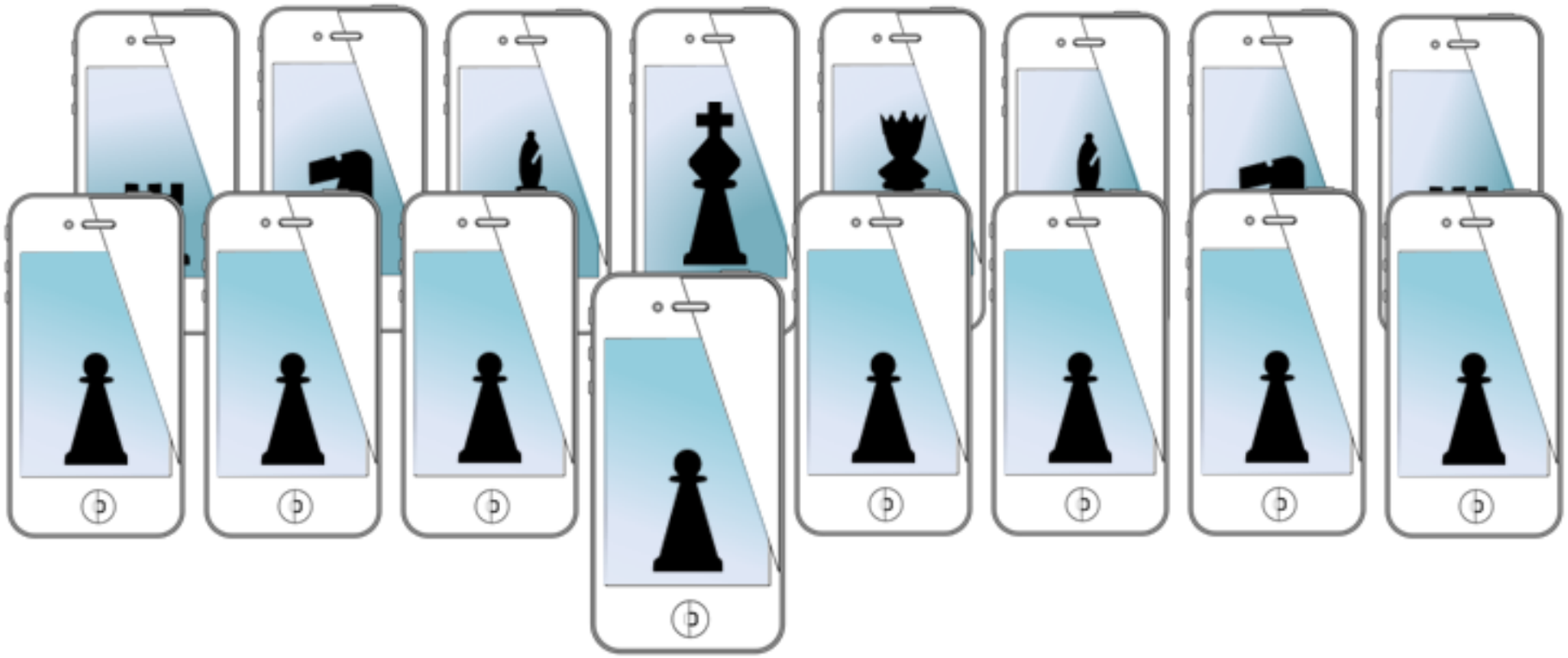
MOBILE SENSING LEARNING & CONTROL



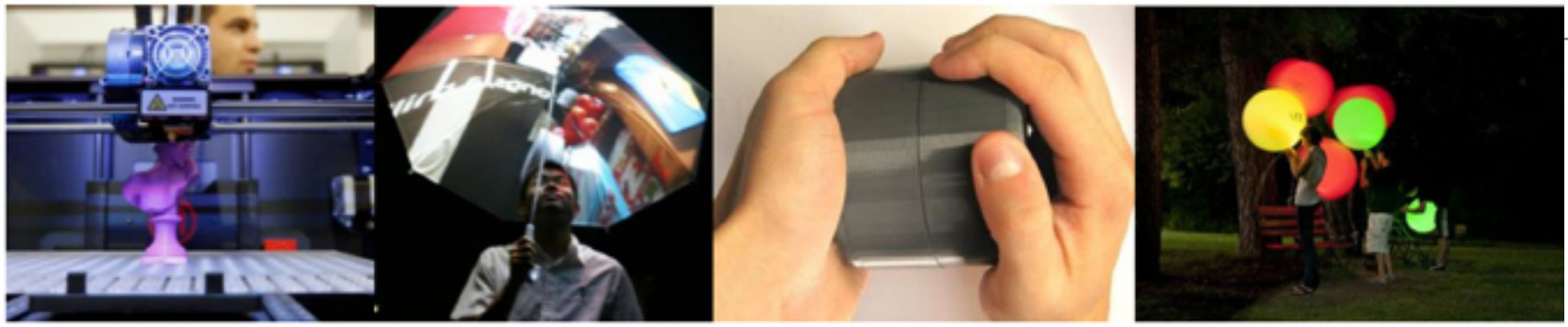
CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture sixteen: control and bluetooth

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University

UBICOMP



- offering fall semester 2014
- MW 2:00-3:20

Ubiquitous Computing

CSE 5390 & CSE 7390

Eric C. Larson

eclarson@lyle.smu.edu

- History of UbiComp
- Human Computer Interaction
- Advanced Prototyping
  - 3D printing
  - software radio
  - scikit-learn
- Sensing and machine learning
- Wireless technologies
- Input methods for UbiComp
- Activity sensing
- Location tracking
- Wearable computing
- Assistive technology
- mobile health
- Sustainability and feedback
- Evaluation techniques
- Privacy and Security

course logistics

- A5 is due next week
- its time to start nailing down a final project
  - look at the final project proposal requirements
  - proposal is due with A6
  - talk with group about the MOD
- …so, we will talk more as A6 approaches
- …but, A6 should be a first draft of proposal
  - highly open ended lab

agenda

- microcontrollers
  - motor control
  - shields
- bluetooth communication
  - with arduino
  - primer
  - iOS

assignment 5

Assignment Five - Bluetooth Arduino Sensing

Create an iOS application using the Bluetooth template and the Arduino Bluetooth Shield that:

