3D MODELING WORKSHOP

OVERVIEW

During this workshop you will learn how to use Tinkercad to create & customize 3D models.

Tinkercad is an easy-to-use tool for creating digital designs that are ready to be 3D printed into physical objects.

And at the end you will have a task to make an object that you can print later on one of our 3D printers

PREPARATIONS

- We will be using browser based software so you don't have to install anything!
- All you need to do is to go to https://tinkercad.com/ & register for a free account.
- Later on we will also be using Google Docs drawing tool (https://docs.google.com/drawings), so if you don't have a Google account this is a good time to go and get one.

MAIN FLOW

1. LEARN THE LAYOUT

Log in to your Tinkercad account, click on "Create new design" button & let's get started!

Take some time to know your way around Tinkercads layout. Basically all the tools you will need are located in the right menu bar.

2. CHECK YOUR GRID

Near the bottom right corner of the workplane you have *"Edit grid"* options & *"Snap grid"* value.

Leave the value as it is, but we need to adjust the grid size to those of the 3D printer bed. It will be easier for you to understand the dimensions you are working in.

3. LEARN THE BASICS.

Drag the red Box to the workplane (close to middle is usually best).

You can drag the box to any place on the workplane. Click & drag to move it in x & y axis, pull the black arrowhead on the top plane to move it in the "z" axis.

Click on the Box and play with the visible controls.

Black dots on the base of the box allow you to scale the box in "y" or "x" axis, and the white dot on the top plane lets you scale it in the "z" axis.

You can also scale the box freely in x&y axis while pulling one of four white dots on the base plane. Use uniform

scaling of the objects just by holding shift while pulling any of the control dots.

Notice that you can also rotate the objects in any direction. Holding Shift button while doing so will snap the rotation every 45 degrees.

4. WORKING WITH DIMENSIONS

First of all delete the object you were working on before.

Ok, now get *Cylinder* on the plane. Now from the right menu bar choose the *Ruler* and put it on the workplane.

Notice that when you click on the cylinder now, when the ruler is also on the workplane, you can see the dimensions of the object. You can click on them to enter values.

Change the dimensions to:

- height: 15mm
- diameter: 25mm.

Apart from the object dimensions, you can also change the distance from the workplane & distance from the ruler axis.

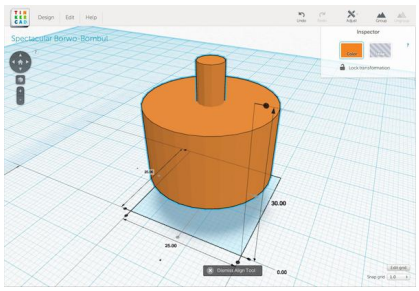
5. 3D MODELLING: THE BASICS

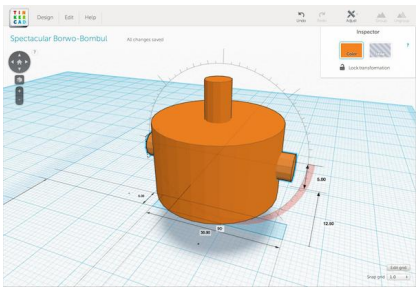
Ok, so we have a cylinder with the dimensions set in the previous step. Now lets get one more cylinder to the workplane.

1. Set the cylinder dimensions to

- height: 30mm
- diameter: 5mm.

2. Select both objects either by dragging a window over them, or by shift clicking both of them. Go to *"Adjust"* in the top bar on the right, and choose *"Align"* option.
3. Center align the cylinders in all three dimensions.
4. Copy & paste the thin cylinder and again select all object and center align them in all dimensions. Rotate one of the thin cylinders 90 degrees.

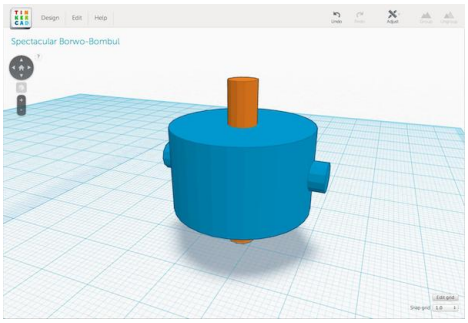


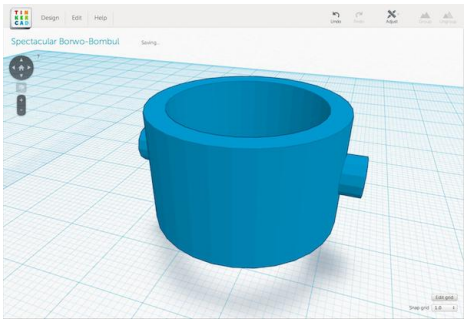


5. Select the horizontal thin cylinder and the big one and group* them together using either keyboard shortcut or command from the top menu bar.

> Group command in Tinkercad “welds” the objects together, you can unweld them simply by ungrouping them.

6. Change the color of the grouped object.
7. Change the diameter of the thin vertical to 20mm. Realign the objects.
8. Click on the thin cylinder and set it to “Hole” in the inspector.
9. Group all objects together and congrats you have your first 3D model ready!





6. USE THE BASIC MODELING TECHNIQUES YOU LEARNED TO CREATE A GAMING DICE

Use the predefined dice model from the right menu bar & the numbers also found there. Feel free to experiment.

ADDITIONAL TASKS

Do you feel good with your skills? If so let’s get to the next step if not keep on experimenting or ask us for some help.

UTILIZE YOUR 3D MODELLING SKILLS TO MAKE SOMETHING USEFUL

We dare you to design and model a bottle opener and since the material we will be printing with is not so strong, we will be using a coin in the design, so it will be more sturdy and open the bottle instead of breaking ;)

1.1. USING GOOGLE DOCS DRAWING TOOL TO MAKE CUSTOM SHAPES & TEXT.

Go to https://docs.google.com/drawings to start a new drawing.

Here you can create custom shapes with curve tools or use some predefined shapes that are still much more than the ones in Tinkercad or even create custom text!

When you have your shape or text ready, go to *“File” -> “Download as” -> “.svg”* You can now import this file to Tinkercad using the *“Import”* command at the top of the right menu bar.

1.2. TIPS FOR DESIGNING THE BOTTLE OPENER.

You can visit http://www.thingiverse.com/ and search for some bottle openers to have some inspiration of what you can do. If you don't have the patience you can even download the opener you really like and personalize it.

You should know the dimensions of the coin you will be using in your design - you can use the web to find them or use our calliper to measure it.

Your design should not be bigger than x=60mm, y=60mm, z=30mm - printing takes some time and everyone would like their design to be printed :)

ADDITIONAL SOURCES

You want to tinker more in 3D modelling or Tinkercad did not satisfy you?

Here is a list of free 3D modelling software we created for you:

- Autodesk 123D - design: http://apps.123dapp.com/design/
- Sketchup - http://www.sketchup.com/ (plugin needed for stl exports)
- Blender - http://www.blender.org/
- Rhino4Mac - http://www.rhino3d.com/mac (pre alpha version)
- OpenSCAD - http://www.openscad.org/

TINKERING WITH ARDUINO

OVERVIEW

During this workshop you will learn the basics of communicating with an Arduino using your favorite scripting language (Python, Ruby, or Javascript).

We'll start with the basics like blinking an LED and reading data from a button, and by the end of the workshop we'll work our way up to a fully functioning light switch that you can control from a simple Web Application.

This workshop is open to hackers of all skill levels. Even if you've never touched an Arduino before, we'll help you get from zero to "Hello World" in no time flat. Preparations To complete this workshop, you only really need two things:

1. A COMPUTER WITH THE ARDUINO IDE AND DRIVERS INSTALLED ON IT.

You can download the latest version of the Arduino IDE (Version 1.0.5) and drivers from the official Arduino website. There are Mac, Windows, and Linux binaries available.

http://arduino.cc/en/Main/Software#toc1

2. A WORKING PYTHON, NODE.JS, OR RUBY ENVIRONMENT.

We're going to be using a serial connection to communicate with the Arduino using one of your favorite languages. In order to do this, you need to have a working environment setup on your computer.

Python If you're going to be using python, make sure you have Python 2.7 and PIP installed. The library we'll be using, BreakfastSerial does have Windows support. You can download a copy of python on the official website (http://www.python.org/getit/).

Once you have Python installed, make sure you can install the BreakfastSerial library:

```
$ pip install breakfastserial
```

Node.js If you're going to be using Node.js, make sure you have the latest stable version of node installed (Version 10.25 as of 2/14/14). The library we'll be using Johnny-Five does have Windows support. You can download a copy of Node.js from the official website (http://nodejs.org/).

Once you have Node.js installed, make sure you can install the Johnny-Five library:

```
$ npm install johnny-five
```

Ruby If you're going to be using Ruby, make sure you have Ruby 1.9.3 or higher installed. The library we'll be using Artoo does NOT have windows support. If you're using windows, we can set you up with Python or Javascript instead (sorry!). You can download Ruby from the official website (https://www.ruby-lang.org/en/downloads/).

Once you have Ruby 1.9.3 or higher installed, make sure you can install the Artoo gem and the serialport library:

```
$ gem install artoo-arduino
```

```
$ gem install hybridgroup-serialport
```

MAIN FLOW

1. LEARN ABOUT THE LAYOUT OF AN ARDUINO AND THE COMPONENTS YOU HAVE.

Goal: Understand the basic layout of an Arduino board and the components we have.

Arduinos look a lot more complicated than they actually are. Let's learn about the board and how it works.

The board you have in front of you is an Arduino Uno, which is the basic Arduino board. During this workshop, there are three components that you will need to interact with:

- The USB Interface - We talk to our Arduino using a USB serial connection to our computer. This is where we connect them.
- The Ground Ports - You'll see a number of tiny square holes around the outside of the board. Each of them has a label next to it that describes the purpose. The ports labeled "GND" are for Grounding.
- The Digital Ports - On one side of the board, you will see a "Digital" label and ports numbered from 0 to 13. These are the digital pins that all our diagrams will use.

These ports provide power and can be turned on and off.

During this workshop, you will be building a series of circuits. You can think of a circuit like a simple loop. In order for the circuit to work, one end must be connected to a port that supplies power and the other end must lead to group.

2. "HELLO WORLD" IN ARDUINO.

Goal: Blink an LED using the Arduino language and IDE.

First thing first, let's get our Arduino to blink an LED. LEDs have two ends - the long end goes to power and the short end goes to ground. We can insert the long end of an LED into the pin on our board labeled "13" and the short end into the "GND" pin next to it to create a simple circuit. (See Diagram)

Making the LED blink is pretty simple. The Arduino IDE actually has the code pre-written for us so we don't have to even write it ourselves. Open it up by clicking:

```
File > Examples > 01.Basics > Blink
```

The code is pretty well documented, so read it and get a feel for how it works. When you're ready click the "Upload" button and watch your very first LED blink!

3. INTRODUCTION TO FIRMATA.

Goal: Understand what Firmata is an how to upload it to our Arduino.

Firmata is a generic protocol for communicating with a microcontroller using software on a computer over a serial connection. You can think of it like an API for talking to Arduinos. We compile the firmata program and push it to

our board. Then a program can send and receive commands to/from the Arduino.

There are implementations of Firmata in a bunch of languages and flashing it on to our Arduino is super easy using the Arduino IDE. Open up the IDE again and click on:

```
File > Examples > Firmata > StandardFirmata
```

This should open up a new window that contains the code we need to upload to your Arduino so it "speaks" Firmata. Click "Upload" to compile the code and push it to the board.

Hint: You should see the LED flash a few times and then it should stop. ### 4. "Hello World" in your favorite language. Goal: Blink an LED using either Javascript, Python, or Ruby.

Now that we have Firmata running on our Arduino board, we can start using our favorite scripting language to control it. Let's rewrite our "Hello World" example in either Python, Javascript, or Ruby. We already have the LED wired up, so no need to worry about that.

Python To communicate with the Arduino in Python, we'll be using a library called BreakfastSerial. You should already have it installed when you setup your python environment. Using BreakfastSerial is really simple. The library gives us objects that represent the real world components in our circuit. For example, there is an Arduino object which represents an arduino and an Led object which represents an LED. Here's how you import them:

```
from BreakfastSerial import Arduino, Led
```

Now that we have them imported, we can create new instances of the objects and use them to interact with our LED. For our example, we'll need a board object and an LED that lives on pin 13.

```
board = Arduino()
led = Led(board, 13)
```

The led object has methods that we can call to effect the state of the real world LED. To turn it on, you would do the following:

```
led.on()
```

There are also methods for off, toggle, and blink. Make the led blink every second and then move on to the next step.

Node.js To communicate with the Arduino in Node.js, we'll be using a library called Johnny-Five. You should already have it installed when you setup your node environment. Using Johnny-Five is really simple. The library gives us objects that represent the real world components in our circuit. For example, there is a Board object which represents an arduino and an Led object which represents an LED. Here's how you create a board:

```
var five = require("johnny-five")
   , board = new five.Board();
```

Every new Board object emits a "ready" event when the board is connected. It takes a callback function that you can then instantiate new objects within.

```
board.on("ready", function() {
```

```
    led = new five.Led(13);
});
```

The led object has methods that we can call to effect the state of the real world LED. To turn it on, you would do the following:

```
led.on()
```

There are also methods for off, toggle, and blink. Make the led blink every second and then move on to the next step.

Ruby To communicate with the Arduino in Ruby, we'll be using a library called Artoo. You should already have it installed when you setup your ruby environment. Using Artoo is really simple. The library gives us devices that represent the real world components in our circuit. For example, there is a board device which represents an arduino and an led device which represents an LED. In order to connect to the board, you'll need to know what Serial Port it is connected to. You can get this from the Arduino IDE by clicking on Tools > Serial Port and seeing what is checked.

Here's how you create a board and led:

```
require 'artoo'

connection :firmata, :adaptor => :firmata, :port => '/dev/tty.usbmodem1411'
device :board
device :led, :driver => :led, :pin => 13
```

Every artoo program has a "work" loop that you can access objects from. For example we can interact with our LED from the work loop. The led object has methods that we can call to effect the state of the real world LED. To turn it on, you would do the following:

work do led.on end

There are also methods for off and toggle. Make the led blink every second and then move on to the next step.

5. Reading data from a Button. Goal: Build a simple button circuit and print out "Button Pressed" whenever someone pushes the button.

So far we've only seen how to write data to the Arduino, but the Arduino is capable of reading data as well. We do this using sensors like buttons, knobs, temperature, sound, and so on. In your kit you have a button, which you can wire up according to the following diagram.

Python

Just like the Led, we need to import a Button object from BreakfastSerial to be able to use it. You instantiate a new button by passing it a reference to the board and the pin number that the button is connected to. In our diagram, the pin number is "8".

```
from BreakfastSerial import Arduino, Button
board = Arduino()
button = Button(board, 8)
```

Our new button object has methods like "down", "up", and "hold", which we can use to tap into events as the occur.

Each of these methods takes one argument, which is a function. That function gets called when someone presses the button. For example:

```
button.hold(play_some_sound)
```

When ever someone presses and holds the button down, a function called “play_some_sound” is called.

Now write a program that prints out “Button Pressed” whenever someone presses your button.

Hint: If you python program exits immediately after you start it, that’s because there are no other actions on the stack and it’s exiting by default. You can solve this by starting your program with the “-i” flag or manually adding a REPL at the end of your program.

Hint: Stuck? Check out this example: https://github.com/theycallmeswift/BreakfastSerial/tree/sw-update-examples#push-a-button

Node.js Just like the Led, we need to instantiate a new five.Button object within our “ready” loop to be able to use it. Instantiating a new button takes one argument, which is the associate pin number. In our diagram, the pin number is “8”.

```
var button = new five.Button(8);
```

Our new button object has methods like “down”, “up”, and “hold”, which we can use to tap into events as the occur. Each of these methods takes one argument, which is a function. That function gets called when someone presses the button. For example:

button.on(“hold”, function() { // DO SOMETHING HERE });

Now write a program that prints out “Button Pressed” whenever someone presses your button.

