Goal

- Build a desktop fan from parts in the Arduino Inventor's Kit
- Work in teams of two
- Learn new skills
  - Controlling a servo and DC motor
  - Make a 2D drawing with Solidworks
  - Send drawings to Laser cutter
  - Soldering
- Due in two weeks
  - In-class demonstration of your working fan

Tasks

- Measure servo and DC motors
  - prepare for structural design
  - learn how to use your calipers
- Sketch design of support structure on paper
- Create Solidworks model of the base and DC motor support
- Cut acrylic parts
- Re-solder the DC motor leads
- Assemble the system
- Write Arduino program to control servo and DC motor
Propellor and motors from Inventor’s Kit

Acrylic parts after cutting and bending

One idea for a base design
Alternative base designs

Servo motor parts

Fan Project: First Steps

1. Make a hand sketch of the structural parts
2. Measure the servo and mounting screws
3. Use measurements to add dimensions to the sketch
4. Redraw the sketch as a 2D “flat” drawing in Solidworks
5. Email the drawing to the instructor
   a. Laser cutter works on thin sheets in 2D
   b. Use the acrylic bender after parts are cut

Watch this video to see the laser cutter and acrylic bender in action:

http://www.youtube.com/watch?v=DJ8EmBUfLo
Soldering Leads to the DC Motor

Desktop fan project
EAS 199A, Fall 2010

Overview

The DC motor that comes with the Arduino Inventor’s Kit has short and delicate leads. We need to replace the leads with more robust wiring and soldered connections.

Temperature-controlled soldering iron and flux
Procedure

1. Cut a length of wire
2. Strip and tin the ends of the wire
3. Make note of polarity and remove leads from DC motor
4. Insert tinned wire through tabs and bend into position
5. Secure new leads by soldering to motor tabs
Cut new lead wires

Strip the leads

Tin the leads
Remove the old leads and note the polarity

Motor supported. Extension wires in place

Bend the tinned wires around the supports
Secure the leads with solder

Basic DC Motor Circuits

Desktop fan project
EAS 199A, Fall 2010

Simplest DC Motor Circuits

Connect the motor to a DC power supply

Switch closed
Current continues after switch is opened

Opening the switch does not immediately stop current in the motor windings.

![Diagram](image)

Inductive behavior of the motor causes current to continue to flow when the switch is opened suddenly.

Charge builds up on what was the negative terminal of the motor.

Reverse current

Charge build-up can cause damage

![Diagram](image)

Current surge through the voltage supply

Arc across the switch

Motor Model

Simple model of a DC motor:

- Windings have inductance and resistance
- Inductor causes a storage of electrical charge in the windings
- We need to provide a way to safely dissipate the charge stored in the motor windings
Flyback Diode

A flyback diode allows the stored charge to dissipate safely.

The flyback diode allows charge to dissipate without arcing across the switch, or without flowing back to ground through the +5V voltage supply.

Replace the Switch with a Transistor

A transistor allows on/off control to be automated.

Control the DC motor with PWM Output

```cpp
// Function: PWM_output
// PWM output to control a DC motor
int motor_pin = 5;    // must be a PWM digital output
void setup()
{
  pinMode(motor_pin, OUTPUT)
}
void loop()
{
  int motor_speed=200;  // must be >0 and <= 255
  analogWrite(motor_pin, motor_speed);
}```
Overview

Part I
❖ Circuits and code to control the speed of a small DC motor.
❖ Use potentiometer for dynamic user input.
❖ Use PWM output from an Arduino to control a transistor.
❖ Transistor acts as variable voltage switch for the DC motor.

Part II
❖ Consolidate code into reusable functions.
❖ One function maps 10-bit analog input to 8-bit PWM output.
❖ Another function controls the motor speed.
❖ Functions developed here are useful for more complex control tasks, e.g. the desktop fan project.

Part 1: Control motor speed with a pot

Increase complexity gradually. Test at each stage.
1. Use a potentiometer to generate a voltage signal
   a. Read voltage with analog input
   b. Print voltage to serial monitor to verify
2. Convert 10-bit voltage scale to 8-bit PWM scale
   a. Voltage input is in the range 0 to 1023
   b. PWM output needs to be in the range 0 to 255
   c. Print voltage to serial monitor to verify
3. Connect PWM output to DC motor
4. Write a function to linearly scale the data
5. Write a function to update the motor
Potentiometer Circuit

Use the potentiometer from the Arduino Inventor’s Kit

![Potentiometer Circuit Diagram]

Code to print potentiometer reading