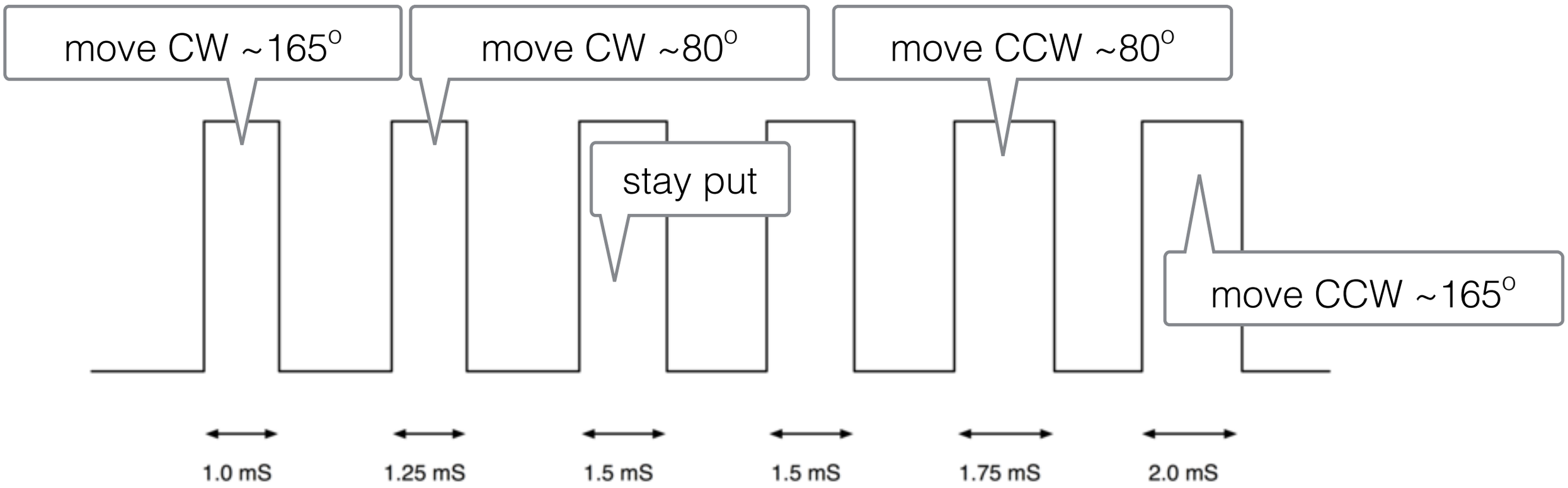
- Reads and displays sensor data from two or more sensors attached to the arduino
  - use proper sampling rates to display the sensor output
  - use proper dynamic range and voltage references
- Sends two or more control commands to the microcontroller that change the behavior of the arduino
  - make the controls change something noticeable in the operation of the Arduino
  - the output must make use of a PWM signal (implemented in hardware)
- The Arduino sketch should:
  - use interrupts
    - for example, use a button as input
    - the interrupt must change something noticeable in the operation of the Arduino
  - use digital outputs
    - for example to control LED(s), pin 13 is already setup for this
  - use PWM
  - use the ADC

Use the tips from lecture for proper interfacing of sensors, buttons, timers, PWM, and interrupts.

NOT allowed to use the **FIRMATA** ONLY

continuous servos

- each pulse is a new command
- pulse width is angle to move from current angle
  - servo can rotate all the way around
  - counter clockwise (CCW) or clockwise (CW)