Hint: Stuck? Check out this example: https://github.com/rwaldron/johnny-five/blob/master/docs/button.md

Ruby

Just like the Led, we need to tell our program that there is a button device to be able to use it. We need to indicate which pin it lives on to be able to use it. In our diagram, the pin number is “8”.

```
device :button, :driver => :button, :pin => 8
```

Now in our work loop, we can create a new listener for a “push” event on our button. The push event takes in a proc, which is a function that gets called every time the event happens.

```
on button, :push => proc {
    # DO SOMETHING HERE
}
```

Now write a program that prints out “Button Pressed” whenever someone presses your button.

Hint: Does your button seem a bit slow to react to you pushing it? This is a know issue with the Artoo library, just press and hold the button a little longer than you would expect to.

Hint: Stuck? Check out this example: http://artoo.io/documentation/examples/ruby/firmata_button/

6. Connect the dots - Make a lightswitch! Goal: Build a functional light switch. When you press the button, the light should come on. When you press the button again, the light should go off.

Now that we've managed to turn an LED on and off and we can read data from a button, how do we combine the two concepts to create a simple light switch?

Hint: You can reuse a lot of the code we wrote for the previous exercises. Each library has a "toggle" method that we can use to make things easier.

Additional tasks Do you feel good with your skills? If so let's get to the next step if not keep on experimenting or ask us for some help.

1. Build a simple web interface for your Light Switch. Using a simple server, create a web interface for your light switch. You should display the current state of the light (on vs off) and be able to toggle between them from both the web and the button on the Arduino.

If you're looking for a simple server, here are some that you might want to check out:

Flask (Python) - http://flask.pocoo.org/ Express (Node.js) - http://expressjs.com/ Sinatra (Ruby) - http://www.sinatrarb.com/

Bonus: Integrate either SendGrid or Twilio's API so you can control you light via email or SMS message.

2. Explore interacting with some of the other available components Ask one of the workshop leaders for some advanced components and practice working with them in your language of choice. You may have to refer to the documentation for the library you're using to find out what components are available.

BUILD-A-BOT WORKSHOP

OVERVIEW

In this workshop you will learn how to assemble and program an arduino based robot that you can control with your laptop, teach to 'see' by using sensors, or use to battle other bots in the sumo ring.

PREPARATIONS

We will be using a piece of software called **Robotnik** that lets you program and control the Arduino robot. In order to run Robotnik on your laptop, you will first need to install Node.js. Please go to http://nodejs.org/, click *Install* and then follow the instructions.

Once you have installed Node.js, you can install Robotnik using npm. If you are on a Mac or Linux, just open a command propmt. If you are a Windows user, open 'Node.js command prompt' that was installed along with Node.js. Then you can installed Robotnik by typing:

```
npm install -g robotnik
```

Once the install is complete, connect the Arduino to your laptop using the USB cable:

Then you can start the program by typing:

```
robotnik
```

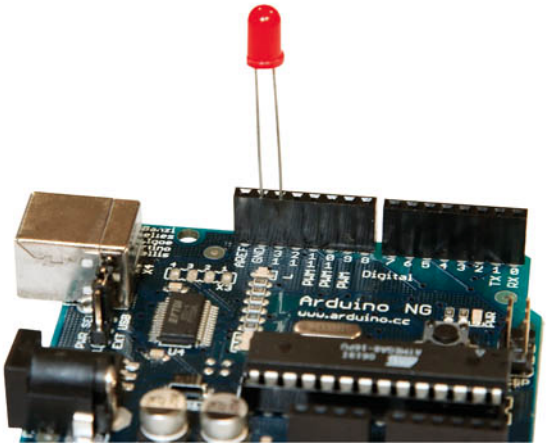
That's it! The program may instruct you to open a web browser to http://localhost:8057/ if it couldn't do so for you automatically. You are now ready to begin the tutorial. Remember, if you run into any trouble, just ask one of the workshop assistants and we'll help you as much as you require!

MAIN FLOW

QUEST 1: MAKE A LED BLINK

Robotnik is a programming system that works like Lego™ or puzzle pieces. On the left are available blocks and on the right is where you drag them to make your program. We're going to make a program that turns a LED on when you press a virtual button in Robotnik.

First, you'll need to plug a LED (any color) into the Arduino. The short leg goes into GND and the long leg goes into 13.



Next, find a block like this:

and drag it on to the right side of the screen. As you can see, the block describes what it will do in English. It's just up to us to finish the sentence now. If you wish, you can change 'red' to 'blue' to change the sentence to talk about a different button. Let's leave it at 'red' for now.

Next, look for the 'turn LED on' block and drag it inside of the first block next to 'do', so that it looks like this:

Make sure the two pieces are connected - you'll briefly see a yellow outline and hear a clicking noise if you did it right.

Now click the *Run Program* button. If you did everything correctly, a virtual joystick will appear on your screen and clicking on the red button will make the LED blink!

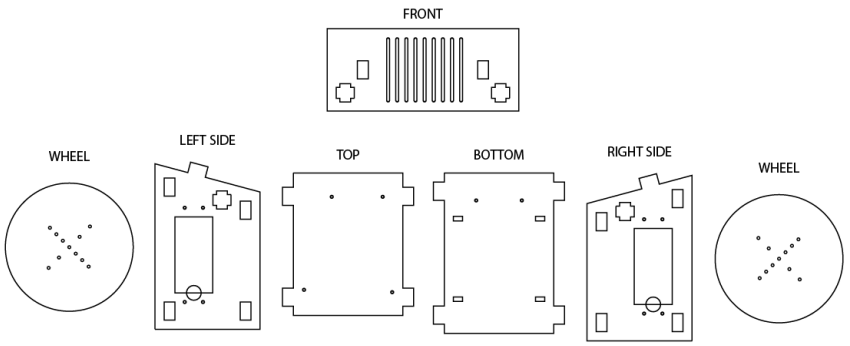
Didn't work? Uh oh. Make sure:

- your USB cable is plugged into your computer and Arduino
- the short leg of the LED is in GND and the long leg is in 13 and it is plugged in
- that Robotnik is still running - look at your terminal window and run `robotnik` again if needed

Don't forget to ask for help if you need it!

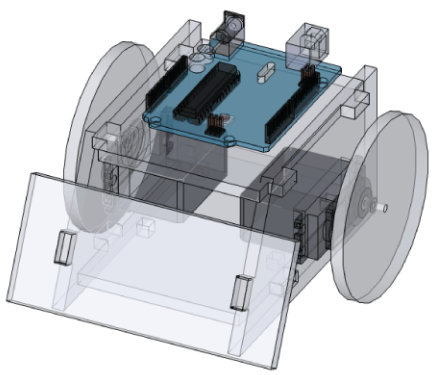
QUEST 2: CONTROLLING A ROBOT

You now know the how to create programs by using Robotnik blocks and how to run programs. We are ready to move on to something more advanced. The first step is to assemble the robot. All of the wooden pieces are labeled and slot together - look at the example robot and ask questions if you need help.



Put all the pieces together, then pop the motors in through the sides and screw them in. Attach the Arduino to the top, plug the shield on to the top of the Arduino and plug the motors and batteries in to the labeled ports of the shield. Attach the wheels to the sides of the motors. Many Arduino shields add additional hardware, but ours is just used to connect everything together in the right way. Under the shield and behind the scenes, the batteries are connected to the motors, and one pin from each motor called the 'control pin' goes to the appropriate numbered pin on the Arduino.

When you're done, your robot should look something like this:



Good job! Our goal is now to make the robot move forward when we press the joystick forward, backward when we move backward and to turn left and right when we move that way. You can do this by assembling the blocks correctly. Remember that each motor moves independently, and when they move in the same direction, the robot will move in that direction. When they move in opposite directions, the robot will twist in place. When only one motor turns, the robot will pivot around the wheel that isn't moving. Remember to think about the way the motors are facing.

Once you have this down, you can move pilot your robot around the obstacle course, or battle someone in the sumobot ring! The first robot to leave the circle loses the match!

Problems?

- Make all the cables are plugged in correctly
- Is Robotnik is still running? Look at your terminal window and run `robotnik` again if needed
- Are there batteries in the battery case and is it turned on?
- Remember that the motors are facing opposite directions
- Are there any 'extra' sticking out of the Arduino shield? The shield and Arduino should align perfectly and there should be no 'extra' pins.

QUEST 3: TEACHING THE ROBOT TO THINK FOR ITSELF

You've made a robot that you can control with your computer. This is one kind of robot, but to really be useful, a robot can be made autonomous so that it senses the world around it with 'sensor modules' and makes decisions automatically based on the data that it gets from them.

We have provided a 'proximity sensor' that tells the robot how far it is from an object. Attach it to the front of the robot, and plug it into the labeled port on the shield. Now the robot should be able to 'see'. Our goal is to tell the robot to:

1. Move forward
2. If there is something 5cm in front of the robot, turn right slightly
3. Repeat

Again, you can do this by dragging the spaces Once you have this program working, you should be able to set your robot down in the obstacle course and let it go without bumping into anything.

Did it work? Congratulations on creating a robot that can move and think for itself! Do your part to prevent the robot apocalypse by always staying one step ahead of the machines.

ADVANCED FLOW

You program? Great! I want you to do the same three quests as in the main flow, but I'm going to give you extra information so you can go on side quests. :)

Robotnik is a Google project that actually generates source code when you place the blocks. You can see the source code by clicking on the 'Code' tab. Behind the scenes, we are generating Javascript code and using Rick Waldron's excellent Johnny-Five library to talk to the Arduino over protocol called Firmata.

You can modify the code in place in the editor and run directly from there instead of using the block interface. This

gives you much more flexibility and you can do all sorts of things! The Johnny-Five documentation is linked in the 'Code' tab. Between looking at the code generated by the blocks, and the documentation, you should be able to complete all the quests and add some of your own special twists.

NOT AT MAKERLAND?

Are you home from the conference and want to do this again? No problem! You missed Makerland but want to follow this tutorial anyway? No problem! Everything we use is open source and readily available. First, you'll need an Arduino Uno R3 with the firmata program uploaded to it. This is a requirement to use Johnny-Five. To do that:

- Download Arduino IDE (http://arduino.cc/en/main/software)
- Plug in your Arduino or Arduino compatible microcontroller via USB
- Open the Arduino IDE, select: *File* > *Examples* > *Firmata* > *StandardFirmata*
- Click the *Upload* button.

Then you'll need the parts for the robot and schematics for the shield. This is all available at http://sumobotkit.com - if you have access to a nice Hackerspace or a laser cutter and 3D printer, you can make everything yourself, otherwise you can find places to order the parts through a service like http://ponoko.com

Finally, here is the shopping list of parts from Adafruit:

- Arduino Uno R3: http://www.adafruit.com/products/50
- Proximity Sensor: http://www.adafruit.com/products/164
- Battery Case: http://www.adafruit.com/products/830
- Continuous Rotation Servo Motor (2): http://www.adafruit.com/products/154

Any color of LED will do, and you'll also need a 'type B' USB cable to connect the Arduino. You can find them at your local electronics store or at Adafruit if you search.

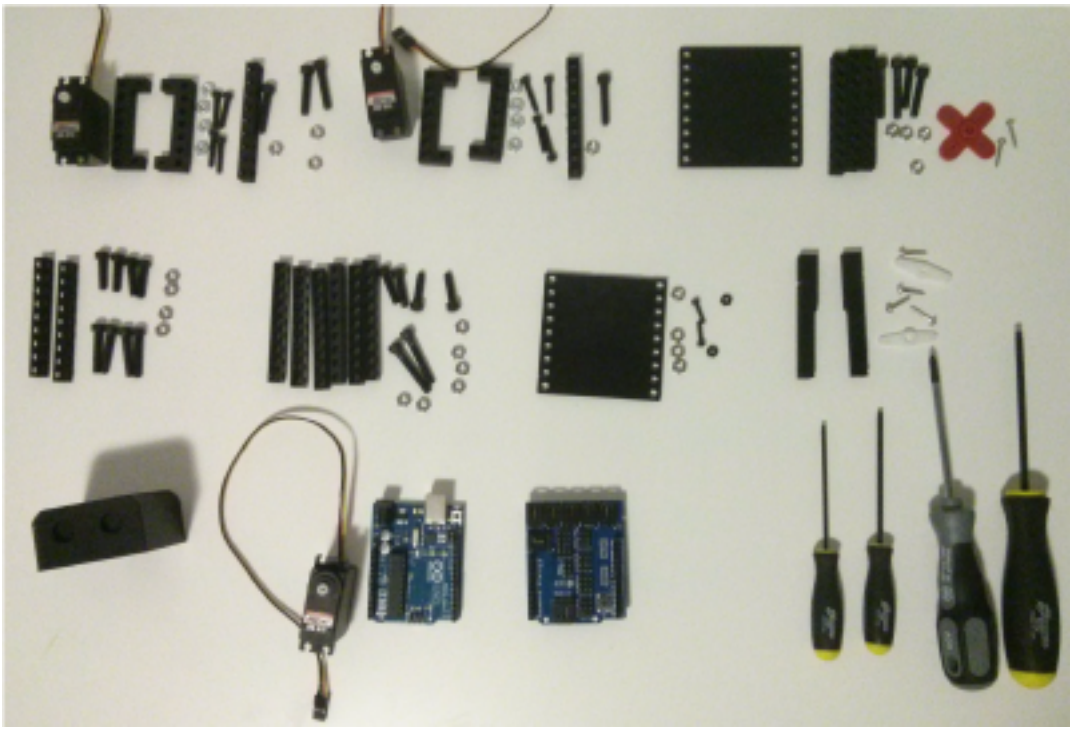
DANCEBOT - THE DANCING ROBOT

OVERVIEW

In this chapter we're going to assemble a small, humanoid robot designed to dance. The moving parts of the robot are called "Bitbeam". Bitbeam is LEGO Technic-compatible, but made with a 3D printer. 3D printing is a beginner-friendly, low-cost, open-source alternative for creating mechanical robot parts.Dancebot will have the following features:

- Swivel head side to side.
- Rotate each arm up and down.
- Bust a move.

PREPARATIONS



To build a Dancebot, you'll need:

Electronics:

- 1 Arduino UNO
- 1 Arduino Sensor Shield V4
- 3 HiTec HS-311 servos
- 1 servo horn (Red, X-shaped)
- 2 servo horns (White, straight)
- 6 servo horn mounting screws

Printed parts:

- 1 Dancebot head- 2 Bitbeam Arduino mounting plate
- 4 Bitbeam servo mounts
- 2 Bitbeam servo arms
- 13 Bitbeam 1x9 beam

Nuts and bolts:

- 8 x 3/4-inch 10-32 bolts
- 10 x 1-inch 10-32 bolts

- 4 x 1 1/2-inch 10-32 bolts
- 8 x 6-32 bolts
- 2 x 1/2 inch 4-40 bolts
- 22 x 10-32 nuts
- 8 x 6-32 nuts
- 2 x 4-40 nuts

Assembly tools:

- 1 x 7/32 hex driver- 1 x 3/32 hex driver- 1 x 5/32 hex driver- 1 x Phillips #1 screwdriver

LET'S BUILD A DANCEBOT!

FIRST ARM

Parts you need to build first arm:

Use 6-32 nuts and bolts to attach 2 Bitbeam servo mounts to the servo. After that, use 1-inch 10-32 nuts and bolts to attach 1x9 Bitbeam beams to Bitbeam servo mount.


STANDARD
HITEC
HS-311

DANCING DRONES

OVERVIEW

In this workshop you'll learn about the wonderful world of autonomous flying robots, specifically the AR Drone 2.0 provides a high level API to send commands, read data back and stream video from it's HD camera.

We'll start writing basic programs to take off and land, and before you know it you'll be using feedback from a wealth of onboard sensors to perform more impressive maneuvers and behaviours.

PREPARATIONS

- Insert a fully charged battery into the AR Drone: https://www.youtube.com/watch?v=QdFsd9R3vJ8
- Download the FreeFlight app for your iOS or Android device.
- Create a folder to work in (something like `nodecopter`)
- *Optional*: Install Node.js on your computer: http://nodejs.org/download/
- *Optional*: Install the ar-drone npm module with `npm install ar-drone` into the folder

MAIN FLOW

Now connect to the drone's WiFi with your smartphone, start the FreeFlight app and make a test flight with it's Piloting feature to learn how the drone behaves.

