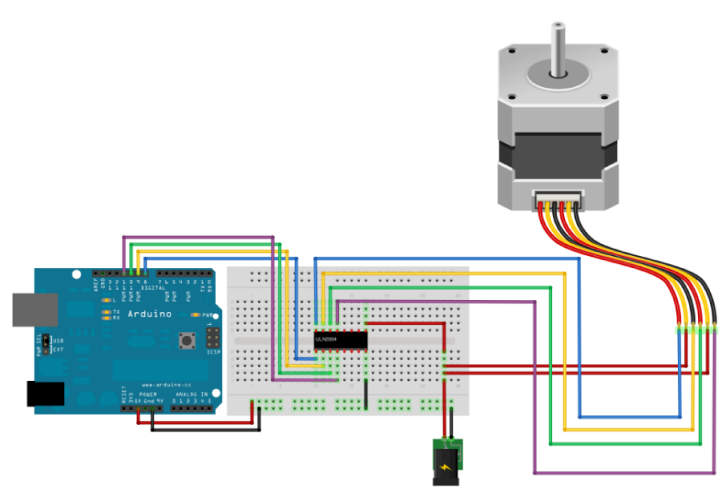
What's in the Darlington Array?

- The ULN2004 array provides buffers for each line to handle current demand
- Each channel is essentially a pair of transistors in a Darlington configuration
  - when input goes high, the output will be pulled down near ground
  - which then presents motor with voltage drop across coil (COMMON is at the supply voltage)



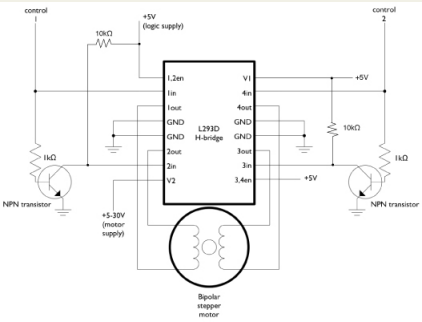


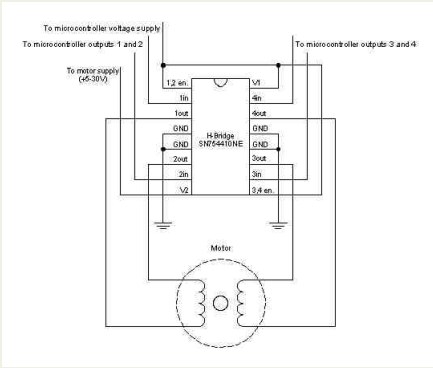
Unipolar hookup; control with four pins



- Yellow motor leads are center tap, connected to external power supply (jack hanging off bottom)

A Bipolar Stepper Motor: No Center Tap

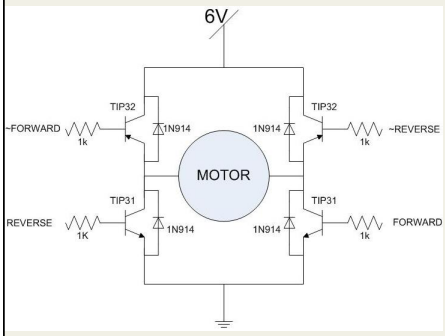


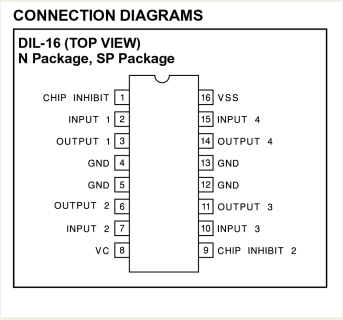


- In this case, the coil must see one side at ground while the other is at the supply voltage
- At left is 2-pin control; right is 4-pin control
  - H-bridge is L293D or equiv.
  - transistors just make for logic inversion (1in opp. 2in, etc.)

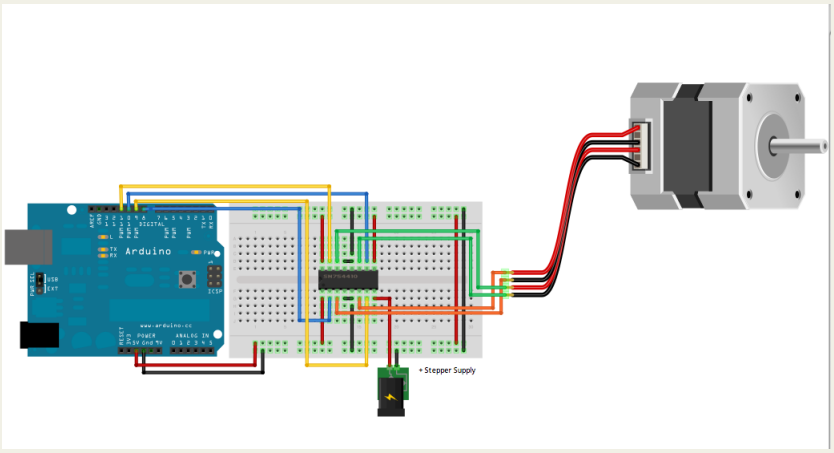
H-bridge Internals

- An H-bridge is so-called because of the arrangement of transistors with a motor coil spanning across
  - two transistors (diagonally opposite) will conduct at a time, with the motor coil in between