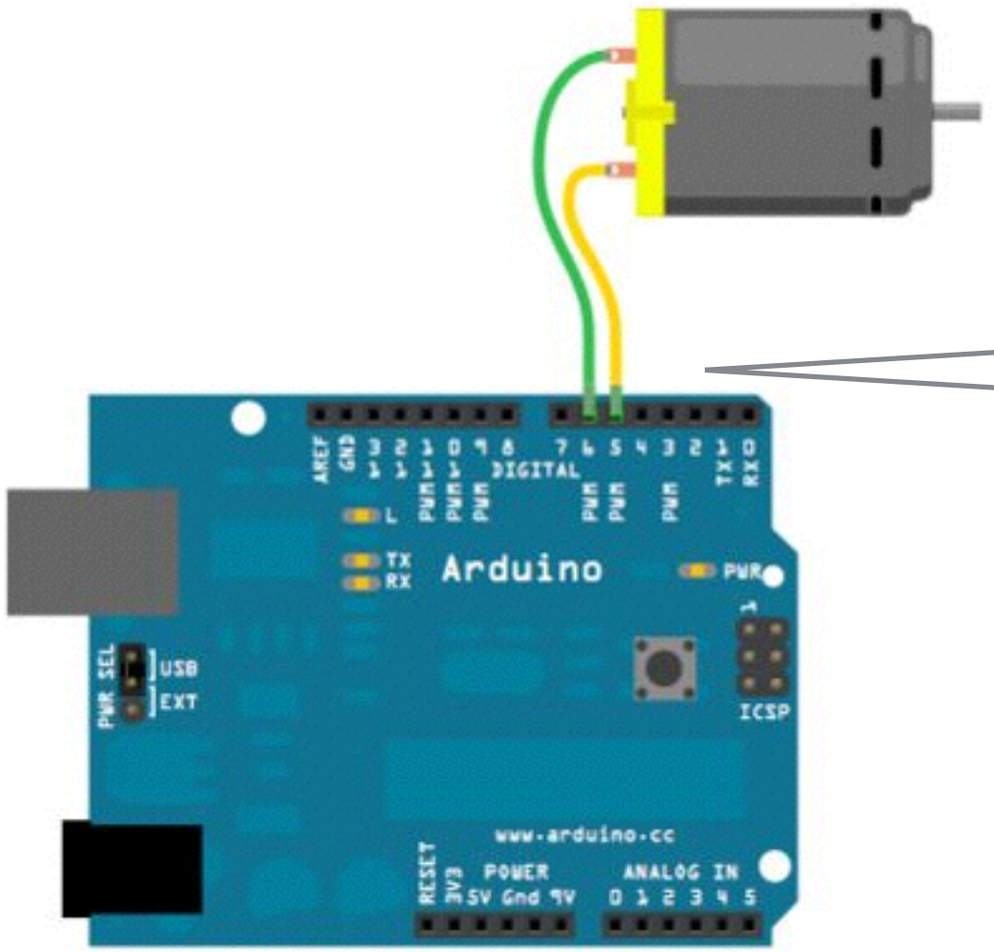


continuous servo demo

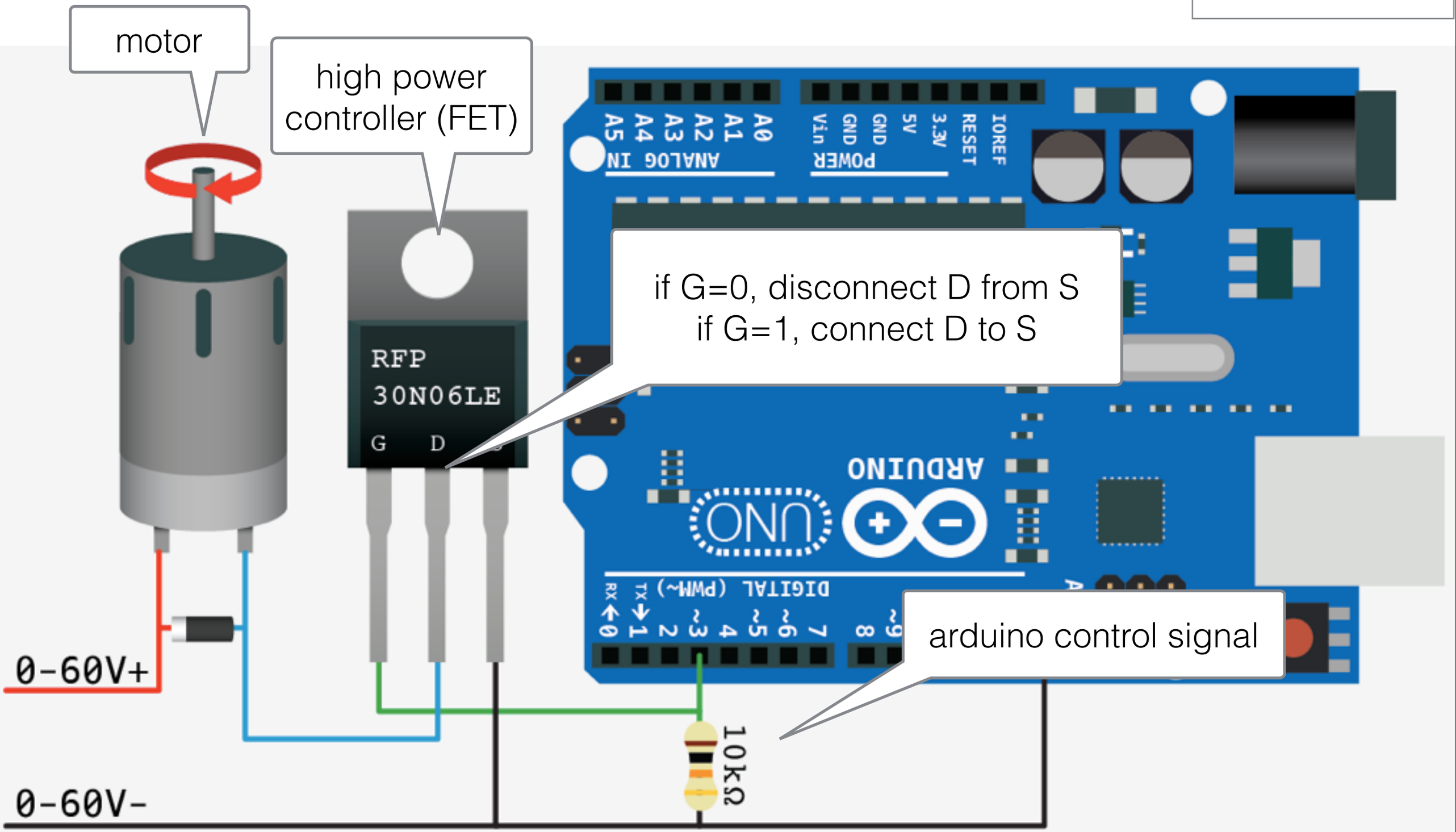
DC motor control

- give power to the motor in bursts
  - longer bursts == more power!
- just like LED brightness
- …or like the dimmer switch in your home
- motors use a lot of power!
  - arduino board will not cut it
  - arduino just provides the control signal to motor

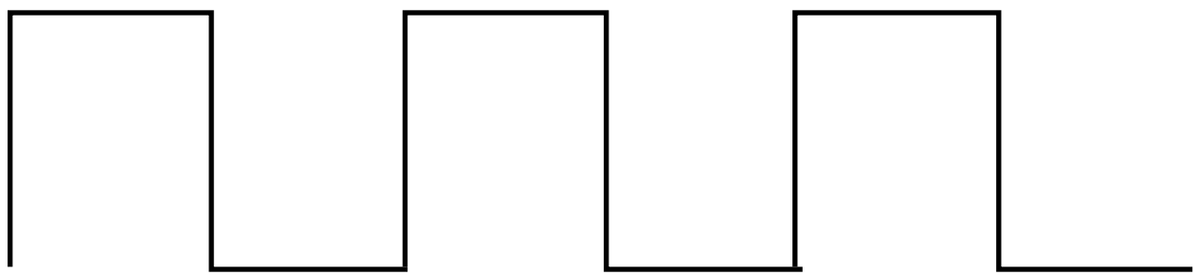
DC motor control



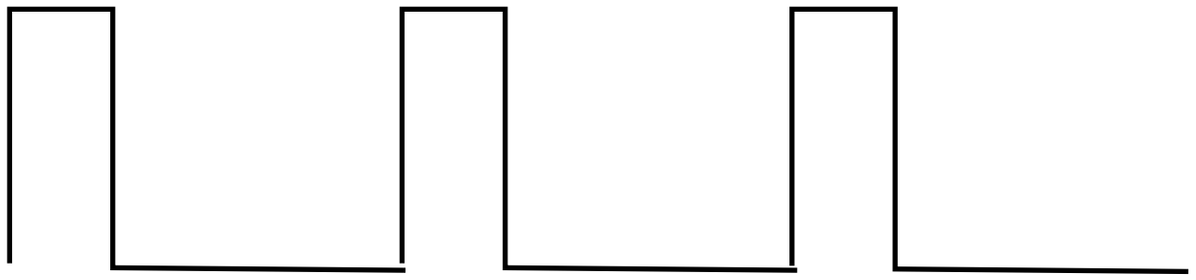
DC motor control



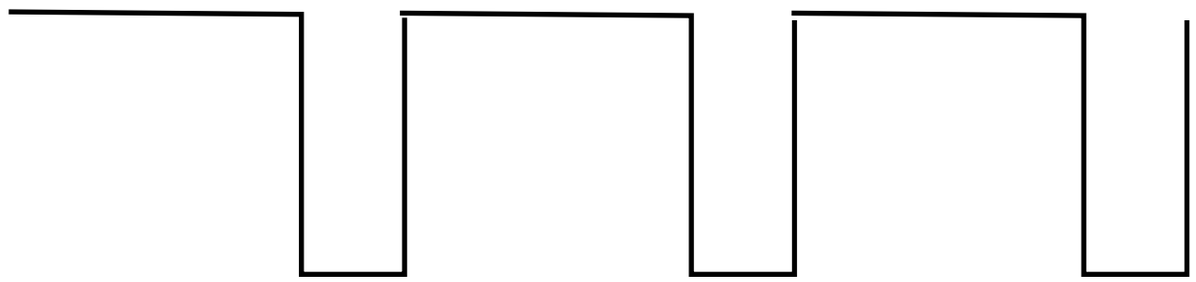
DC motor control



medium



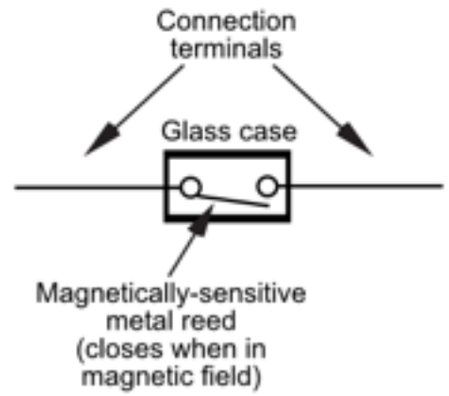
slow

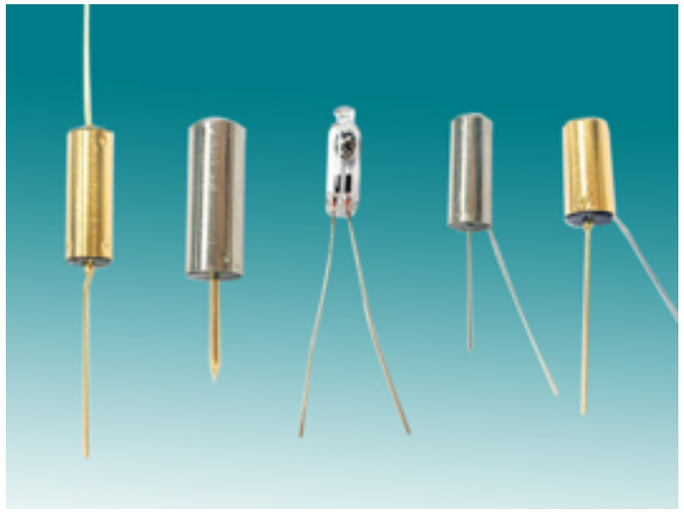


fast

other sensors

- binary
  - reed switches
  - buttons
  - ball switches
- analog
  - hall effect sensors
  - capacitive touch
  - accelerometers
- http://playground.arduino.cc/Main/InterfacingWithHardware#arstat