Once you've done that, save this to a file and execute it:

```
var arDrone = require('ar-drone');
var client = arDrone.createClient();

client.takeoff();

client
  .after(5000, function() {
    this.clockwise(0.5);
  })
  .after(3000, function() {
    this.animate('flipLeft', 15);
  })
  .after(1000, function() {
    this.stop();
    this.land();
  });
```

See how your drone takes of, rotates clockwise and even does a flip! Amazing. Now let's try customising your script with different commands:

Basic directional commands:

- `client.takeoff()`
- `client.land()`
- `client.up(speed)`
- `client.down(speed)`
- `client.clockwise(speed)`
- `client.counterClockwise(speed)`
- `client.front(speed)`
- `client.back(speed)`
- `client.left(speed)`
- `client.right(speed)`
- `client.stop()`

Animation options:

- `phiM30Deg`
- `phi30Deg`
- `thetaM30Deg`
- `theta30Deg`
- …

> More details for the API can be found in the readme: https://github.com/felixge/node-ar-drone#ar-drone

Combine these together and get your drone dancing around the room!

Now that you've got the hang of the basics, there are three different challenges to attempt, you can try them in any order.

MAKE YOUR OWN DRONE CONTROLLER

sending commands from a controller (xbox, keyboard, arduino, browser)

SECOND FLOW

reading nav data from the drone and visualising it (in a browser, or terminal)

EYE IN THE SKY

streaming video/png data back and displaying it (most likely in a browser)

ADDITIONAL TASKS

Additional tasks to be filled here. For those who want more or are more advanced if you don't have second flow.

FAQ

- Crashes
- Won't take off
- The App
- How much can an AR Drone lift? - Not much, about 100g before it becomes unstable.

ADDITIONAL RESOURCES

- Nodecopter website - http://nodecopter.com/hack
- Nodecopter modules on NPM - https://npmjs.org/browse/keyword/nodecopter
- Nodecopter projects on GitHub - https://github.com/search?q=nodecopter

THERMOSTAT EASY AS PIE

– DESIGNING PHYSICAL INTERFACES

OVERVIEW

Most of contemporary technology depends on interaction with people. The key element of successful devices is the way they meet needs of users. That means they should do something users need and do it in a way that is easy to understand and learn.

That's where interaction design comes in. The process and tools for creating interactions and interfaces that are easy to use and that empowers users and make them happy.

We will try to learn the basics and a little bit more of interaction design approach. The example that is going to be used will be a thermostat. This kind of devices are often too complex and designed in a way that requires too much effort to learn. But the example of innovative Nest thermostat shows that it might be done in a simpler way.

BUTTONS AND MINDS – BRIEF INTRODUCTION TO INTERACTION DESIGN

There are three main areas that should be taken into consideration when designing the device (interface, service, almost anything):

Users – Who is going to use the device?

(Is it a man, women or a child? How old is he or she? Is he or she familiar with technology? Motivated?)

Tasks (and needs) - Why is the device being used?

(Is it for fun or work? What kind of goal is user going to accomplish? Why? Is the task complex? Repetitive? Important?)

Context (and situation) – When and where does the interaction take place?

(When the device is going to be used – inside or outside? Is the situation stressful? Is the device connected? What are the default parameters? Will user wear gloves? Is there a lot of sun? Is it noisy?)

The basic concepts that should be with us during whole workshop are the following:

- **Affordance** – as Wikipedia puts it "is a quality of an object, or an environment, which allows an individual to perform an action" – in other words, the shape, behavior or other feature that suggests how the element should be used e.g. physical button by its shape suggest pressing
- **Control** – the way the devices state or parameter can be changed – control allows user to operate the device – it can be button, touch panel, knob, switch etc.
- **Ergonomy** – all the efforts and approach "intended to maximize productivity by reducing operator fatigue and discomfort" i.e. how to adjust technology to our mental and physical capability
- **Feedback** – information about what is happening or what are the consequences of performed action - all interactions should generate some feedback – turning device on/off, changing it's state, changing parameters of the

device etc. Feedback can be provided through sound, colour, text etc.

- **Feeling of control** – user want and should be able to control the device – user often try to control the device based on the mental model that they have created.
- **Interaction** – continuing cycle of action and reaction (e.g. between humans and devices)
- **Mental model** – the concept of how the device works that user creates in his own mind (it does not have to - and often doesn't – reflect the reality); "Great devices work as expected"
- **Relation** – special or logical connection between elements – e.g. labels should be places on or next to controls that they refer to so it is easy to recognize their function
- **User** Centered Design – the design methodology that stresses the role of the user in the design process – the product should be tested as soon as possible and challenged by real users
- **Persona** – a tools that helps to focus on the needs of users – basically a description of typical user with the context and needs

THE TASK

Your task will be to design the prototype of new and innovative thermostat.

ASSUMPTIONS:

The thermostat is the only tool for adjusting the temperature in the whole room (no need to adjust single radiator).

The thermostat should inform the user about current temperature in the room and the temperature that is programmed as target temperature (the temperature that thermostat tries to achieve).

User should be able to program the temperature for specific hours and days of week. (e.g. after 23.00 on Monday to Friday, whole Saturday) and possibly to use weather reports to adjust the temperature (e.g. if the temperature outside is over 18 degrees, switch heating off).

User should be able to adjust the temperature right now (thus overwriting current program).

User should be able to switch thermostat to "out of home" state.

SOLVING A PROBLEM

Use materials provided to design the mock-up (prototype) of the device. Work in teams of 2 or 3.

You can use shapes that are prepared on the station or cut out your own shapes. If possible prepare a few (4-6) mock-up showing different states that will allow to "tell the story" and will help you to simulate and to describe the behavior of the device.

Think about controls for input (e.g. changing the temperature, creating new program, choosing the day of the week) as well as a way of providing feedback (e.g. current temperature, program set etc.).

You have about 30 minutes to prepare the prototype and brief presentation.

PRESENTATION OF THE SOLUTION

Each team will be asked to prepare a short presentation of their mockup. The presentation should take no more than 3 to 5 minutes. It should include presentation of the general concept as well as description of how device allows to:

- inform about current temperature in the room
- inform about the temperature that is programmed as target temperature
- program the temperature for specific hours and days of week. (e.g. after 23.00 on Monday to Friday, Whole Saturday) and possibly to use weather reports to adjust the temperature (e.g. if the temperature outside is over 18 degrees, switch heating off).
- adjust the temperature right now (thus overwriting current program).
- switch to "out of home" state.

ADDITIONAL RESOURCES

Designing Devices, Dan Saffer http://www.amazon.com/Designing-Devices-Dan-Saffer-ebook/dp/B006QY2GAQ

HACK YOUR HOUSE

OVERVIEW

In this workshop you will learn how to build your own smart RFID-controlled deadbolt lock. We will be using an Arduino Uno, a servo, and a few sensors to modify a deadbolt lock so that you can control it via software and unlock it with an RFID tag. You will be able to mount our Arduino to your existing deadbolt without physically modifying or damaging it!

PREPARATIONS

Download the Arduino IDE from http://arduino.cc/en/Main/Software and install it on your system.

MAIN FLOW

STEP 1: HELLO ROBOT!

We are going to start off by making sure that you are able to upload code to your Arduino using the Blink tutorial (http://arduino.cc/en/tutorial/blink). One of the best things about Arduino is the amazing open source community that exists around it. We will be making use of a number of open source examples as we build our smart lock!

Plug your LED power into pin 13 of your Arduino and ground into the GND pin next to it. The shorter pin is Ground (-) for LEDs.

Plug your Arduino into your computer via USB.

Open the Arduino IDE, select: File > Examples > 01.Basics > Blink and click the "Upload" button to install the Blink program on your board.

If you successfully uploaded your program, the LED that you plugged into your Arduino should be blinking on and off every second!

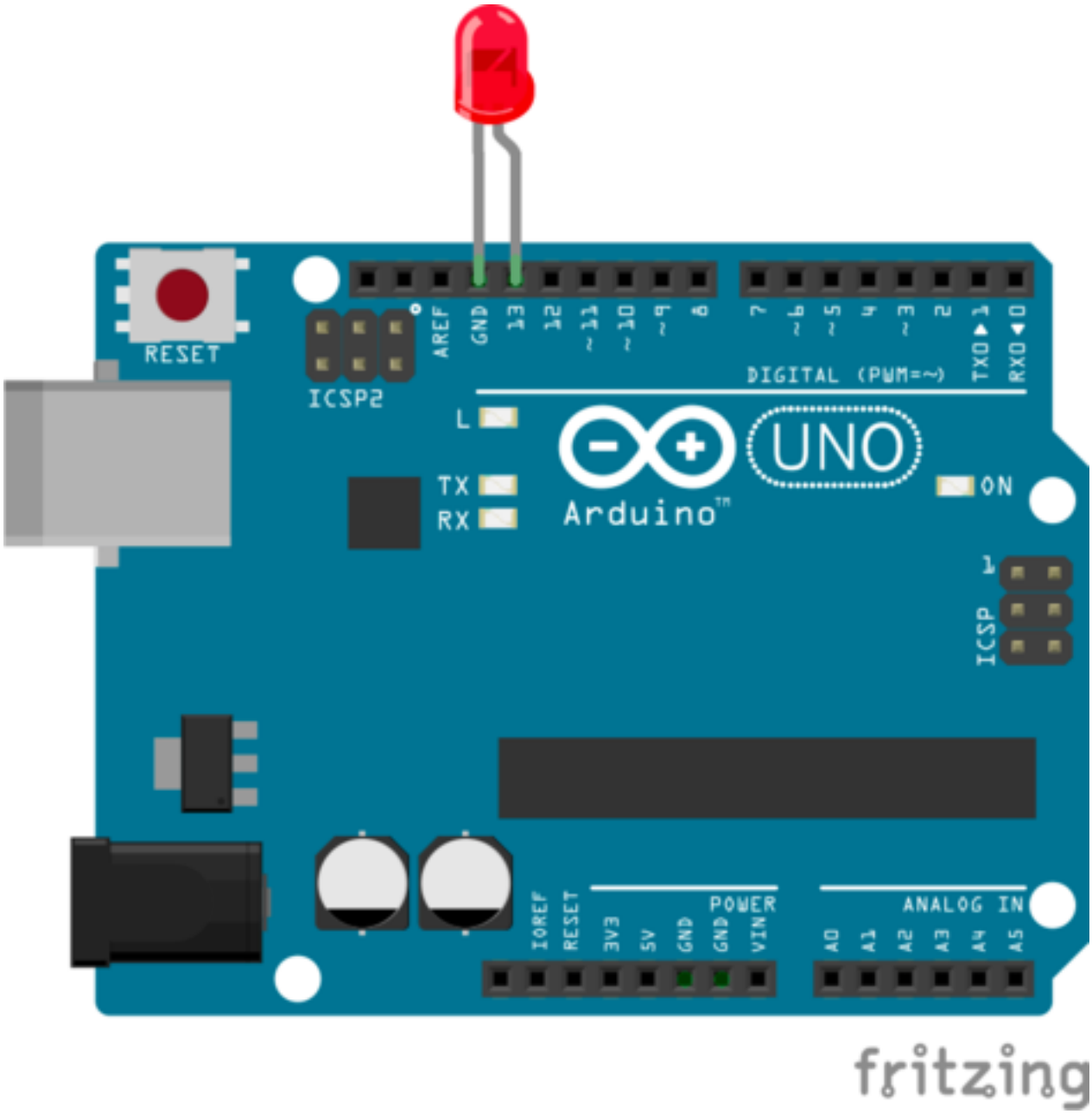
STEP 2: CONNECT AND TEST A SERVO

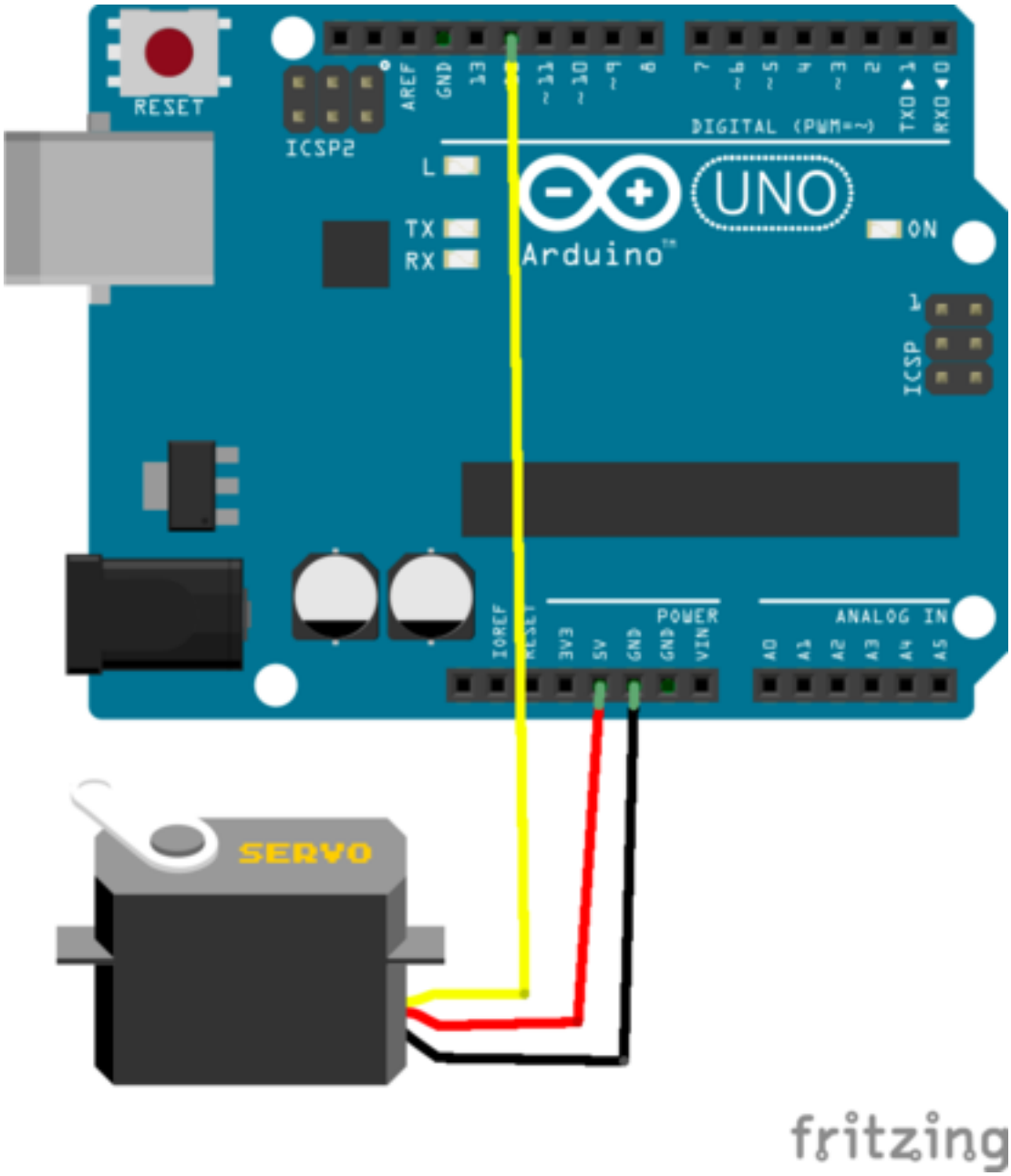
Now that we are sure that you can program your Arduino, we will move on to the fun stuff and connect our servo to our board.

A servo is a type of motor that typically has a 180 degree movement radius. Later in the workshop, we will be using our servo to control the movement of a deadbolt lock.

For now, we just want to connect our servo to our Arduino and make sure we are able to control it via a simple program.