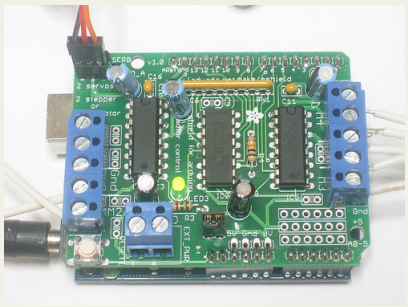
Bipolar Hookup; control with four pins



- Input supply shown as jack hanging off bottom

The Motor Shield

- We have kit shields that can drive a "motor party"
  - 2 servos plus 2 steppers, or
  - 2 servos plus 4 DC motors, or
  - 2 servos plus 2 DC motors plus 1 stepper
- Allows external power supply: motors can take a lot of juice



The Motor Shield's Associated Library

- See instructions at
  - http://learn.adafruit.com/adafruit-motor-shield
  - Install library linked from above site
  - follow instructions found at top of above page
  - may need to make directory called `libraries` in the place where your Arduino sketches are stored
    - specified in Arduino preferences
  - and store in it the unpacked libraries as the directory AFMotor
- Once installed, just include in your sketch:
  - `#include <AFMotor.h>`
- Open included examples to get going quickly

Example Code

- Stepper Commands in AFMotor
  - `#include <AFMotor.h>`
    - grab library
  - `AF_Stepper my_stepper(`*# S/R*, *port*`);`
    - `my_stepper` is arbitrary name you want to call motor
    - arguments are steps per revolution, which shield port (1 or 2)
  - `my_stepper.setSpeed(30);`
    - set RPM of motor for large moves (here 30 RPM)
  - `my_stepper.step(`*NSTEPS*, *DIRECTION*, *STEP_TYPE*`);`
    - take NSTEPS steps, either `FORWARD` or `BACKWARD`
    - can do `SINGLE`, `DOUBLE`, `INTERLEAVE`, `MICROSTEP`
  - `my_stepper.release();`
    - turn off coils for free motion

Step Types

- SINGLE
  - one lead at a time energized, in sequence 3, 2, 4, 1
    - as counted downward on left port (port 1) on motor shield
  - normal step size
- DOUBLE
  - two leads at a time are energized: 1/3, 3/2, 2/4, 4/1
    - splits position of previous steps; tug of war
  - normal step size, but twice the current, power, torque
- INTERLEAVE
  - combines both above: 1/3, 3, 3/2, 2, 2/4, 4, 4/1, 1
  - steps are half-size, alternating between single current and double current (so 50% more power than SINGLE)
- MICROSTEP
  - uses PWM to smoothly ramp from off to energized
  - in principle can be used to go anywhere between hard steps

DC Motors with motor shield/AFMotor

- DC motors are handled with the following commands
  - `#include <AFMotor.h>`
    - grab library
  - `AF_DCMotor mymotor(`*port*`);`
    - port is 1, 2, 3, or 4 according to M1, M2, M3, M4 on shield
  - `mymotor.setSpeed(200);`
    - just a PWM value (0–255) to moderate voltage sent to motor
    - not RPM, not load-independent, etc. — crude control
  - `mymotor.run(`*DIRECTION*`);`
    - `FORWARD`, `BACKWARD`, or `RELEASE`
    - depends, of course, on hookup direction

Servos on the Shield

- Two Servo hookups are provided on the shield
- Really just power, ground, and signal control
  - signal control is Arduino pins 9 and 10
  - use Servo.h standard library
  - pin 9 → Servo2 on shield; pin 10 → Servo1 on shield

Announcements

- TA office hours:
  - Clayton: M 3-4; Tu 1-2
  - Paul: F 2-3; M 2-3
- Turn in prev. week's lab by start of next lab period, at 2PM (day dep. on Tue/Wed section)
  - can drop in slot on TA room in back of MHA 3544 anytime
- Midterm to verify basic understanding of Arduino coding
  - blank paper, will tell you to make Arduino do some simple task (at the level of first week labs, without complex logic aspects)