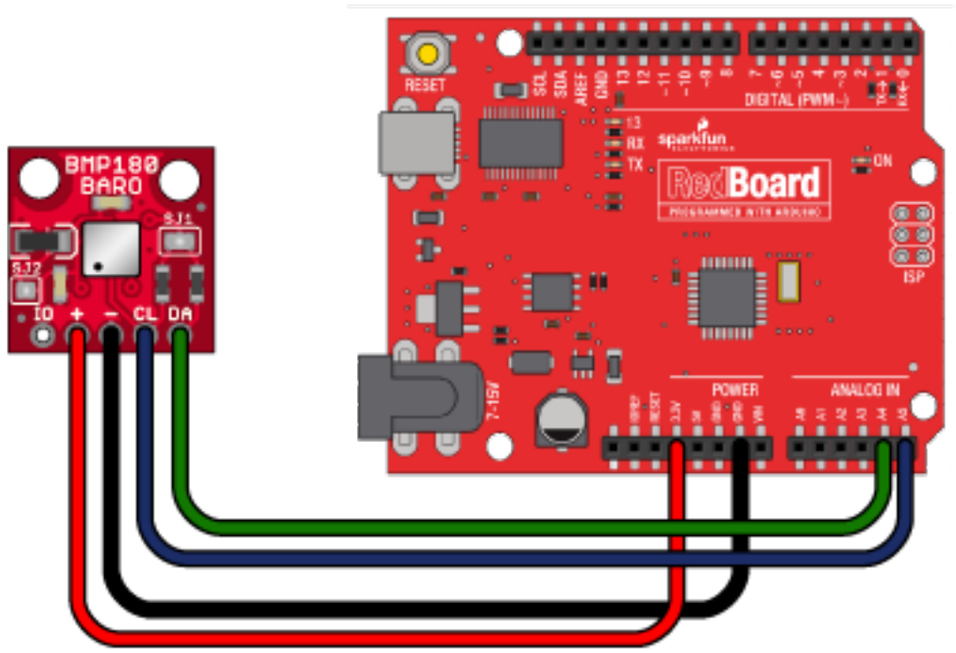


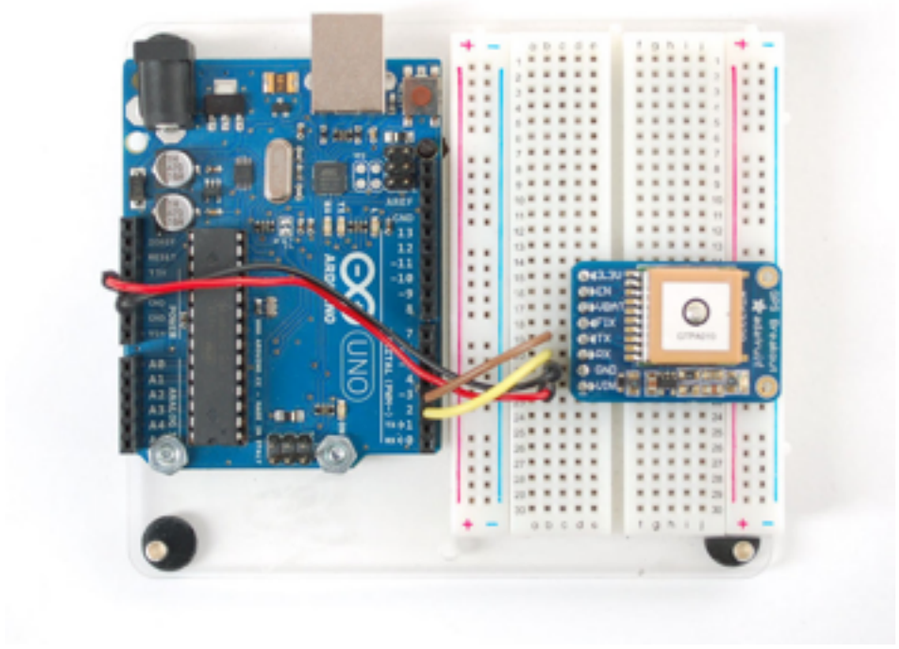


other sensors

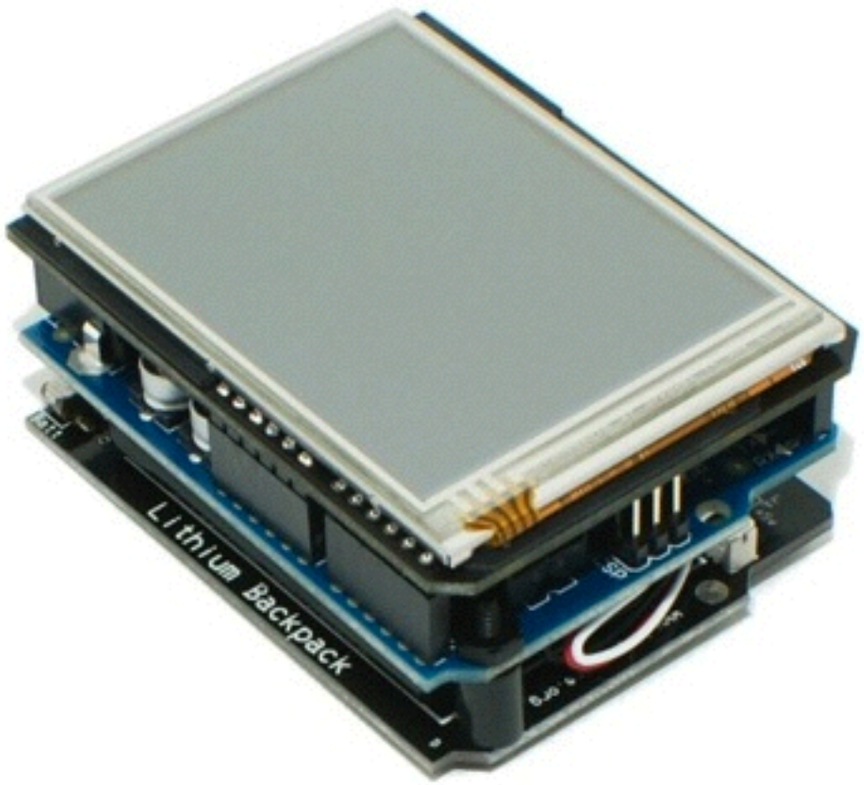
- digital sensors

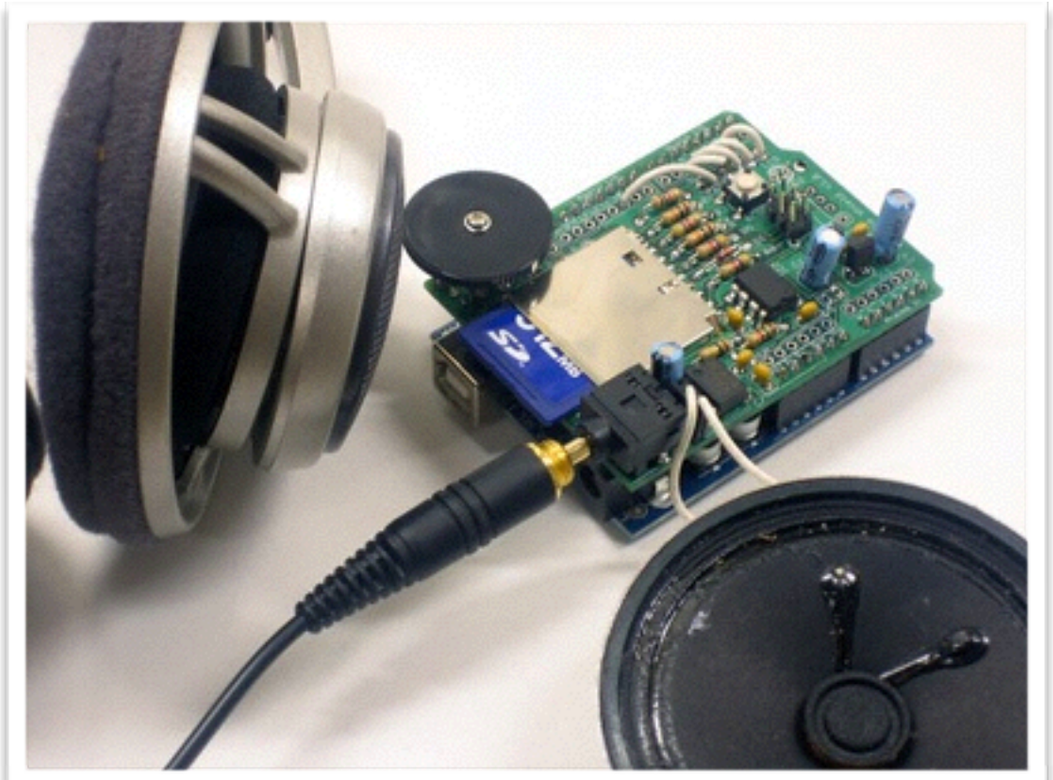
TMP04FS: Serial Digital Output Thermometer