A servo motor has three wires - power (red), ground (black), and control (frequently yellow or white). Connect the power wire to the 5V pinout on the Arduino and the ground wire to the GND pinout next to it. We can then connect the control wire to Digital Pin 12 on our Arduino as in the following diagram:


RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM=~)
L
TX
RX
Arduino™
UNO
ON
ICSP
POWER
IOREF
RESET
3V3
5V
GND
GND
VIN
ANALOG IN
A0
A1
A2
A3
A4
A5
fritzing


RESET
ICSP2
AREF
GND
13
12
11
10
9
8
7
6
5
4
3
2
TX→1
RX←0
DIGITAL (PWM=~)
L
TX
RX
Arduino
UNO
ON
ICSP
1
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
SERVO
fritzing

Now that our servo is connected to the Arduino, we will need to write the software to control it. Let's create a new file and start it off with the following code:

```
#include <Servo.h>
Servo myservo;
```

This snippet includes the Servo library and defines a new Servo object that we will use later to control our servo motor.

Next, we can create two methods: `loop()` and `setup()` - these methods are common to every Arduino program. The `setup()` method runs when the Arduino turns on, and the `loop()` method constantly runs on a loop (as the name would suggest).

In our `setup()` method, we will attach our myservo object to pin 10 where we plugged in the servo earlier. The method should look like this when we are done:

```
void setup()
{
  myservo.attach(12);
}
```

Now that the Arduino knows where to find our Servo, we can experiment a bit. In order to move the servo to a new position, you use the `myservo.write(pos);` method. pos is an integer from 0 to 180. Try using `myservo.write(pos);` in both the `setup()` and `loop()` methods - see what happens when you set pos to different values. Note that you can use the delay() method to pause execution between different commands. For example:

```
myservo.write(180);
delay(1000);
myservo.write(0);
```

This snippet would move the servo to position 180, pause for 1 second, and then move the servo to position 0.

After each modification, Upload your code to the Arduino again. You can view the progress of your upload in the log console on the bottom of your Arduino IDE.

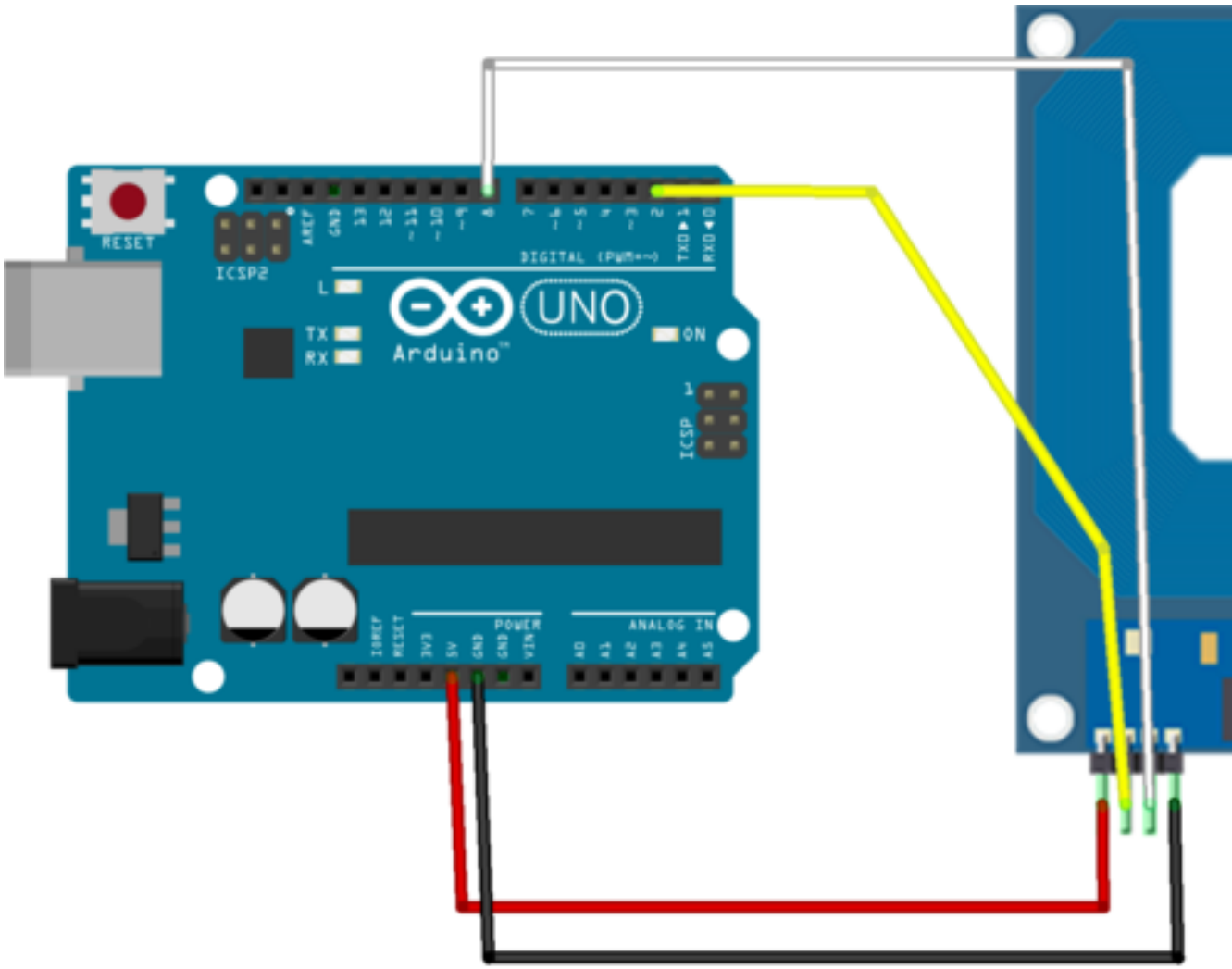
Once you are comfortable with controlling your Servo, we can move on. Additional examples are available here: http://arduino.cc/en/Tutorial/Sweep

STEP 3: CONNECT AND TEST RFID READER

Now that we have programmed our Arduino to control a Servo, we can add our first sensor - the RFID reader. We will be testing using an example sketch from http://playground.arduino.cc/Learning/PRFID

The Parallax RFID Reader that we are using is actually fairly simple to program. It has four pins: GND, VCC (power), ENABLE, and SOUT. The ENABLE pin takes a digital voltage - either high or low. When the voltage is low, the RFID reader is on and actively reading inputs. When it is high, it is deactivated. The SOUT pin is a Serial interface that outputs the RFID tag code as a series of 10 digits.

You will need to wire the GND to your Arduino's GND, VCC to Arduino's +5V, ENABLE to Digital Pin 2 on the Arduino, and SOUT to Digital Pin 8 on the Arduino. I like to plug my RFID reader into a breadboard before wiring


RESET
ICSP2
L
TX
RX
UNO
Arduino
ON
ICSP
DIGITAL (PWM~)
POWER
ANALOG IN

it to my Arduino, as it makes it easier to prototype.

Once it is wired up, we will be writing a program that uses the Arduino's SoftwareSerial library to read values from the RFID reader and print them out in our Serial Monitor (a simple tool that the Arduino IDE provides to receive data from the Arduino via a Serial port and display it to you, the developer).

Create a new Sketch (Arduino program) that has a `setup()` and `loop()` method. Then include the SoftwareSerial library at the top of the sketch:

```
#include <SoftwareSerial.h>
```

Next we will set up some variables that will be used in our `loop()` method a little later:

```
int  val = 0;
char code[10];
int bytesread = 0;

#define rxPin 8
#define txPin 9
SoftwareSerial RFID = SoftwareSerial(rxPin,txPin);
```

An important thing to take note of here are that we are defining our `SoftwareSerial` pins as digital pins 8 and 9, which explains why our RFID Reader's SOUT is connected to pin 8. We also define the val, code, and bytesread variables. val will store the single digit value read by our RFID reader. code is an array that we will push the digits read by val onto to build the full tag code. And bytesread is a counter to determine how many digits we have read from the RFID reader for an expected tag code, which is 10 digits.

Now we can move on to our `setup()` method. Most of our logic for the RFID reader is going to be in the `loop()` method, so all our `setup()` method needs to do is initiate our Serial monitor and enable our RFID reader by setting digital pin 2 to LOW and opening the 2400-baud SoftwareSerial connection. Add the code from below:

```
Serial.begin(2400);    // Hardware serial for Monitor 2400bps
pinMode(2,OUTPUT);     // Set digital pin 2 as OUTPUT to connect it to the RFID /ENABLE pin
digitalWrite(2, LOW);  // Activate the RFID reader
RFID.begin(2400);
```

Now we can move on to the meat of our application, the `loop()` - I will explain each block of code line by line, as it is fairly complex the first time you encounter it.

First we need to check the output of the RFID reader:

```
if((val = RFID.read()) == 10)
{
}
```

The RFID reader outputs 10 at the beginning of a tag code, and 13 at the end. This lets us know when to stop reading bytes for the current tag code, and move on to the next one.

Within this if statement, we will be running a while loop that reads in 10 digits from the RFID reader:

```
bytesread = 0;
while(bytesread<10)
{
}
```

The expected length for our tag code is 10 digits, so we only loop until we have read in 10 bytes.

Then we can add the rest of our code within the while loop:

```
val = RFID.read();
if((val == 10)||(val == 13))
{  // if header or stop bytes before the 10 digit reading
  break;                        // stop reading
}
code[bytesread] = val;          // add the digit
bytesread++;                    // ready to read next digit
```

For every iteration of the loop, we read in a single byte outputted by our RFID reader. If that byte is 10 or 13 (our start or stop codes), we break out of our loop. If it is any other value, we add it to the code array and increment bytesread. Last but not least, we need to print our tag code out to the Serial Monitor. After our while loop's end bracket (but still within our large if statement), we will add the following code:

```
if(bytesread == 10)
{  // if 10 digit read is complete
  Serial.print("TAG code is: ");   // possibly a good TAG
  Serial.println(code);            // print the TAG code
}
bytesread = 0;
delay(500);                       // wait for half a second
```

If we have read in 10 bytes successfully, we print it out in the Serial Monitor and then reset our bytesread counter and wait for half a second to prevent us from looping faster than the RFID reader actually reads values. And that's it, now you can Upload your code and open the Serial Monitor via the Tools menu. Once you have the Serial Monitor open, make sure to change the baud rate to 2400. Now when you hold an RFID tag near the RFID reader, you should get a tag code outputted to your Serial Monitor!

STEP 4: TURNING OUR SERVO WITH RFID

Now that we have successfully moved our Servo and read an RFID tag in separate steps, we can try to put it together.

Take the working RFID reader code from Step 3, and attempt to combine it with our working Servo movement code from Step 2.

In order for this to work, we will actually have to connect a separate power supply for our servo to use as the Arduino

does not provide enough voltage for both the RFID reader and the servo at the same time.

Take the provided battery, and re-wire the Servo to it. We connect all GND wires to the Arduino's GND, but the RFID reader's VCC goes to the Arduino's +5V output and the Servo's VCC goes to our external battery pack. This becomes much easier if you experiment with moving the Servo when you print out the Tag Code in our Serial Monitor. Note that since the Servo takes time to move, you may need a larger delay in your RFID reader code.

The Servo library documentation is available here if you would like to dive deeper into it: http://arduino.cc/en/reference/servo

If you get stuck and would like to compare your code with the solution, you can see it here: https://gist.github.com/jonmarkgo/9058657

STEP 5: ACCESS CONTROL

Hooray, now you can move a servo whenever someone swipes their RFID tag on our reader! But how do we know if the card being swiped has access to our home? In this step, we will add access control to only allow certain RFID tags to unlock our door.

To find the Tag Code for your RFID tag, open the Arduino IDE Serial Monitor and look for the code that is printed out when you swipe your tag. Copy that code, and put it in a variable in your sketch.

Now, where you currently move the Servo after a tag is detected, add an if statement to check if the tag that was scanned is the verified tag! You may need to use the String library for this comparison: http://arduino.cc/en/Reference/string

If you get stuck and would like to compare your code with the solution, you can see it here: https://gist.github.com/jonmarkgo/9058825

STEP 6: MOUNTING TO YOUR LOCK

Now that the software side of our project is done, we must mount the servo to our deadbolt lock. I prefer to use household items when prototyping these types of applications, though if you have access to a 3D printer I would recommend designing and printing your own lock mount.

For the purpose of prototyping, we will be using cardboard and duct tape to mount our servo to our lock - just like astronauts do!

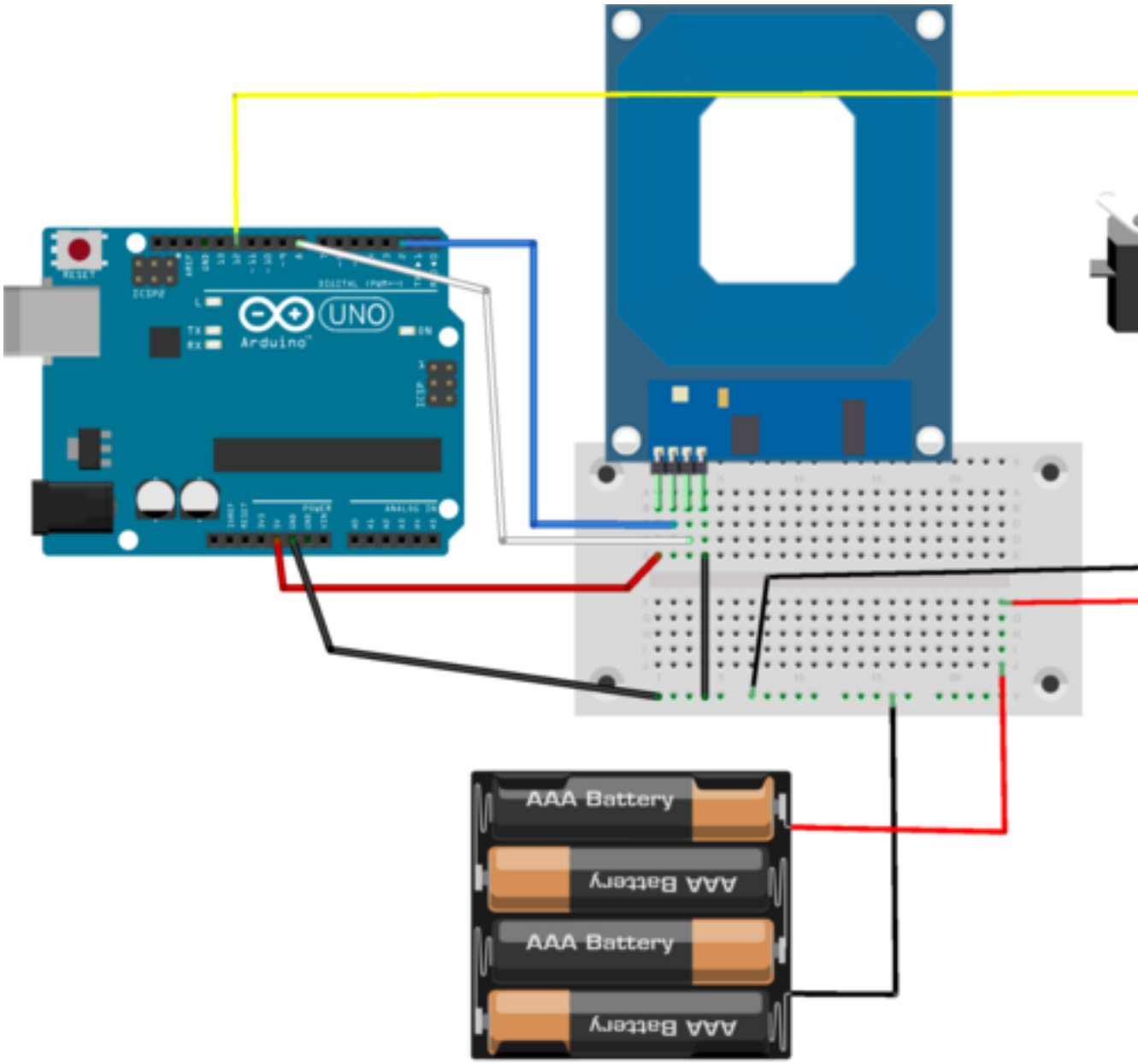
Attach the two metal rods to your servo with screws and washers.