```cpp
// Function:  read_potentiometer
// Read a potentiometer and print the reading
int sensor_pin = 3;  // Wire sweep of pot to
// analog input pin 3
void setup()
{
    Serial.begin(9600);
}
void loop()
{
    int val;
    val = analogRead( sensor_pin );
    Serial.print("reading = ");
    Serial.println( val );
}
```

DC Motor Control Circuit

![DC Motor Control Circuit Diagram]

Add this to the breadboard with the potentiometer circuit
DC Motor Control Circuit

Arduino

Control the DC motor with PWM Output

```c
// Function: DC_motor_control_pot
// // Use a potentiometer to control a DC motor
int sensor_pin = 3;
int motor_pin = 5; // must be a PWM digital output
void setup()
    { Serial.begin(9600);
      pinMode(motor_pin, OUTPUT); }
void loop()
    { int pot_val, motor_speed;
      pot_val = analogRead( sensor_pin );
      motor_speed = pot_val*255.0/1024.0; // Include decimal
      analogWrite( motor_pin, motor_speed); }
```

Subtle: Don’t use integer values of 255 and 1024 here. Aggressive compilers pre-compute the integer division of 255/1024 as zero.

Part II: Create functions for reusable code

```c
// Function: DC_motor_control_pot
// // Use a potentiometer to control a DC motor
int sensor_pin = 3;
int motor_pin = 5; // must be a PWM digital output
void setup()
    { Serial.begin(9600);
      pinMode(motor_pin, OUTPUT); }
void loop()
    { int pot_val, motor_speed;
      pot_val = analogRead( sensor_pin );
      motor_speed = pot_val*255.0/1024.0; // Include decimal
      analogWrite( motor_pin, motor_speed); }
```

Adjust motor speed

Map input values to output scale
Final version of the loop() function

```c
// Function: DC_motor_control_pot
// Use a potentiometer to control a DC motor
int sensor_pin = 3;
int motor_pin = 5; // must be a PWM digital output
void setup()
{
  Serial.begin(9600);
  pinMode(motor_pin, OUTPUT);
}
void loop()
{
  adjust_motor_speed(sensor_pin, motor_pin);
  ... // do other useful stuff
}
```

adjust_motor_speed takes care of the two main tasks: reading the potentiometer outputs and setting the PWM signal to the transistor

Using and Writing Functions

Additional information on the Arduino web site

Functions are reusable code modules:
❖ Functions encapsulate tasks into larger building blocks
❖ Functions hide details and variables local to each task
❖ Well-written functions can be reused
❖ Functions can accept input (or not) and return output (or not)
❖ All Arduino sketches have at least two functions
  ‣ setup: runs once to configure the system
  ‣ loop: runs repeatedly after start-up is complete
❖ Users can add functions in the main sketch file, or in separate files

The setup() Function

Consider the simple blink sketch

```
// Blink.pde: Turn on an LED for one second, then off for one second. Repeat continuously.
void setup() {
  pinMode(13, OUTPUT);
}
void loop() {
  digitalWrite(13, HIGH); // set the LED on
  delay(1000); // wait for a second
digitalWrite(13, LOW); // set the LED off
delay(1000); // wait for a second
}
```

“void” means Returns nothing

No inputs

“setup” is the name of the function
A Function to Translate Linear Scales

Linear scaling from $x$ values to $y$ values:

$$ y = f(x) $$

where $f$ is a linear mapping

$$ \frac{y - y_{min}}{y_{max} - y_{min}} = \frac{x - x_{min}}{x_{max} - x_{min}} $$

$$ \implies \quad y = y_{min} + \left( \frac{y_{max} - y_{min}}{x_{max} - x_{min}} \right) \frac{x - x_{min}}{x_{max} - x_{min}} $$

In words: Given $x$, $x_{min}$, $x_{max}$, $y_{min}$, and $y_{max}$, compute $y$

---

Enter the code at the bottom into your sketch

* The code is not inside any other program block (like setup or void)

How would you test that this function is working?