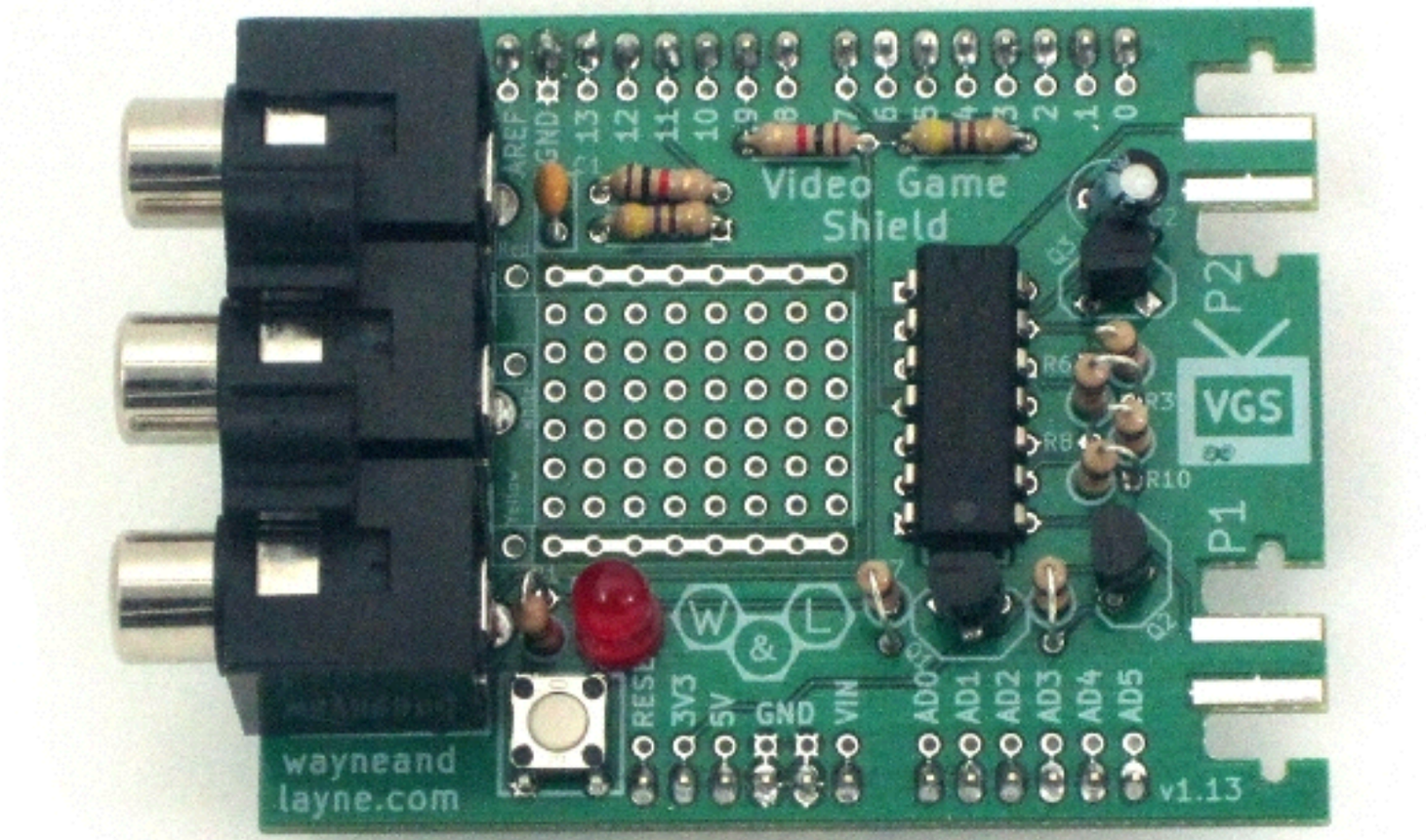


arduino shields



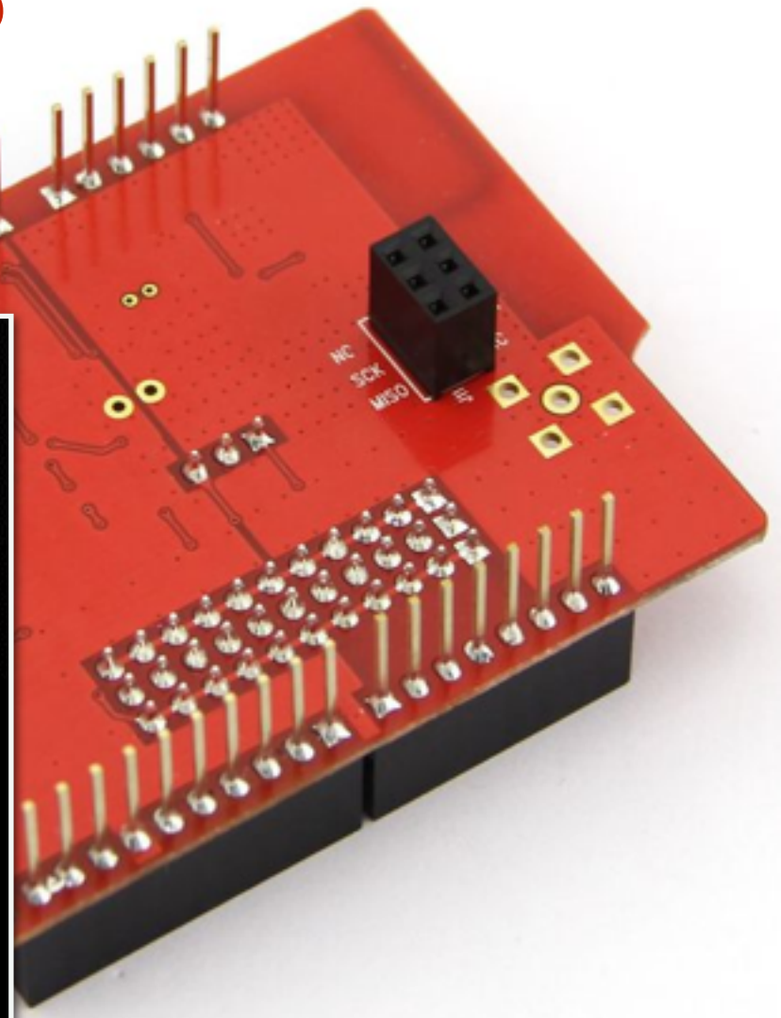


arduino shields



arduino shields

what is ESD?



potentiometer

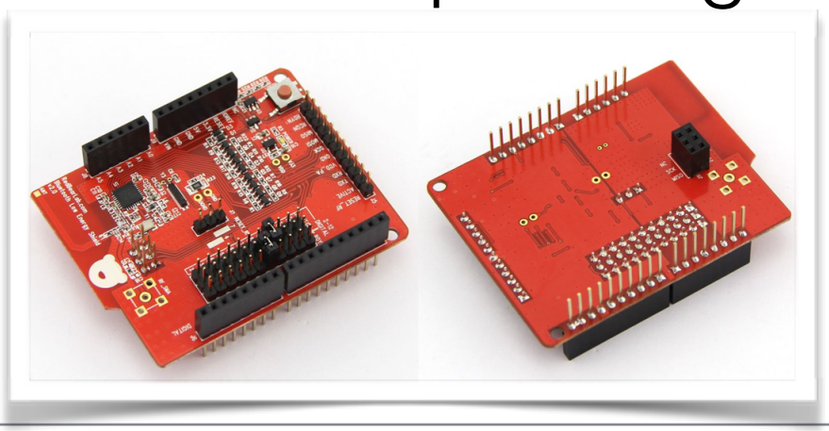
rotational servo

button input

LEDs

BLE shield

- BLE made really easy
- takes care of all the protocol of BLE
- uses pins 8-13 of the arduino (SPI)
  - 10 (SS), 11 (MOSI), 12 (MISO), and 13 (SCK)
  - 8 and 9 are RDYN and REQN for handshaking
  - remaining pins are **A0-A5** and **0-7**
- all communication is setup through `<ble_shield.h>`



BLE shield

setup

```
// Init. and start BLE library.
ble_begin();

// Enable serial debug
Serial.begin(57600);
```

setup SPI &
BLE shield controls

read from BLE, write to serial port

```
while ( ble_available() )
  Serial.write(ble_read());
```

if data available on SPI

read the byte

Serial is the USB port, not SPI

BLE shield

send text that we read from the serial port

```
while ( Serial.available() )
{
  unsigned char c = Serial.read();
  if (c != 0x0A)
  {
    if (len < 16)