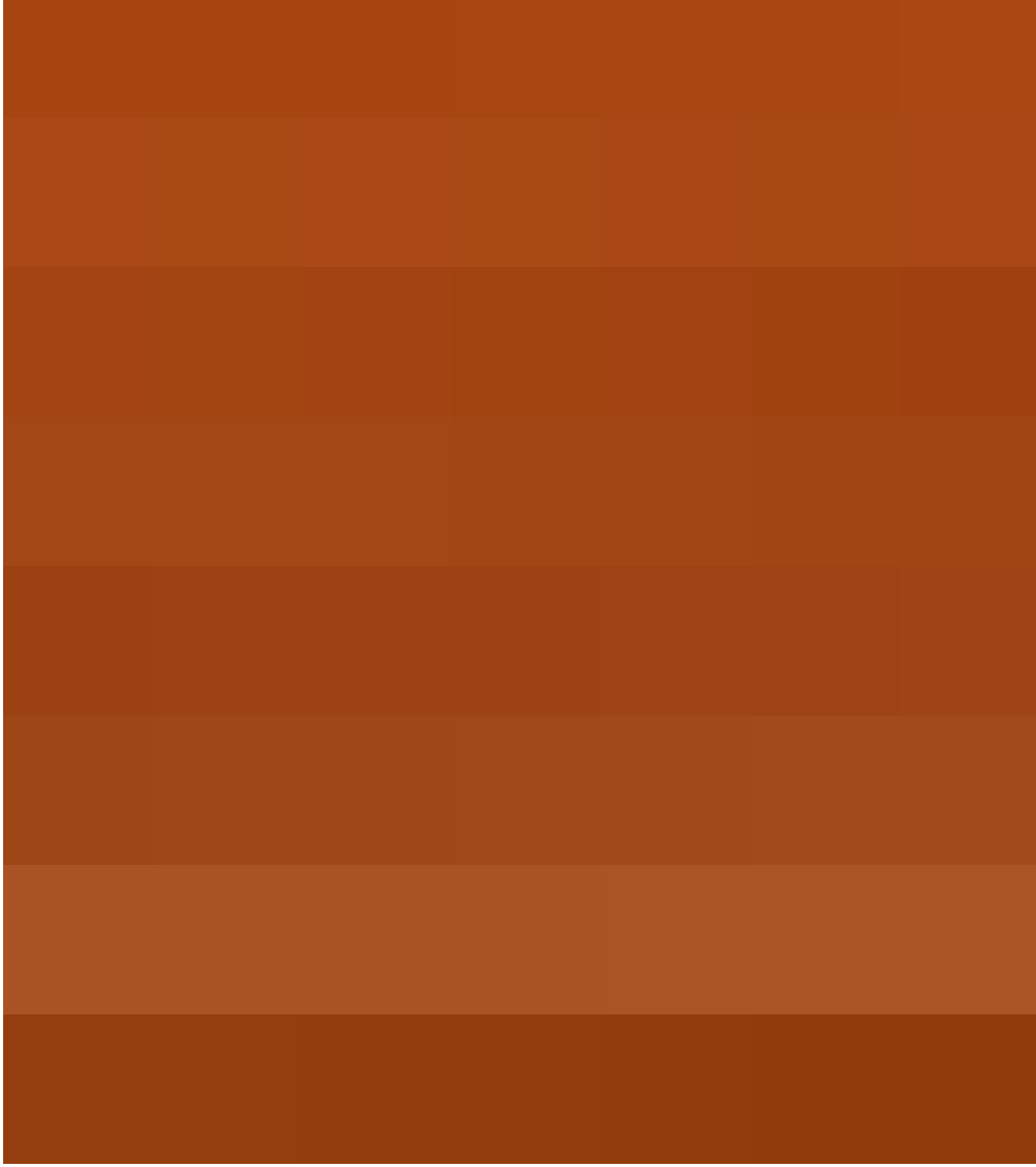
Now use a piece of cardboard (or other stiff material) to make a tighter bond between the servo and the lock:

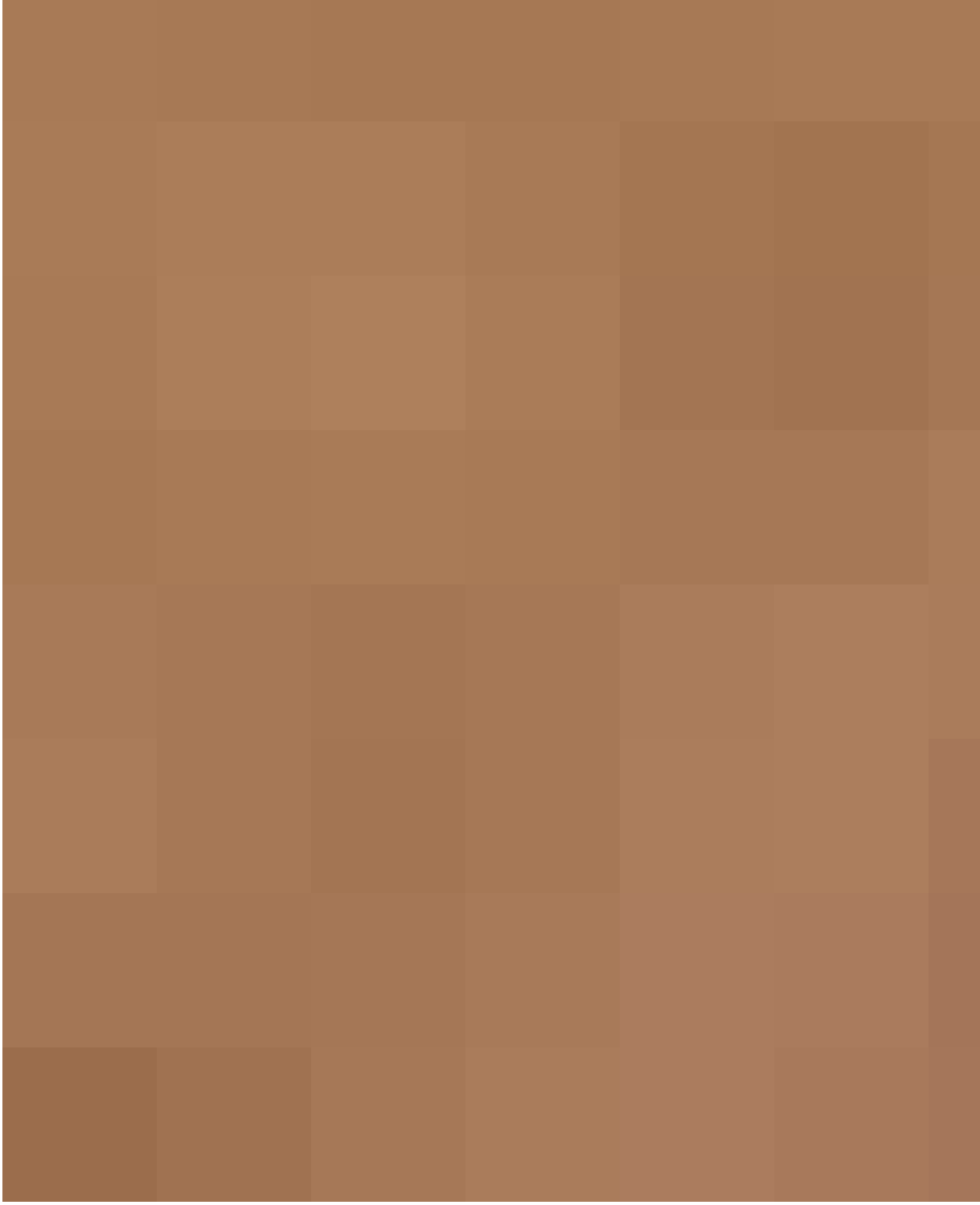
Now you can tape the servo to the deadbolt lock. Make sure it is positioned on the correct side so that the direction that the servo turns in is aligned with the direction that the lock turns in:

And last but not least, we will tape our servo's arms to the deadbolt itself:

Now you have your fully mounted (and fully impermanent) servo-controlled deadbolt:


UNO
Arduino
AAA Battery
AAA Battery
AAA Battery
AAA Battery





SECOND FLOW (OPTIONAL)

At this point, you have built an RFID-controlled deadbolt using your Arduino, a servo, and an RFID reader. But when it comes to home automation, there is always more to be done!

EXPERIMENT 1: MOTION-CONTROLLED DEADBOLT

This first experiment is the simplest (and most insecure) - we will unlock or lock our door every time we detect movement in front of it.

Take the PIR motion sensor and wire it into your Arduino (VCC to +5V, GND to GND, and control to digital pin 2) - NOTE that the middle wire of the PIR sensor is GND, not the black wire. This is strange, but true. The red (+) wire is still VCC (the sensor should have a + and AL marking on it to distinguish this):

You will need to use an internal pull-up to activate the motion sensor. This is done by setting digital pin 2 as an INPUT pin and then writing a HIGH value to it. Once you Arduino turns on, you need to wait a second or two for the PIR motion sensor to get a reading on a room without movement. Then you can start waving your arms all around!

When motion is detected, lock or unlock your deadbolt.

You can find additional details on the sensor here: https://www.sparkfun.com/products/8630

If you get stuck you can find the solution here: https://gist.github.com/jonmarkgo/9060847

EXPERIMENT 2: SMS-ACTIVATED DEADBOLT

For this experiment we only need our Servo connected to our Arduino. Most of the work will be on the Software side.

Once your Arduino is wired up, you need to make sure you have node.js installed and that you have a Twilio account.

You can go to https://www.twilio.com/ to sign up for a Twilio account. Twilio is an API that makes it easy for us to send and receive text messages or make and receive phone calls. In this case, will be receiving text messages.

You will then need to install node.js from http://nodejs.org/ and the ngrok utility from https://ngrok.com/download

Once you have your Twilio account and you have successfully installed Node.js and ngrok, we can install the necessary modules with the following command in your Terminal:

```
npm install serialport twilio express
```

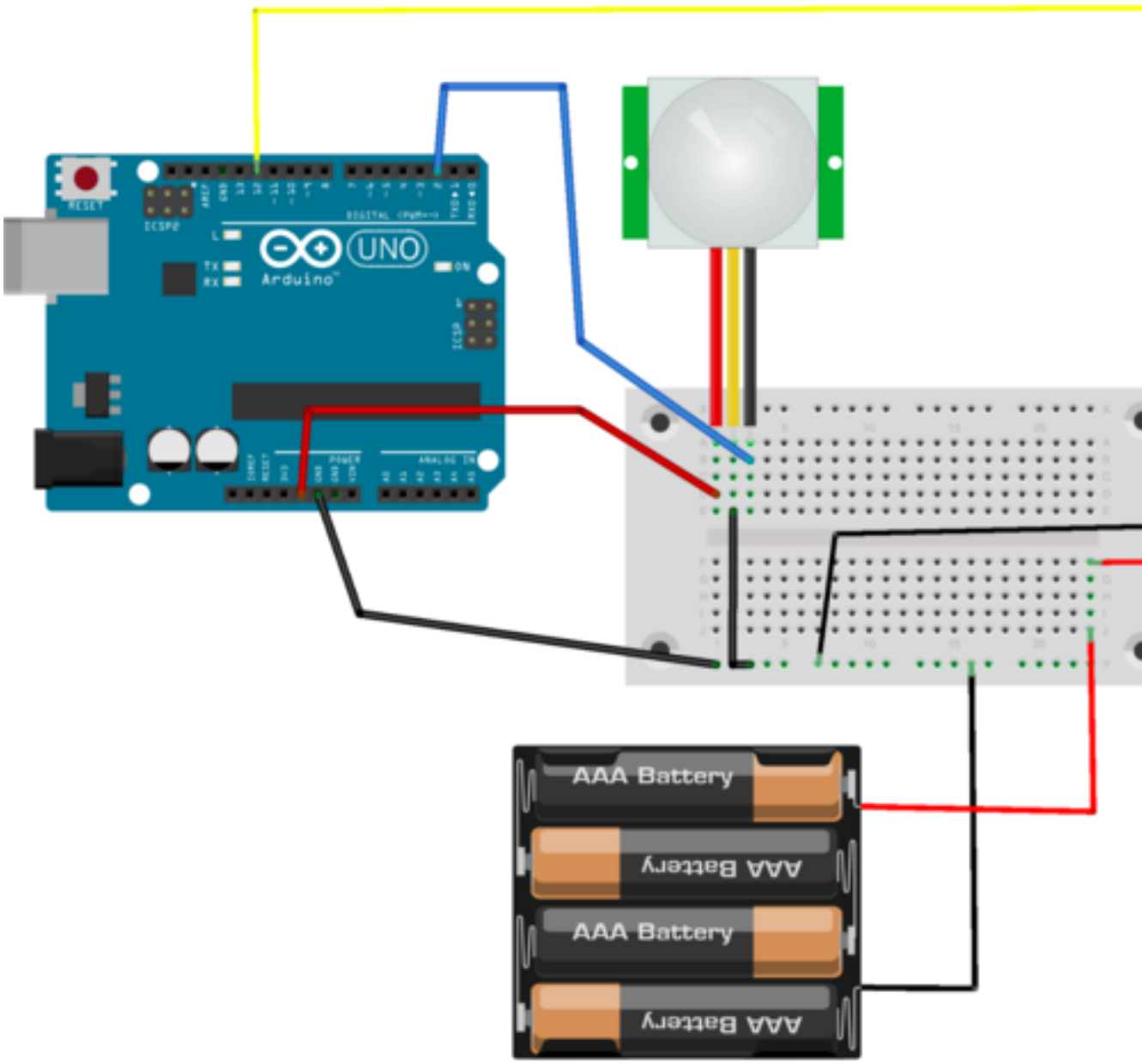
Node-Serialport makes it easy to communicate with your Arduino via a Serial connection from a Node.js program. We will be using it to receive HTTP request for incoming text messages from Twilio, and passing instructions along to the Arduino to lock or unlock your deadbolt.

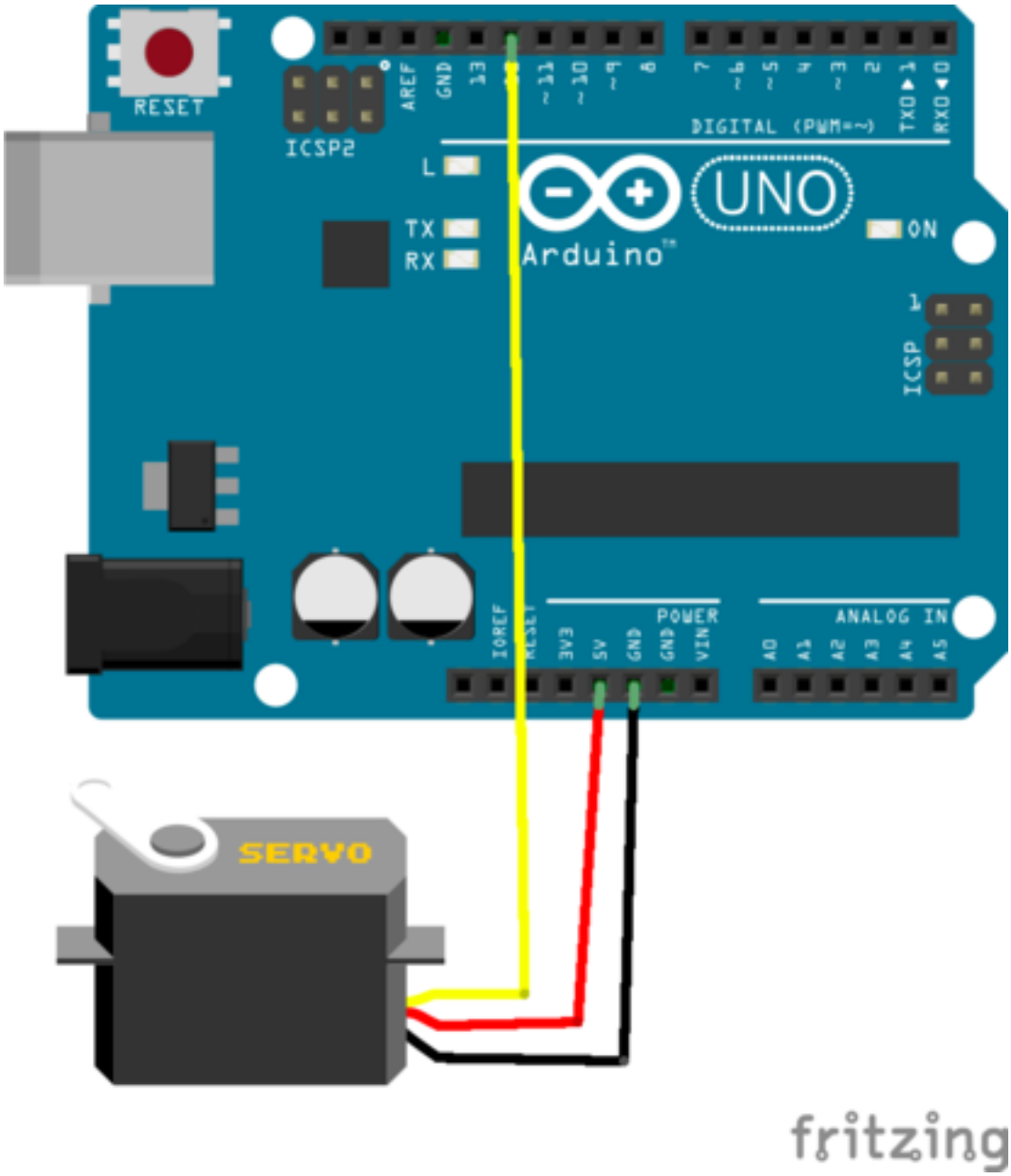
Express is a simple node.js web framework. And the twilio module makes it easy to communicate with the Twilio API.