```cpp
int int_scale(int x, int xmin, int xmax, int ymin, int ymax)
{
    int y;
    y = ymin + float(ymax - ymin)*float(x - xmin)/float(xmax - xmin);
    return(y);
}
```

N.B. This code is essentially a reimplementation of the built-in map function.
See http://arduino.cc/en/Reference/Map

---

A Function to Translate Linear Scales

returns an int

name is int_scale

first input is an int named "x"

int int_scale(int x, int xmin, int xmax, int ymin, int ymax)
{
    int y;
    y = ymin + float(ymax - ymin)*float(x - xmin)/float(xmax - xmin);
    return(y);
}

Use float for better precision

return the value stored in "y"
Functions are not nested

```cpp
int int_scale(int x, int xmin, int xmax, int ymin, int ymax)
{
    ...  
    return( y );
}
```

```cpp
void setup()
{
    ...
}
void loop()
{
    ...
}
```

```cpp
int int_scale(int x, int xmin, int xmax, int ymin, int ymax)
{
    ...  
    return( y );
}
```

// Contents of sketch, e.g. motor_control.pde

Functions call other functions

```cpp
int int_scale(int x, int xmin, int xmax, int ymin, int ymax)
{
    ...  
    return( y );
}
```

```cpp
void setup()
{
    ...
}
void loop()
{
    ...
    motor_speed = int_scale( pot_val, 0, 1024, 0, 255);
}
```

```cpp
int int_scale(int x, int xmin, int xmax, int ymin, int ymax)
{
    ...  
    return( y );
}
```

// Contents of sketch, e.g. motor_control.pde

Functions call other functions

```cpp
int int_scale(int x, int xmin, int xmax, int ymin, int ymax)
{
    ...  
    return( y );
}
```

```cpp
void setup()
{
    ...
}
void loop()
{
    ...
    motor_speed = int_scale( pot_val, 0, 1024, 0, 255);
}
```

// Contents of sketch, e.g. motor_control.pde
Use the int_scale function

```c
int sensor_pin = 3;    // must be a PWM digital output
int motor_pin = 5;

void setup() {
  Serial.begin(9600);
  pinMode(motor_pin, OUTPUT);
}

void loop() {
  int pot_val, motor_speed;
  pot_val = analogRead(sensor_pin);
  motor_speed = int_scale(pot_val, 0, 1024, 0, 255);
  analogWrite(motor_pin, motor_speed);
}

int int_scale(int x, int xmin, int xmax, int ymin, int ymax) {
    int y;
    y = ymin + float(ymax - ymin)*float( x - xmin )/float(xmax - xmin);
    return(y);
}
```

A Function to update motor speed

**Inputs**
- sensor pin
- motor output pin

**Tasks:**
- Read potentiometer voltage
- Convert voltage from 10 bit to 8 bit scales
- Change motor speed

```c
void adjust_motor_speed(int sensor_pin, int motor_pin) {
    int motor_speed, sensor_value;
    sensor_value = analogRead(sensor_pin);
    motor_speed = int_scale(sensor_value, 0, 1024, 0, 255);
    analogWrite(motor_pin, motor_speed);
    Serial.print("Pot input, motor output = ");
    Serial.print(sensor_value);
    Serial.print(" ");  Serial.println(motor_speed);
}
```

Functions call functions, call functions, ...

```c
// Contents of sketch, e.g. motor_control.pde

void setup() {
  ...
}

void loop() {
  ...
  adjust_motor_speed( ..., ... )
  ...
}
void adjust_motor_speed(int sensor_pin, int motor_pin) {
  ...
  motor_speed = int_scale( ..., ..., ..., ..., ... );
  int int_scale(int x, int xmin, int xmax, int ymin, int ymax) {
      ...
      return( y );
  }
```
Button Input: On/off state change
Desktop fan project
EAS 199A, Fall 2010

User input features of the fan

• Potentiometer for speed control
  ❖ Continually variable input makes sense for speed control
  ❖ Previously discussed
• Start/stop
  ❖ Could use a conventional power switch
  ❖ Push button (momentary) switch
• Lock or limit rotation angle
  ❖ Button click to hold/release fan in one position
  ❖ Potentiometer to set range limit

Conventional on/off switch
Basic light switch or rocker switch
❖ Makes or breaks connection to power
❖ Switch stays in position: On or Off
❖ Toggle position indicates the state
❖ NOT in the Arduino Inventors Kit

Image from sparkfun.com
Image from lowes.com
Momentary or push-button switches

- Temporary “click” input
  - Two types: normally closed or normally open
- Normally open
  - Electrical contact is made when button is pressed
- Normally closed
  - Electrical contact is broken when button is pressed
- Internal spring returns button to its un-pressed state

Momentary Button and LED Circuit

Digital input with a pull-down resistor

- When switch is open (button not pressed):
  - Digital input pin is tied to ground
  - No current flows, so there is no voltage difference from input pin to ground
  - Reading on digital input is LOW
- When switch is closed (button is pressed):
  - Current flows from 5V to ground, causing LED to light up.
  - The 330Ω resistor limits the current draw by the input pin.
  - The 10k resistor causes a large voltage drop between 5V and ground, which causes the digital input pin to be closer to 5V
  - Reading on digital input is HIGH

Programs for the LED/Button Circuit

1. Continuous monitor of button state
   - Program is completely occupied by monitoring the button
   - Used as a demonstration — not practically useful
2. Wait for button input
   - Blocks execution while waiting
   - May be useful as a start button
3. Interrupt Handler
   - Most versatile
   - Does not block execution
   - Interrupt is used to change a flag that indicates state
   - Regular code in loop function checks the state of the flag

All three programs use the same electrical circuit
Continuous monitor of button state