      buf[len++] = c;
  }
  else
  {
    buf[len++] = 0x0A;

    for (int i = 0; i < len; i++)
      ble_write(buf[i]);
    len = 0;
  }
}

ble_do_events();
```

read from debug computer

pack in up to 16 bytes

add to transmit buffer

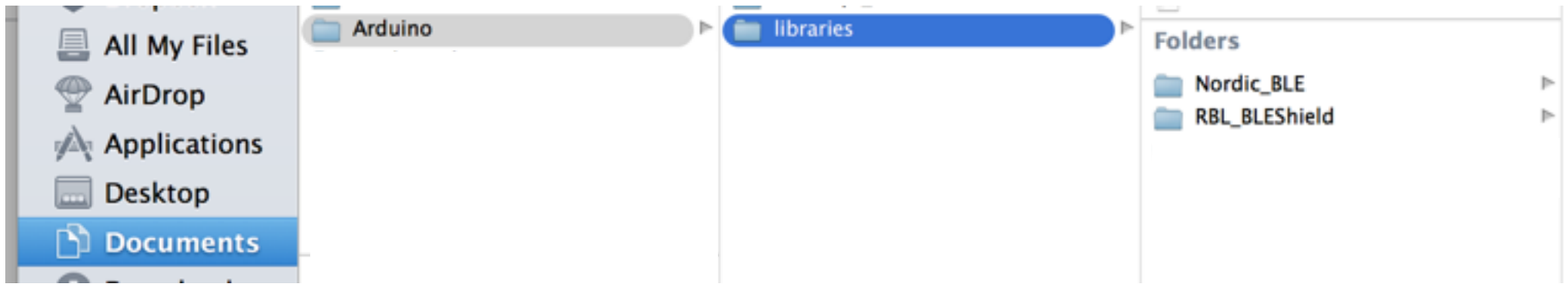
perform transmits
and acks

BLE firmata

- enables custom firmware
  - set pins' output (analog or digital)
  - read input (analog or digital)
  - and essentially tell the arduino to do anything
- an adopted protocol
  - the BLE shield has already implemented this
  - the code can perform almost any operation that includes
    - reads, writes, ADC sampling, PWM, timers, serial comm, dedicated servo control
- http://redbearlab.com/bleshield/

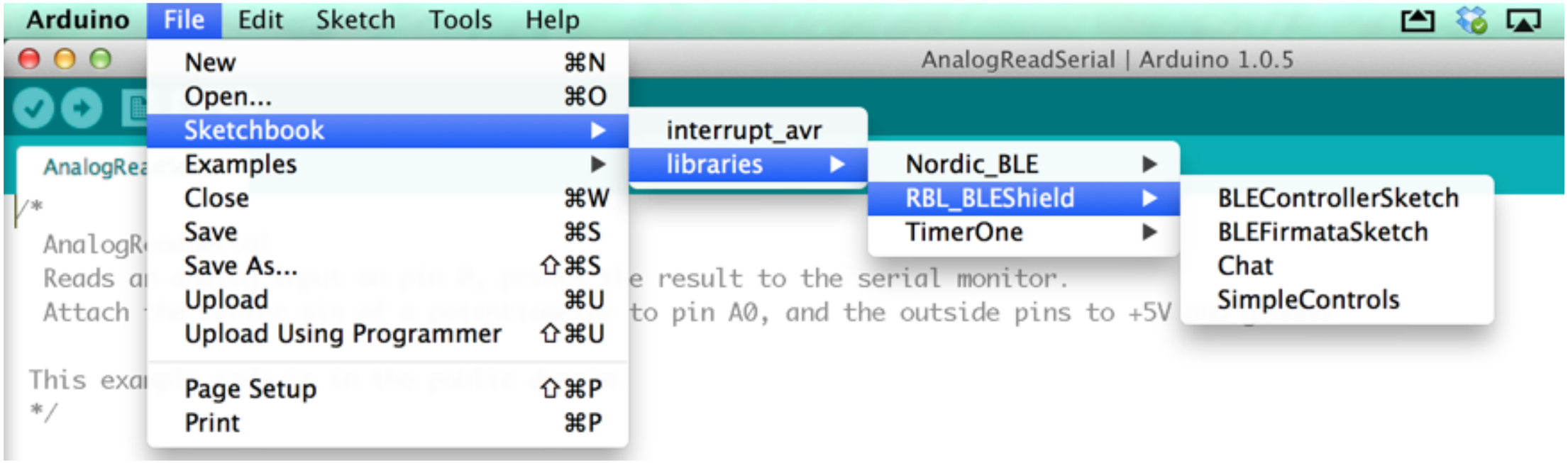
installing library

- download from red bear's github
- https://github.com/RedBearLab/BLEShield/tree/master/Arduino/libraries
- this is a beta SDK
  - but works just fine with iOS7
- place folders into Arduino library
  - ~/Documents/Arduino/libraries
- restart the Arduino sketch program