First things first, we can set up our new Arduino sketch. The sketch is actually fairly simple. All it needs to do is open a 9600-baud Serial connection, attach to your servo (on pin 12), and read in a character from the Serial connection to determine whether or not it should move the servo motor.

I would encourage you to try this on your own by making use of the Servo code from Step 2 (and your experience so far) as well as the Serial library documentation, available here: http://arduino.cc/en/reference/serial

I configured my Arduino to lock or unlock my deadbolt whenever it receives the character “V” on it’s Serial port. You


UNO
Arduino
AAA Battery
AAA Battery
AAA Battery
AAA Battery


RESET
ICSP2
AREF
GND
13
DIGITAL (PWM=~)
TXD
RXD
L
TX
RX
Arduino
UNO
ON
ICSP
POWER
ANALOG IN
IOREF
RESET
3V3
5V
GND
GND
VIN
A0
A1
A2
A3
A4
A5
SERVO
fritzing

can test this functionality by opening the Serial monitor and typing V to see if the servo moves. When the Arduino locks the deadbolt it responds with the "L" character and when it unlocks it responds with "U".

If you run into trouble with this portion of the code, you can see the solution here: https://gist.github.com/jonmarkgo/9061828

Now that our Arduino is prepared to receive and send instructions on the Serial port, we can dive into something brand new: a node.js program to control our Arduino!

Create a new file in your editor of choice called nodelock.js and start off the file by requiring the modules that we installed earlier using npm:

```
var twilio = require('twilio'),
    serialPort = require("serialport").SerialPort,
    express = require('express');
```

Next we will want to set up our new express server (more documentation available here: http://expressjs.com/) and our `serialPort` connection (more documentation available here: https://github.com/voodootikigod/node-serialport):

```
var app = express();
var serialPort = new SerialPort("/dev/tty.usbmodem1411", {
  baudrate: 9600
});
```

Note that we are specifying which USB port to connect to and the baud rate. You may need to change the value of the USB port on your own computer. You can find the name of your active USB port in the Arduino -> Tools -> Port menu.

Next we will set up our HTTP route, called `/sms`

```
app.use(express.bodyParser());

app.post('/sms', function(req, res){
});
```

We are telling express to accept POST requests at the `/sms` route, and to parse the POST body using its bodyParser, which will make it easier to access the parameters sent by Twilio's incoming SMS.

Now that we have our route to accept an incoming SMS, we will want to check if the number the message is from has permission to lock and unlock the door.

```
app.post('/sms', function(req, res){
  if (req.body.From == "+12128675309") {
    console.log("verified number!");
  } else {
    console.log("Wrong number!");
    sendMessage(res, "Invalid number!");
  }
```

```
});
```

We are checking the From POST parameter and logging whether or not it is verified.

In the block for a verified number, we can now add the handler to send and respond to data from the Arduino's Serial connection.

```
serialPort.once('data', function(data) {
  if (data.toString().indexOf('U') > -1) { //check if the Arduino returned a U for unlocking
    sendMessage(res, 'Unlocking!');
  }
  else if (data.toString().indexOf('L') > -1) {
    sendMessage(res, 'Locking!');
  }
  else {
    sendMessage(res, 'ERROR');
  }
  console.log('data received: ' + data);
});

serialPort.write("V", function(err, results) {
  if (err) {
    console.log('err ' + err);
  }
  console.log('results ' + results);
});
```

This looks fairly meaty, but what is really going on is fairly straightforward. We set up an event handler to receive Serial data from the Arduino. This event handler checks if the Arduino has sent "U" or "L" - we then take this value and return an SMS response to the user using the sendMessage function (which we will write shortly).

After setting up our event handler, we write "V" to the Arduino's serial connection to tell it that a verified SMS has been received, and it should now lock/unlock the door.

Towards the top of the file we can now create our sendMessage function, which takes two arguments - `res` and `message`:

```
function sendMessage(res, message) {
  var resp = new twilio.TwimlResponse();
  resp.message(message);
  res.type('text/xml');
  res.send(resp.toString());
}
```

`sendMessage` is called to generate a TwiML response for the user. TwiML is the subset of XML that Twilio uses to pass around instructions. In this case, we are telling Twilio to respond to the SMS message we have received with another

SMS message. So the user might send in “unlock” and we might send back “Unlocking!” via Twilio SMS.

Now that our SMS handler is configured, we can finish up our application by opening our SerialPort and starting up our Express web server:

```
serialPort.open( function () {
  app.listen(3000);
  console.log('Listening on port 3000');
});
```

And that’s all of our code. Now, if you Upload the sketch we wrote to your Arduino and run your nodelock.js script by typing node nodelock.js into your Terminal we will be good to go.

If you have run into errors and would like to compare with the solution, you can check it out here: https://gist.github.com/jonmarkgo/9061701

To finish things off, we will need to configure Twilio.

Once you have made a new Twilio account (which also provides you with your very own phone number), we can head over to our Twilio dashboard at https://www.twilio.com/user/account and click on the Numbers tab. Then click into the number you purchased during account signup.

Here, there are two fields: Voice Request URL and Messaging Request URL. We will be using the Messaging Request URL to tell Twilio where to send data about incoming text messages.

But since Twilio communicates via HTTP requests, we will need a publicly accessible web URL for it to POST to when it receives an SMS to your number. For this, we will be using ngrok - the utility you installed earlier.

Once you have started up your node.js server with `node nodelock.js`, open a new Terminal window and type `./ngrok 3000` from the directory you installed ngrok to. Here you will be given a forwarding URL. Take this forwarding URL, append `/sms` to it, and put it into your Twilio Messaging Request URL in your dashboard. Save your Number’s settings, and try sending an SMS to it! Your lock should lock and unlock as long as you set the verified number to be yours.

Great work and happy hacking!

OVERVIEW

During this workshop you will learn how to load prepared program onto Arduino and use it with PC/Mac running Python or Ruby. You'll blink a LED, check temperature/humidity/light intensity, measure distance or detect motion. All this using only couple lines of code!

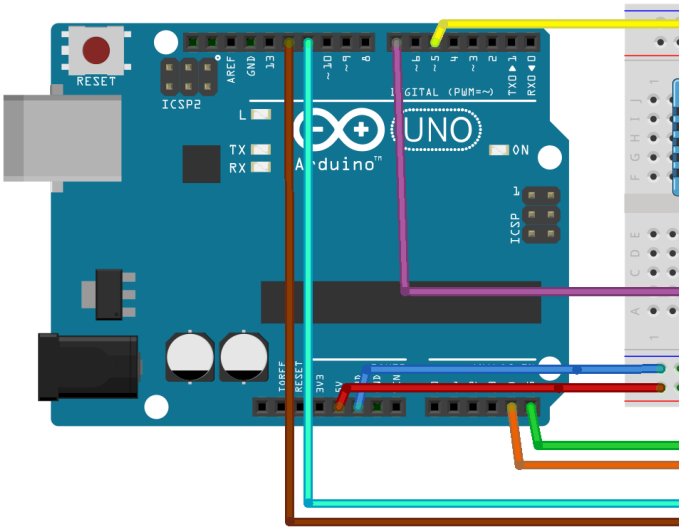
PREPARATIONS

ARDUINO IDE INSTALLATION

To install Arduino IDE, follow guides for your operating system: * Windows: http://arduino.cc/en/guide/windows * OS X: http://arduino.cc/en/Guide/MacOSX * Linux: http://playground.arduino.cc/Learning/Linux

CONNECTING THE WIRES AND SENSORS

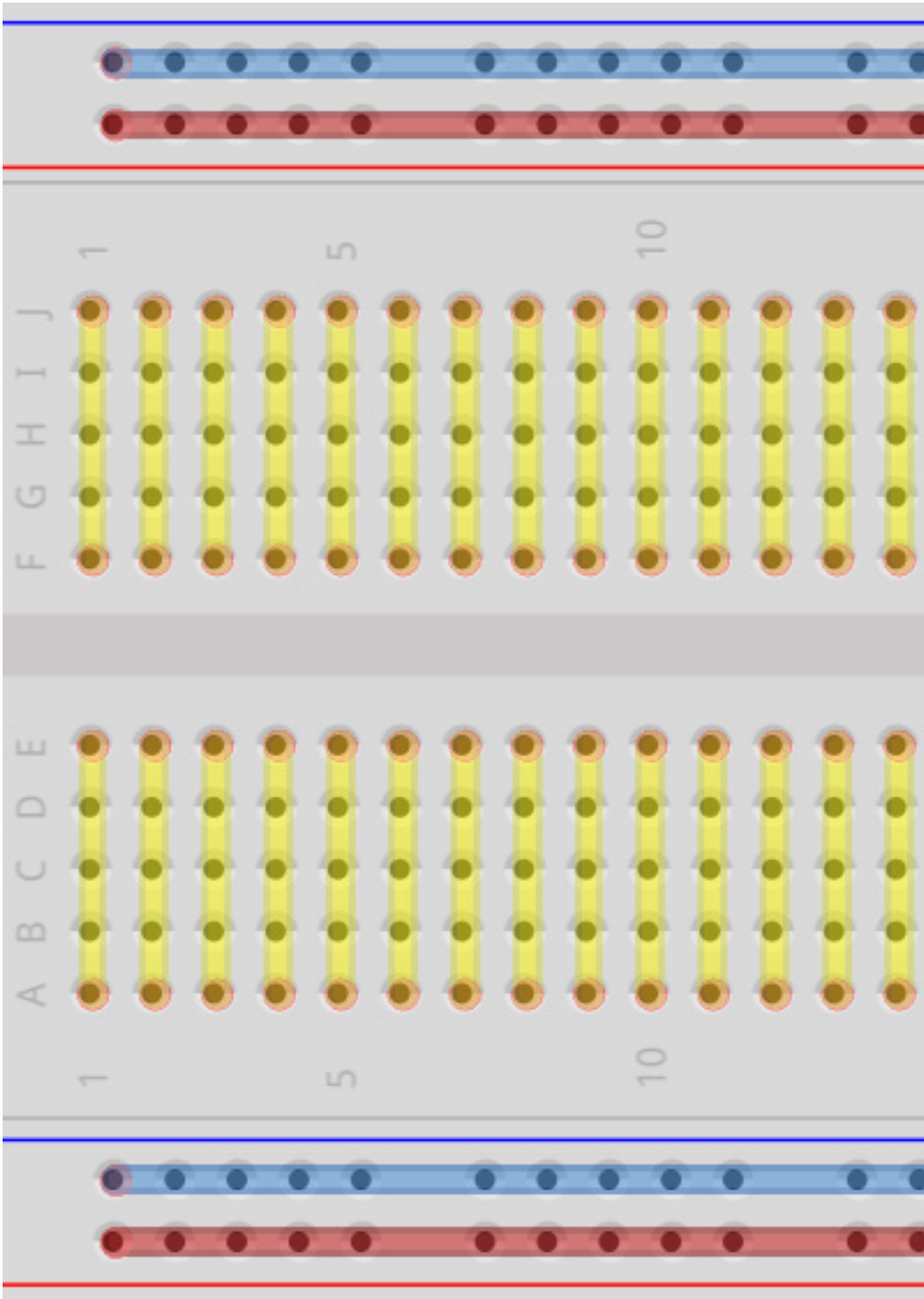
Along with Arduino you will need: * a breadboard (white board with holes ;)) * DHT11 temperature/humidity sensor (the small blue box with holes) * BH1750 light sensor module * HC-SR04 ultrasonic ranging module (which looks a lot like Wall-E) * HC-SR501 PIR motion detector (looks like a dome) * pack of colorful wires



At the end of this workshop, your breadboard will look like this:
You don't have to connect all the wires right now, as we'll explain everything step by step in later parts of this workshop.

If you never used a breadboard before, you may wonder how all those holes are connected with each other. Here is a

small cheat sheet for you:



INSTALLATION OF PYTHON OR RUBY

RUBY

To install Ruby you should, depending on your operating system: * Windows: download and install RubyInstaller from http://rubyinstaller.org/downloads/ * OS X: if you have Mountain Lion (10.8) or Mavericks (10.9) you already have it * Linux: install ruby with your distribution package manager

PYTHON

Python comes out of the box with both OS X and Linux. If you're Windows user, you can follow this guide: http://docs.python-guide.org/en/latest/starting/install/win/

MAINFLOW

SETTING UP YOUR ARDUINO

You won't actually program Arduino using Ruby or Python. You can do it but it's a lot of hustle, so we chose more popular and easier way: communicate with Arduino using serial port. For this to work, you need to load Arduino with our small code which listens for commands and takes action like turning a LED on and of, or returns a value from one of our many sensors.

If you followed Arduino guide (which we hope you did), you should know how to upload a program to the board. Download ZIP file for this workshop: https://github.com/meal/makerland-workshop/archive/master.zip

In *ArduinoSensors* directory you'll find *ArduinoSensors.ino* file. Open it in Arduino IDE and upload to the board.

> Feel free to dig into this code. It contains a lot of comments which explain how it works. You can add new sensors or output devices like servos or relays. Play around with it!

After loading this sketch (BTW Arduino calls programs "sketches" and we'll use both names in this workshop), open *Serial Monitor* in Arduino IDE (loupe button). Try sending: `D1` and watch LEDs on Arduino. Then send `D0` to see one of the LEDs to turn off.

PYTHON

CONNECTING TO ARDUINO

When connect your board to the computer using USB cable, it creates a serial port which you will use to communicate with it from Python.

> Arduino IDE uses the same port to upload sketches to the board. Because serial port can be used by one application at once, your upload may fail when you have same port open somewhere else. But no worries, just close it and you'll be able to upload again.

Depending on your operating system, serial ports have different names. this is why we prepared small programs which list all ports available in your system.

Firstly you have to install some packages which you'll need in this workshop. Go to *Python* directory from ZIP you've downloaded when setting up Arduino. To install requirements type in terminal/command line:

```
cd PATH_TO_PYTHON_DIRECTORY_FROM_ZIP
sudo pip install -r requirements.txt
```

> What did you just installed? You've installed three libraries: PySerial - library for communicating using serial

> port. Essential in our workshop :) PyAudio - library for handling audio. You'll use it to play a WAV sound file request - great library for making HTTP requests in easiest manner possible

Great! Now type in terminal:

```
python 1_ports.py
```

This should print a list of available ports like this:

```
Available ports:

    /dev/cu.Bluetooth-Serial-1
    /dev/cu.Bluetooth-Serial-2
    /dev/cu.Bluetooth-Modem
    /dev/cu.Bluetooth-PDA-Sync
    /dev/cu.usbmodem1d11421
```

In this case we're sure that `/dev/cu.usbmodem1d11421` is our Arduino. If there's more unknown ports, just check your Arduino IDE which port it's using (*Tools -> Ports menu*). Copy it as you'll use it in all the following parts.