```cpp
int button_pin = 4; // pin used to read the button

void setup() {
  pinMode( button_pin, INPUT);
  Serial.begin(9600); // Button state is sent to host
}

void loop() {
  int button;
  button = digitalRead( button_pin);
  if ( button == HIGH ) {
    Serial.println("on");
  } else {
    Serial.println("off");
  }
}
```

Serial monitor shows a continuous stream of "on" or "off"

This program does not control the LED

---

Wait for button input

```cpp
int button_pin = 4; // pin used to read the button

void setup() {
  int start_click = LOW; // Initial state: no click yet
  pinMode( button_pin, INPUT);
  Serial.begin(9600);
  while ( !start_click ) {
    start_click = digitalRead( button_pin);
    Serial.println("Waiting for button press");
  }
}

void loop() {
  int button;
  button = digitalRead( button_pin);
  if ( button == HIGH ) {
    Serial.println("on");
  } else {
    Serial.println("off");
  }
}
```

Same loop() function as before

---

Interrupt handler for button input

```cpp
int button_interrupt = 0; // Interrupt 0 is on pin 2 !!
it toggle_on = false; // Button click switches state

void setup() {
  Serial.begin(9600);
  attachInterrupt( button_interrupt, handle_click, RISING); // Register handler
}

void loop() {
  if ( toggle_on ) {
    Serial.println("on");
  } else {
    Serial.println("off");
  }
}

void handle_click() {
  static unsigned long last_interrupt_time = 0; // Zero only at start
  unsigned long interrupt_time = millis(); // Read the clock
  if ( interrupt_time - last_interrupt_time > 200 ) { // Ignore when < 200 msec
    toggle_on = !toggle_on;
  }
  last_interrupt_time = interrupt_time;
}
```

---
Interrupt handler for button input

```c
int button_interrupt = 0; // Interrupt 0 is on pin 2!!
int toggle_on = false;    // Button click switches state

void setup() {
    Serial.begin(9600);
    attachInterrupt( button_interrupt, handle_click, RISING);  // Register handler
}

void loop() {
    if ( toggle_on ) {
        Serial.println("on");
    } else {
        Serial.println("off");
    }
}

void handle_click()
{
    static unsigned long last_interrupt_time = 0;       // Zero only at start
    unsigned long interrupt_time = millis();             // Read the clock
    if ( interrupt_time - last_interrupt_time > 200 ) {  // Ignore when < 200 msec
        toggle_on = !toggle_on;
    }
    last_interrupt_time = interrupt_time;
}
```

- **Button Interrupt:** The button interrupt must be registered when the program starts.
- **Button ID:** `button_interrupt` is the ID or number of the interrupt. It must be 0 or 1.
- ** Interrupt Handler:** A RISING interrupt occurs when the pin changes from LOW to HIGH.
- **Toggle Variable:** `toggle_on` is a global variable that remembers the "state". It is either true or false (1 or 0).
- **Loop Function:** The `loop()` function only checks the state of `toggle_on`. The value of `toggle_on` is set in the interrupt handler, `handle_click`.
- **Value Retention:** The value of `toggle_on` is flipped only when a true interrupt even occurs. Debouncing is described in the next slide.
Other references

Ladyada tutorial
- Excellent and detailed
- http://www.ladyada.net/learn/arduino/lesson5.html

Arduino reference
- Minimal explanation
- Using interrupts