BLE library

- plenty of examples and firmata included



bluetooth primer

- traditional bluetooth
  - short range wireless
  - low-medium latency
  - good for data like voice and audio
  - fairly power hungry
    - connection oriented
    - connection is maintained even when no data
- one million symbols per second
- but… cannot run from **coin cell battery** and limited to **seven** concurrent connections

bluetooth low-energy BLE

- designed for the internet-of-things
- asynchronous client / server model
- low latency
  - ~3 ms start to finish
- optimized for short bursts of information
  - so not really audio, but that gets abused
- number of connections
  - greater than two billion

theoretically

bluetooth low-energy BLE

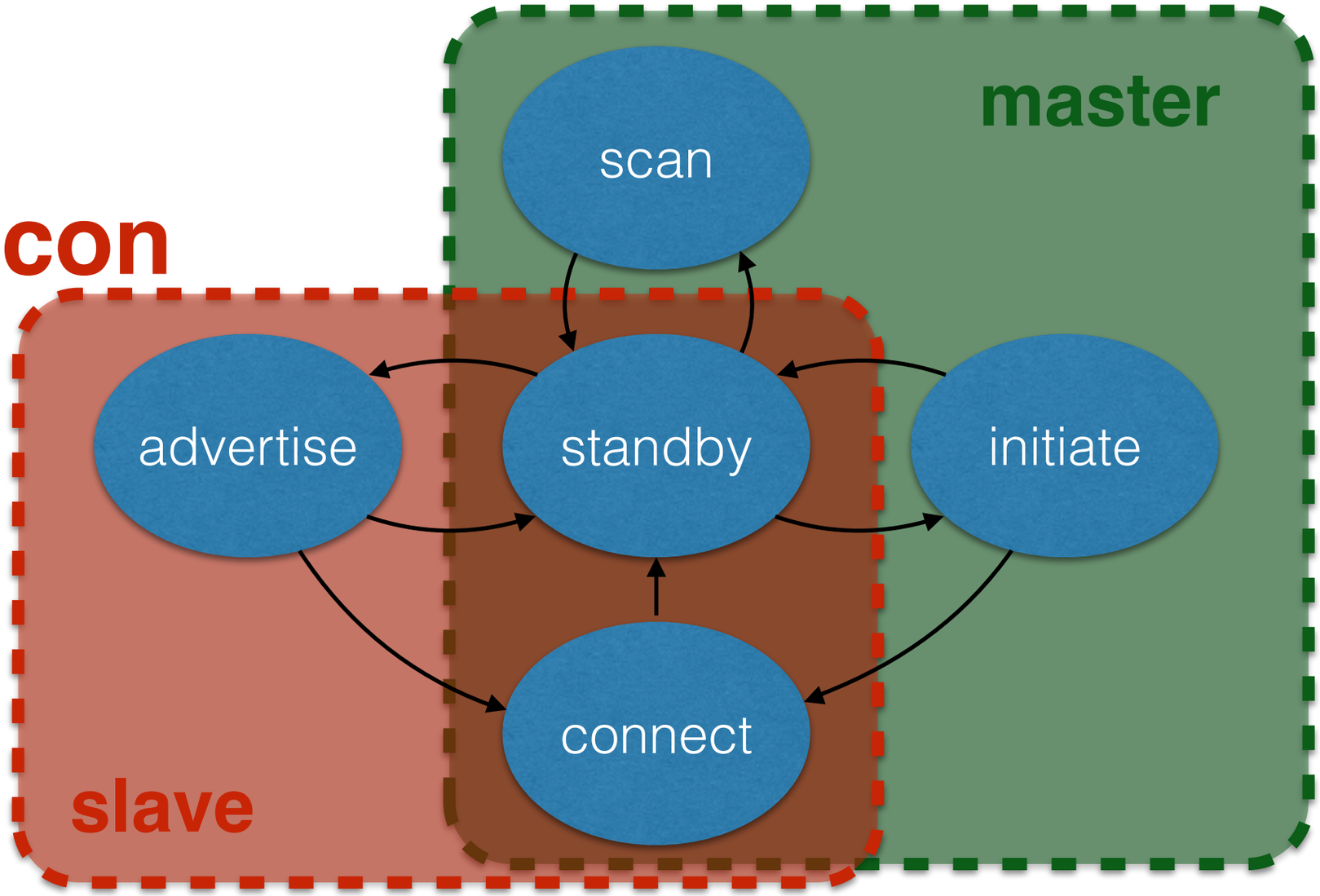
- data transfer can be triggered by events
- read at any time by a client
- interface protocol is Generic Attribute Protocol (GATT)
- client, server
- characteristic (data)
- service (collection of characteristics)
  - model name and serial number are part of GATT
- descriptors
  - more information about a characteristic

GATT protocol

- everything has a universally unique ID (UUID, 128 bit)
- operations:
  - discover UUIDs
  - find service with UUID (or secondary service)
  - discover characteristics for a service
  - find characteristics for a given UUID
  - and get descriptors for characteristic

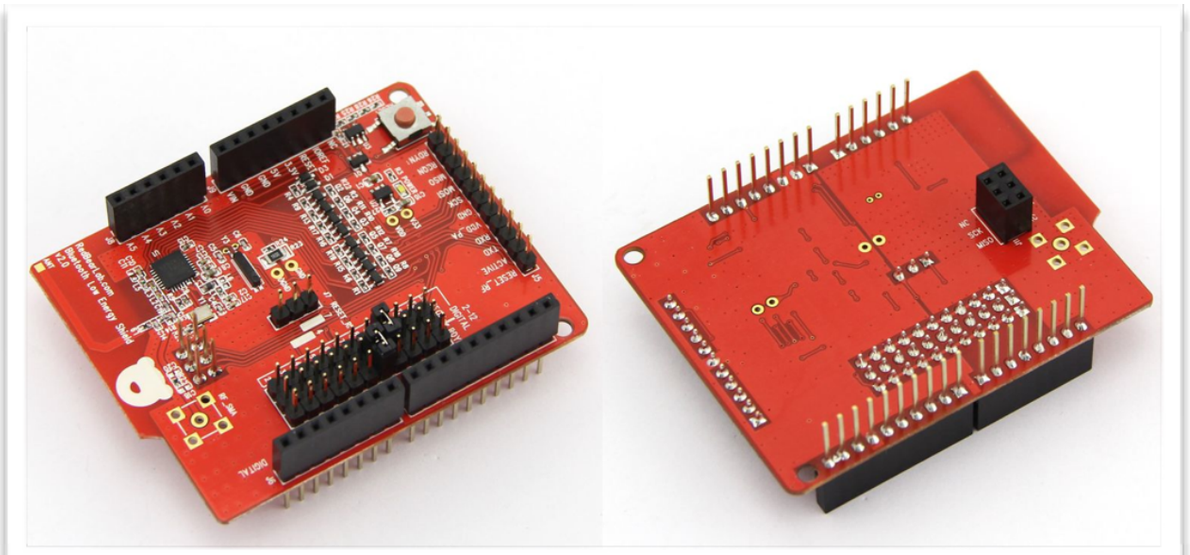
BLE connections

- 3 bands for advertising
- 37 bands for data transmission



who's who?