LET THERE BE LIGHT

Ok, it's finally time to use Python to control Arduino! You can start interactive Python shell just by typing `python` in terminal. Now lets blink some LEDs:

```
# Import time module
import time
# Import PySerial module
from serial import Serial
# Initialize and open serial port
serial = Serial('YOUR_PORT')
# Set variable containing current LED status to False
led_on = False
# Repeat indefinitely
while True:
    # Negate LED status
    led_on = not led_on
    # If it should be on...
        if led_on:
        # ...then send command turning on the LED
            serial.write("D1")
    # But if it should be off...
    else:
        # ...send the off command
```

```
        serial.write("D0")

    # Wait one second
    time.sleep(1)
```

> If this is your first time with Python: you see the indentation? It's very important in Python. You have to be consistent to use Tab or Spaces in your code.

BTW: you should find source to this exercise in `Python/2_blink.py` file.

You can stop this now, pressing `Ctrl+C`.

You've just used your code to influence an existing object :) But this is child's play. Lets try something more advanced.

TIME FOR SOME MEASUREMENTS

First thing before writing code is to connect HC-SR04 ultrasonic ranging module with Arduino. Grab your breadboard, some cables and connect them like this:

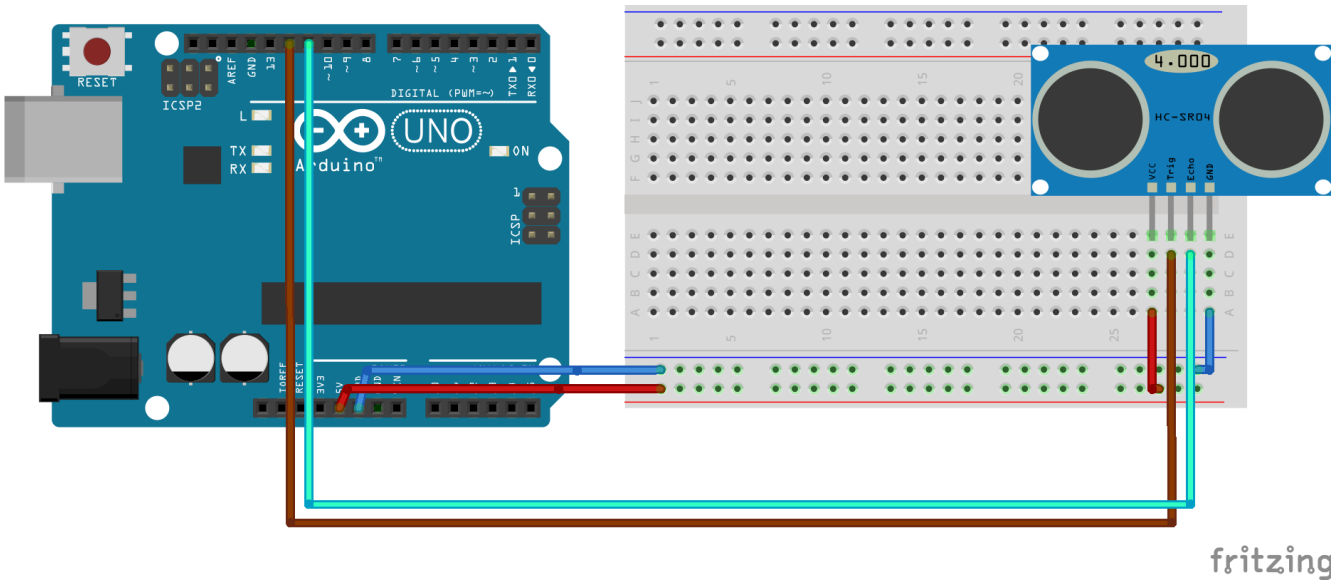


Figure 1: Sonar

Colors are not important, but be concise when using them. This way you won't make silly mistakes like connecting wrong wire to wrong hole.

If everything is connected properly (check it twice) it's time to go to Python console again and check the value of sensor:

```
# Again repeat indefinitely
while True:
    # “S” command tells Arduino to use sonar
    serial.write("S")
    # Wait on data from serial and store it in variable
    distance = serial.readline()
    # Print this value on screen
    print distance
    # Wait a second
    time.sleep(1)
```

Now when your code is running, put your hand in front of sensor and observe value on the screen. See it changing? It's not very accurate but with some tweaks in Arduino code you can make a digital rangefinder!

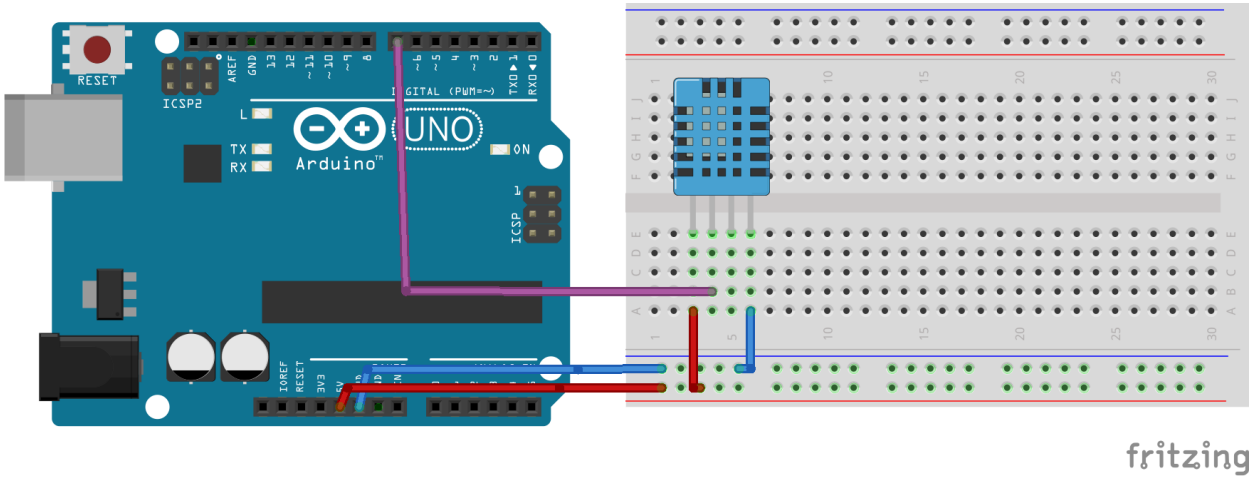
BTW: you should find source to this exercise in `Python/3_sonar.py` file.

Press `Ctrl+C` to stop this code.

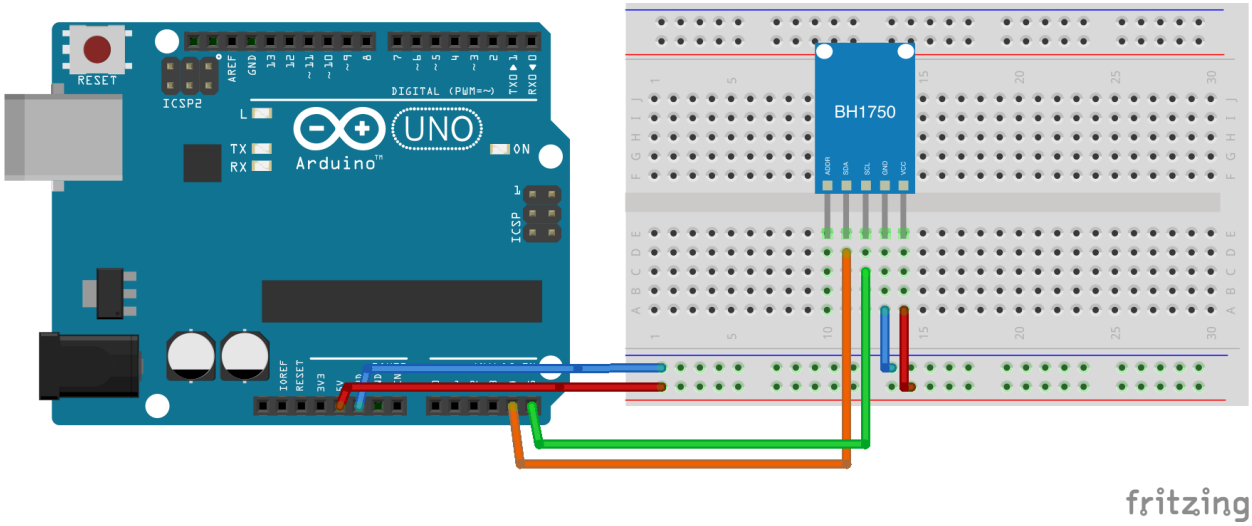
HOT OR NOT

Along our sensors, you have DHT11 which can measure two things: temperature and humidity. Combining it with BH1750 light sensor, you can have a small meteo station, which provides information about current weather outside.

But firstly the wires. Lets start with DHT11 (don't disconnect the sonar, it will be required later):



Now connect BH1750:



Important: connecting BH1750 requires restart of Arduino, so just unplug and plug USB cable again.

It's Python time:

```
# You've disconnected Arduino remember? Now you have to open port again
serial = Serial('YOUR_PORT')
# Endless loop
while True:
    # Command for checking temperature
    serial.write("T")
    # Read the temperature
    temperature = serial.readline().strip()
    # You have to wait a little bit to read humidity
    time.sleep(0.5)
    # Humidity command
    serial.write("H")
    # Store the humidity
    humidity = serial.readline().strip()
    # Get light intensity
    serial.write("L")
    # Store it
    light = serial.readline().strip()

    # Print values
    print "Temperature: %s°C" % temperature
    print "Humidity: %s%%" % humidity
    print "Light: %slx" % light

    # Wait a second
```

```
    time.sleep(0.5)
```

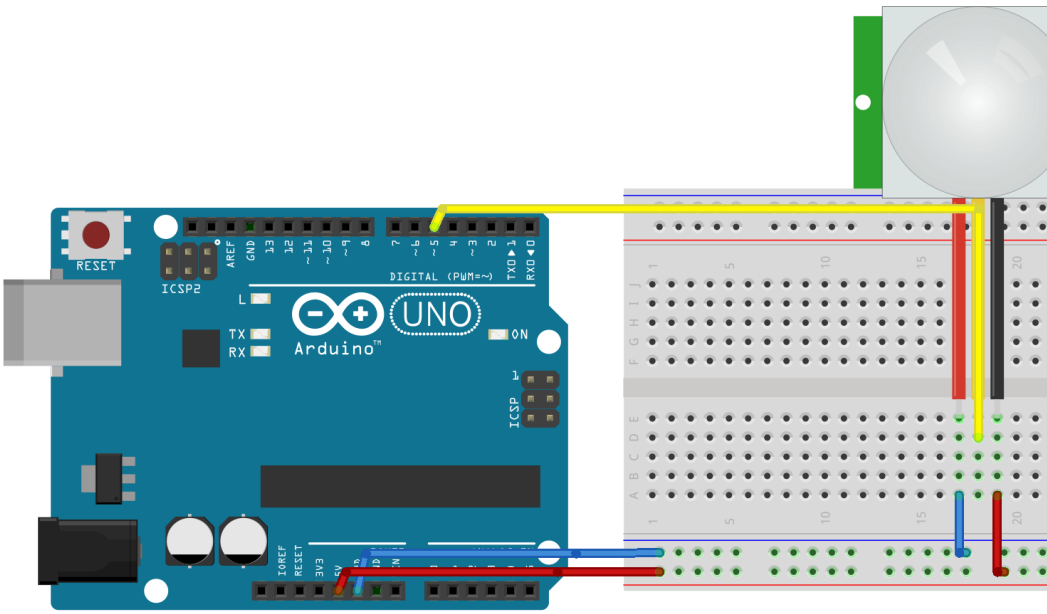
Yay! You can now remove Weather app from your smartphone.

BTW: you should find source to this exercise in `Python/4_meteo.py` file.

INTRUDER ALERT

You have one sensor left: HC-SR501 PIR motion detector. This sensor detects motion based on infrared light. It's similar to ones used in public bathrooms to automatically turn on the light. As it works with infrared it only detects objects which exude heat (i.e. human body, running car). What is important, it detects **motion** not presence. So if you'll hold still in front of it, you won't be detected.

Other quirk of this sensor is long lag after detecting motion. There's potentiometer on a back of it which allows to change it, but event with minimal setting it's about 3 seconds.



Connect the wires like this:
Your PIR may have wires with different colors or don't have wires at all, so just hold it like in the picture above and number pins from left to right.

Supporting code is easy as pie:

```
# Use motion sensor
serial.write("M")
# After first start, sensor needs about 15 seconds to calibrate, just wait for it
serial.readline()
# Loop
while True:
```

```
# Get the value again
serial.write("M")
# Store integer value of the sensor (0 or 1)
motion = int(serial.readline().strip())
# If value is 1
if motion:
    # Alert about suspicious movement!
    print “MOTION DETECTED!”
# Wait one second
time.sleep(1)
```

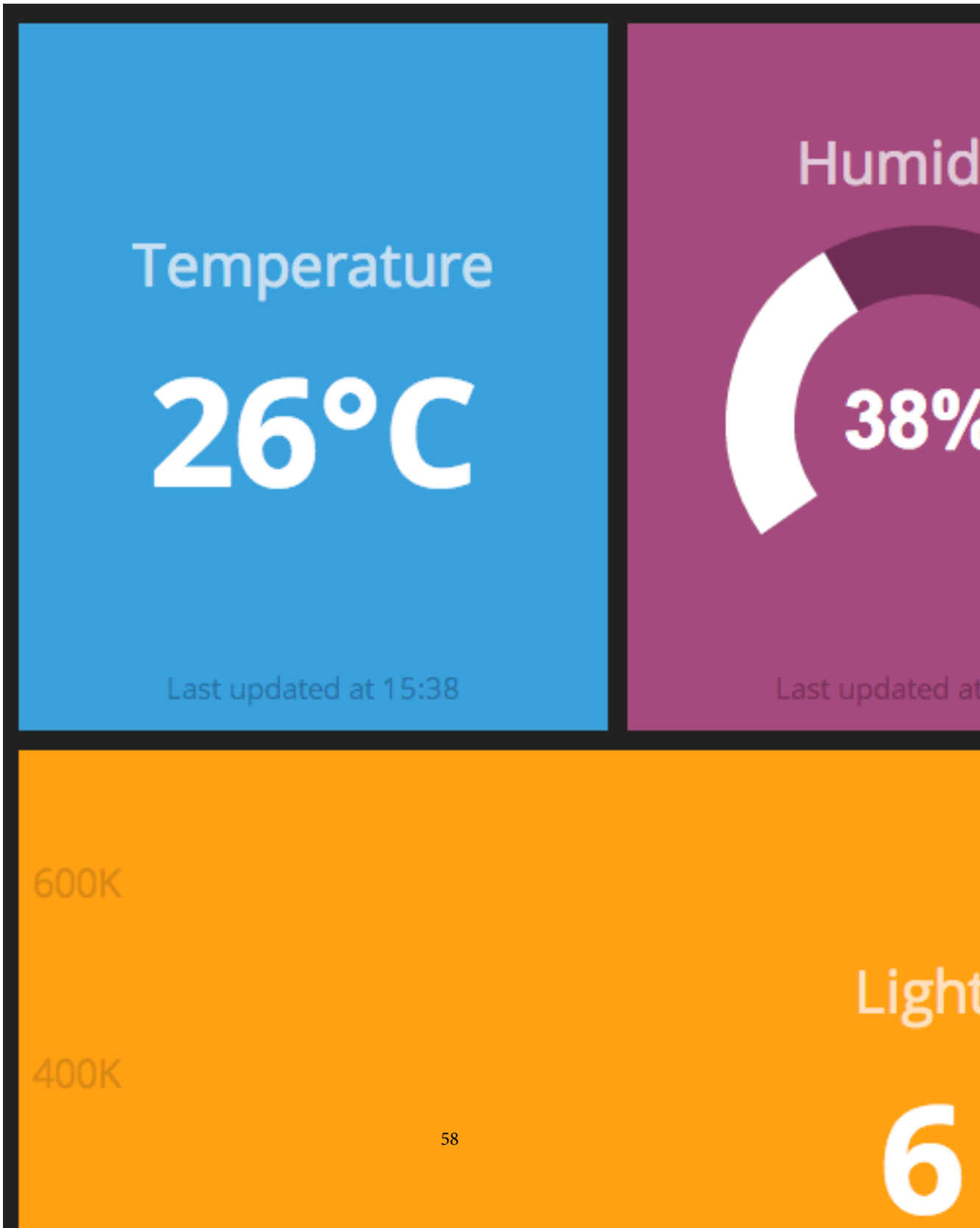
Now wave in front of the sensor :) Downloaded sources contain `5_alarm.py` file which is even cooler as it plays a sound! Try it:

```
python 5_alarm.py YOUR_PORT
```

ALL IN ONE PLACE

Now you know how to use all the sensors, you can use it to create a beautiful web dashboard with all the data.