peripheral



slave

advertise

central

master

initiate

BLE in iOS

- we will use the red bear API
- instantiate BLE object
- connect to advertisers
- use delegation for responding to incoming data
- add as framework to project and that's it
- download from their github page

public API (.h)

```
@protocol BLEDelegate
@optional
-(void) bleDidConnect;
-(void) bleDidDisconnect;
-(void) bleDidUpdateRSSI:(NSNumber *) rssi;
-(void) bleDidReceiveData:(unsigned char *) data length:(int) length;
@required
@end

@property (strong, nonatomic) NSMutableArray *peripherals;
@property (strong, nonatomic) CBCentralManager *CM;
@property (strong, nonatomic) CBPeripheral *activePeripheral;

-(void) enableReadNotification:(CBPeripheral *)p;
-(void) read;
-(void) writeValue:(CBUUID *)serviceUUID
        characteristicUUID:(CBUUID *)characteristicUUID
        p:(CBPeripheral *)p data:(NSData *)data;

-(BOOL) isConnected;
-(void) write:(NSData *)d;
-(void) readRSSI;

-(void) controlSetup;
-(int) findBLEPeripherals:(int) timeout;
-(void) connectPeripheral:(CBPeripheral *)peripheral;
```

peripheral initiation

```
@interface ViewController : UIViewController <BLEDelegate> {
    BLE *bleShield;
}

//start search for peripherals with a timeout of 3 seconds
 // this is an asynchronous call and will return before search is complete
 [bleShield findBLEPeripherals:3];

 // after three seconds, try to connect to first peripheral
 [NSTimer scheduledTimerWithTimeInterval:(float)3.0
                                  target:self
                                selector:@selector(connectionTimer:)
                                userInfo:nil
                                 repeats:NO];

// Called when scan period is over to connect to the first found peripheral
-(void) connectionTimer:(NSTimer *)timer
{
    if(bleShield.peripherals.count > 0)
    {
        // connect the first found peripheral
        [bleShield connectPeripheral:[bleShield.peripherals objectAtIndex:0]];
    }
}

-(void) bleDidConnect
{
    // Schedule to read RSSI every 1 sec.
    rssiTimer = [NSTimer scheduledTimerWithTimeInterval:(float)1.0 target:self
                                selector:@selector(readRSSITimer:) userInfo:nil repeats:YES];
}
```

is this the proper way to connect?

async reading

```
-(void) bleDidReceiveData:(unsigned char *)data length:(int)length
{
    NSData *d = [NSData dataWithBytes:data length:length];
    NSString *s = [[NSString alloc] initWithData:d encoding:NSUTF8StringEncoding];
    self.label.text = s;
}
```

this won't cut it for A5!

you need to send data of different types — how?

SOP: define your own protocol

write data

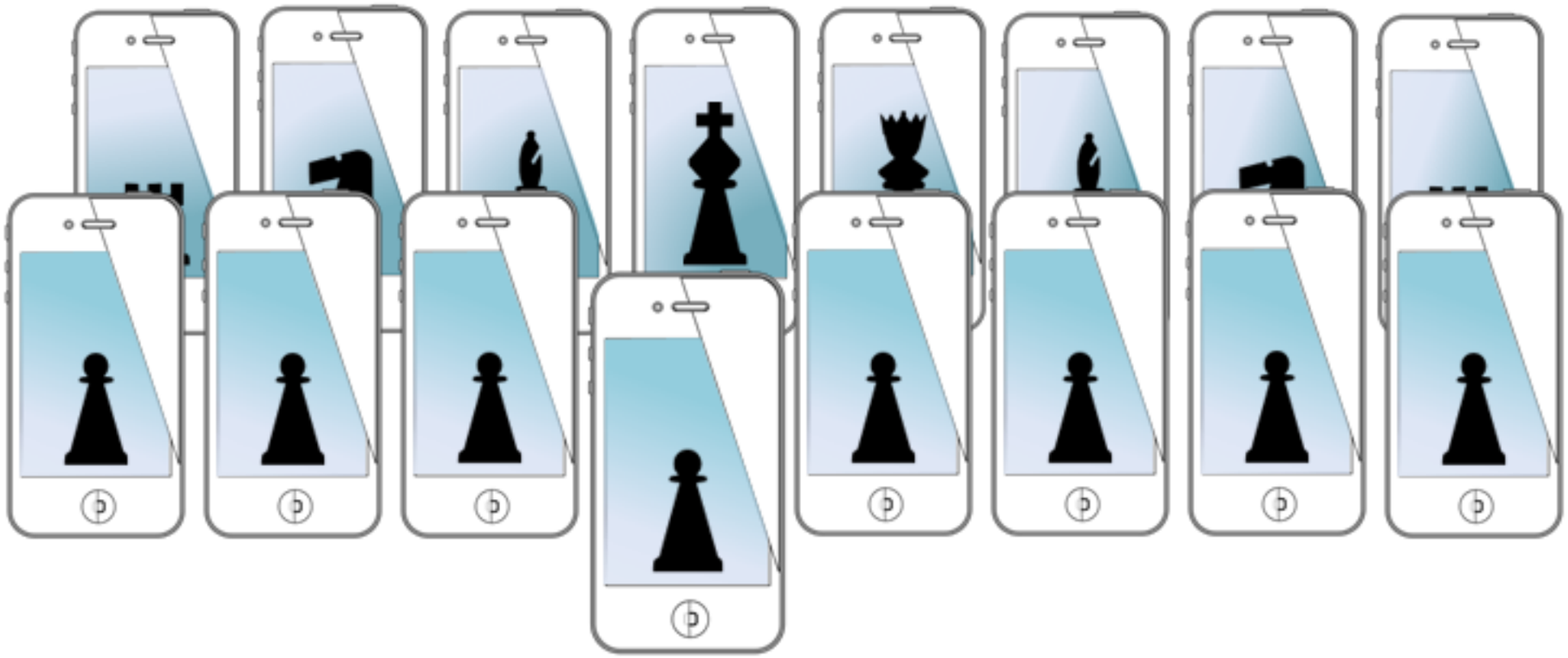
```
NSString *s;
NSData *d;

if (self.textField.text.length > 16)
    s = [self.textField.text substringToIndex:16];
else
    s = self.textField.text;

s = [NSString stringWithFormat:@"%@\r\n", s];
d = [s dataUsingEncoding:NSUTF8StringEncoding];

[bleShield write:d];
```

SOP: define your own protocol

for next time…

- python crashcourse
- preparation for:
  - numpy
  - tornado
  - pymongo
  - scikit-learn

MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture sixteen: control and bluetooth

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University