To make it easier, we've prepared custom dashboard using Dashing. It's super easy to set up and host i.e. on Heroku. Our dashboard looks like this:


Temperature
26°C
Last updated at 15:38
Humid
38%
Last updated at
600K
400K
Light
6

To send data to this dashboard you need to know which ID to use. It should be written on a sticker on your Arduino. If you're not at Makerland, just use `arduino-X` where X is number from 1 to 20.

Now start Python script which sends data to the dashboard:

```
python 6_dashboard.py YOUR_PORT ID
```

If there's no error, visit your dashboard at http://makerland-dashboard.herokuapp.com/ID

Dig into `6_dashboard.py` to see how to send data to Dashing.

> Code for web dashboard is also in ZIP you've downloaded, so feel free to modify it and host wherever you want!

SOUNDS OF THE FUTURE

This one you better try with headphones on :) Do you know what Theremin is? It's a electronic instrument which is controlled without physical contact. Its distinctive sound was often used in older Sci-Fi movies.

Using HC-SR04 ultrasonic ranging module, you can create a poor man's, almost 8-bit sounding theremin with only pitch control.

Start it using:

```
python 7_theremin.py YOUR_PORT
```

Now holding your hand above sonar, move it closer to it and you should hear change in sound pitch!

> How it works? It's very hard to explain it here, so take a look into `7_theremin.py` file which is filled with comments explaining how it works.

Unfortunately this was our last exercise, but now you can go to **Additional tasks** section and try to do it just by yourselve!

RUBY

CONNECTING TO ARDUINO

First at all you need to understand how Arduino communicates with your computer. When you plug your board into your computer's USB port, your operating system will create serial port device.

> Arduino IDE uses the same port to upload sketches to the board. Because serial port can be used by one application at once, your upload may fail when you have same port open somewhere else. But no worries, just close it and you'll be able to upload again.

Names of serial ports depends on your operating system. You can use our programs to list all serial ports available on

your system.

To do so, you have to prepare your environment. Go to *Ruby* directory from ZIP downloaded when you were setting up Arduino. To install required packages type in terminal/command line:

cd PATH_TO_RUBY_DIRECTORY_FROM_ZIP bundle install

> What just happened? You have installed two libraries: serialport - library that will let you communicate using serial port typhoeus - library for making HTTP calls

Okay! Now type in terminal:

```
ruby 1_ports.rb
```

This command should print a list of available serial ports:

```
Available ports:

    /dev/cu.Bluetooth-Serial-1
    /dev/cu.Bluetooth-Serial-2
    /dev/cu.Bluetooth-Modem
    /dev/cu.Bluetooth-PDA-Sync
    /dev/cu.usbmodem1421
```

Assuming that you do not have other devices that may create serial ports connected to computer, we're sure that `/dev/cu.usbmodem1421` is Arduino. If there's more ports, check the right one in your Arduino IDE by using *Tools->Ports menu*.

LET THERE BE LIGHT

Now it's time to use Ruby to control our Arduino! You can start by using interactive ruby shell by typing `irb` in your terminal. Lets blink Arduino built-in LED

```
# require serialport library
require 'serialport'
# Initialize and open serial port
serial = SerialPort.new("YOUR_SERIAL_PORT", 9600, 8, 1)
    # Set variable with LED state to false
    led_on = false
    # Infinitely run this block of code
    loop do
    # negate LED state
    led_on = ! led_on
    # If it should be on...
    if led_on
        # ...then send command turning LED on
```

```
            serial.write("D1")
            # If it should be off..
        else
            # ...send the off command
            serial.write("D0")
        end
    # wait one second for next loop cycle
    sleep(1)
```

> In Ruby, other than in Python it is not require to keep indentation, but it is highly recommended due to code readability.

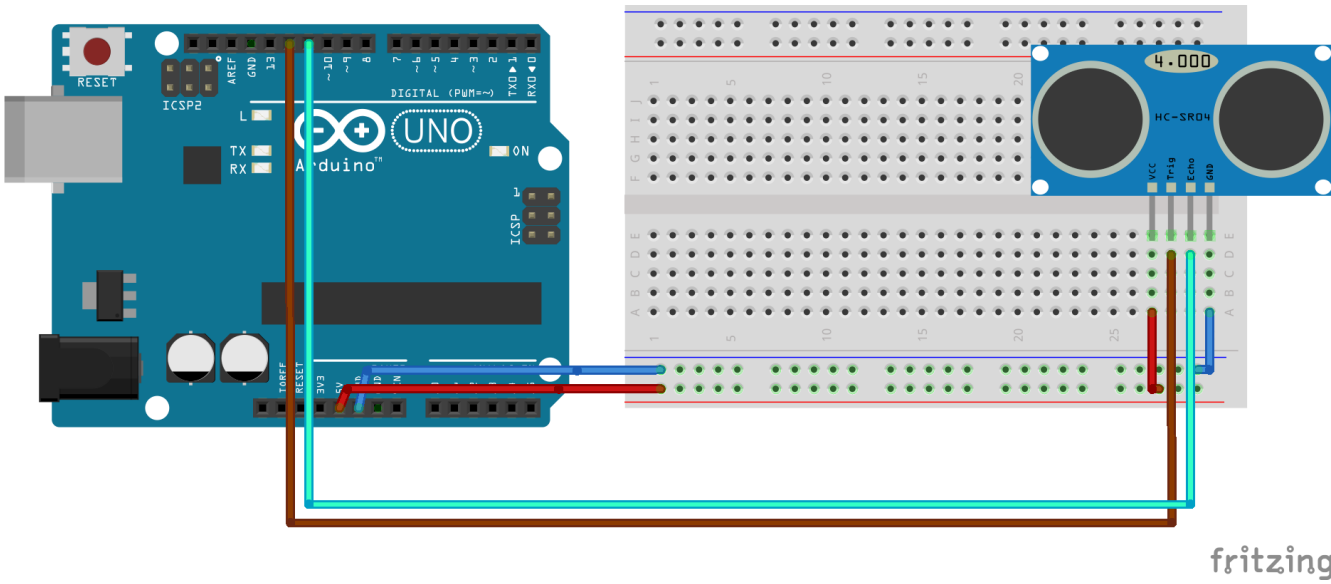
BTW: you should find source to this exercise in `Ruby/2_blink.rb` file.

You can stop this now, pressing `Ctrl+C`.

You've just used your code to play with something already built, lets get to something more advanced.

TIME FOR SOME MEASUREMENTS

Before you write anything lets have fun with wires. Grab HC-SR04 ultrasonic ranging module, Arduino, your breadboard, some wires and connect them all together this way:



You don't have to use the same colors as we do, but be meticulous while using them. By doing so you will avoid mistakes like connecting ground pin from module to Arduino power pin.

Check two or even three times if everything is set up properly and go back to ruby shell:

```
# Lets run it infinitely again
loop do
    # “S”, as you can see in Arduino code, checks sonar
    serial.write(“S”)
    # Read the data from device and store it in a variable
    distance = serial.readline
    # and print it on the screen
    puts distance
    # then wait a second before the next cycle
    sleep(1)
end
```

While your code is running swing your hand in the front of the sensor and observe how value changes on the screen. It might not be very accurate but with some Arduino code tweaks it can get better.

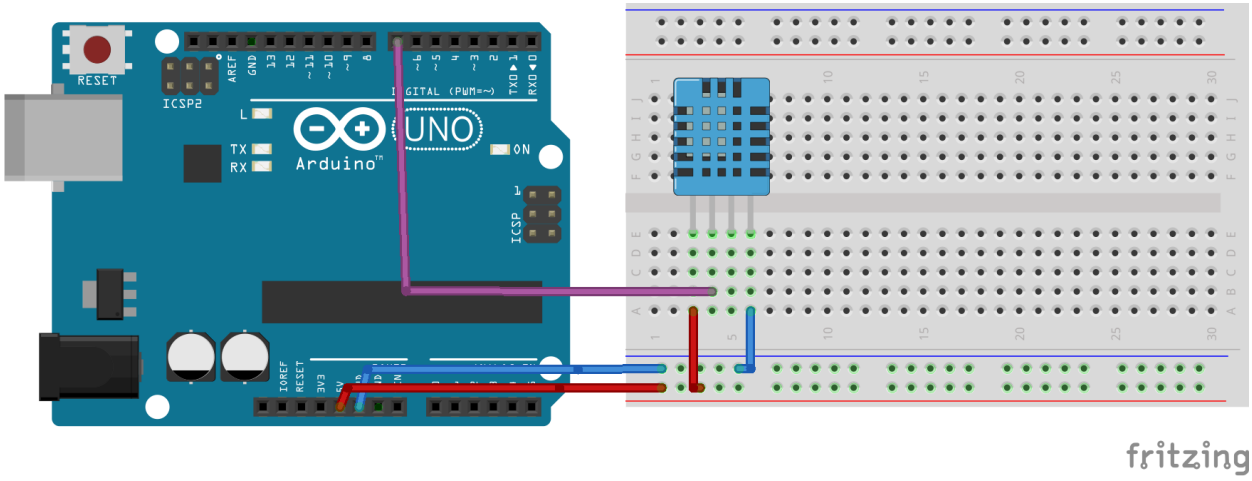
BTW: you should find the source to this exercise in `Ruby/3_sonar.rb` file.

Stop your code by pressing `Ctrl+C`

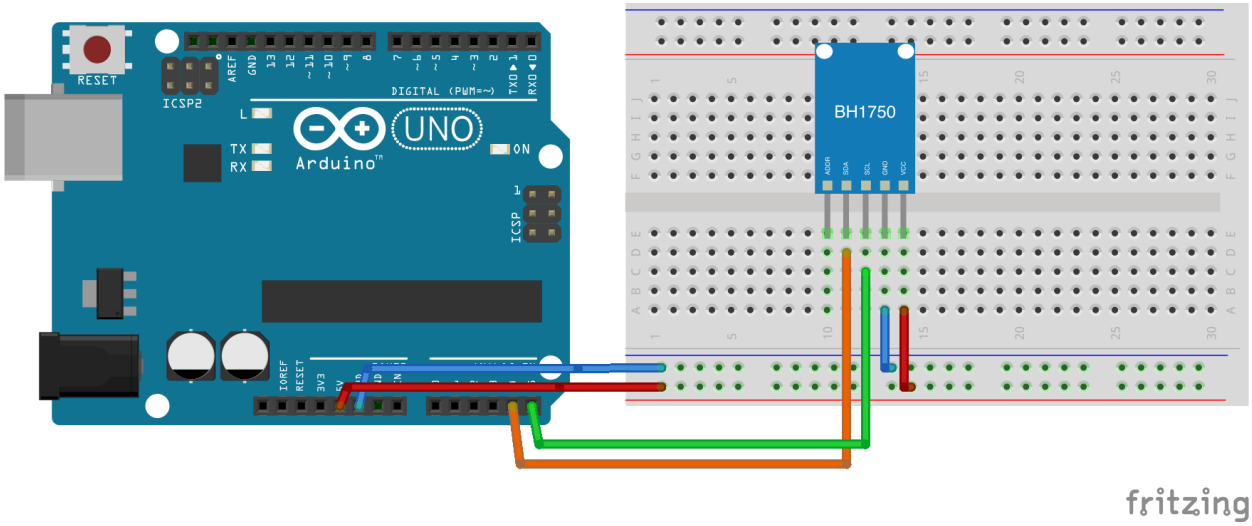
HOT OR NOT

Among sensors you can find DHT11 which can measure temperature and humidity. Together with BH1750 light sensor you can use them to build small meteo station that could provides information about weather outside.

Start with the wires. Don’t disconnect sonar, it will be required late. First lets connect the DHT11:



Now connect BH1750:



Important: connecting BH1750 requires restart of Arduino, so just unplug and plug USB cable again.

Now in Ruby shell:

```
# You have reconnected your Arduino, so you have to initialize serial port again
serial = SerialPort.new("YOUR_SERIAL_PORT", 9600, 8, 1)
# Loop
loop do
    # Tell Arduino to check temperature
    serial.write(“T”)
    # Read value from serial and assign to a variable
    temperature = serial.readline.strip
    # Wait a moment to check humidity
    sleep(0.5)
    # Check humidity on Arduino
    serial.write(“H”)
    # Store value in a variable
    humidity  = serial.readline.strip
    # Get a light intensity
    serial.write(“L”)
    # Store it
    light = serial.readline.strip

    # Print values
    puts “Temperature: #{temperature} *C”
    puts “Humidity: #{humidity}% “
    puts “Light: #{light}lx “

    # Wait a second before the next cycle
```

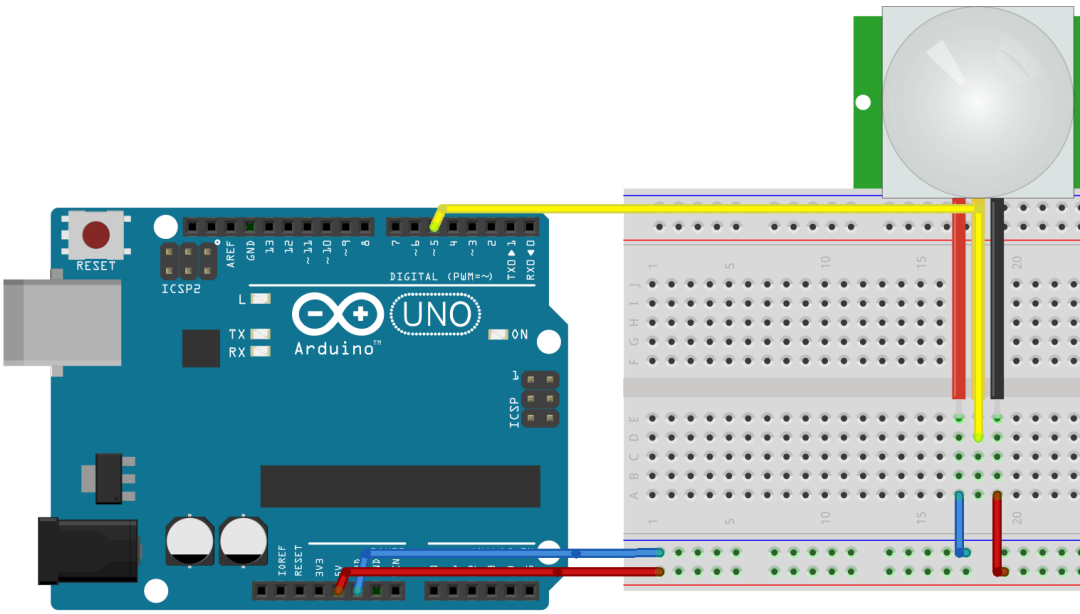
```
    sleep(1)
end
```

Now you have your own meteo station, hooray!

BTW: you should find source to this exercise in `Python/4_meteo.py` file.

INTRUDER ALERT

There is one sensor left: HC-SR501 PIR motion detector. Based on infrared light, this sensor, similar to those used in public bathrooms to turn the lights on, detects moves. As it works with infrared it only detects objects which exude heat (i.e. human body, running car). What is important, it detects **motion** not presence. So if you'll hold still in front of it, it won't detect you



Connect wires this way:

Be sure to hold it like on the picture, so you won't connect it in the wrong way.

Now, the code:

```
# Tell Arduino to use motion sensor
serial.write("M")
# We need  to calibrate sensor first, it takes about 15 seconds
serial.readline
# Loop
loop do
    # We have sensor calibrated, so we can get the data
    serial.write("M")
```

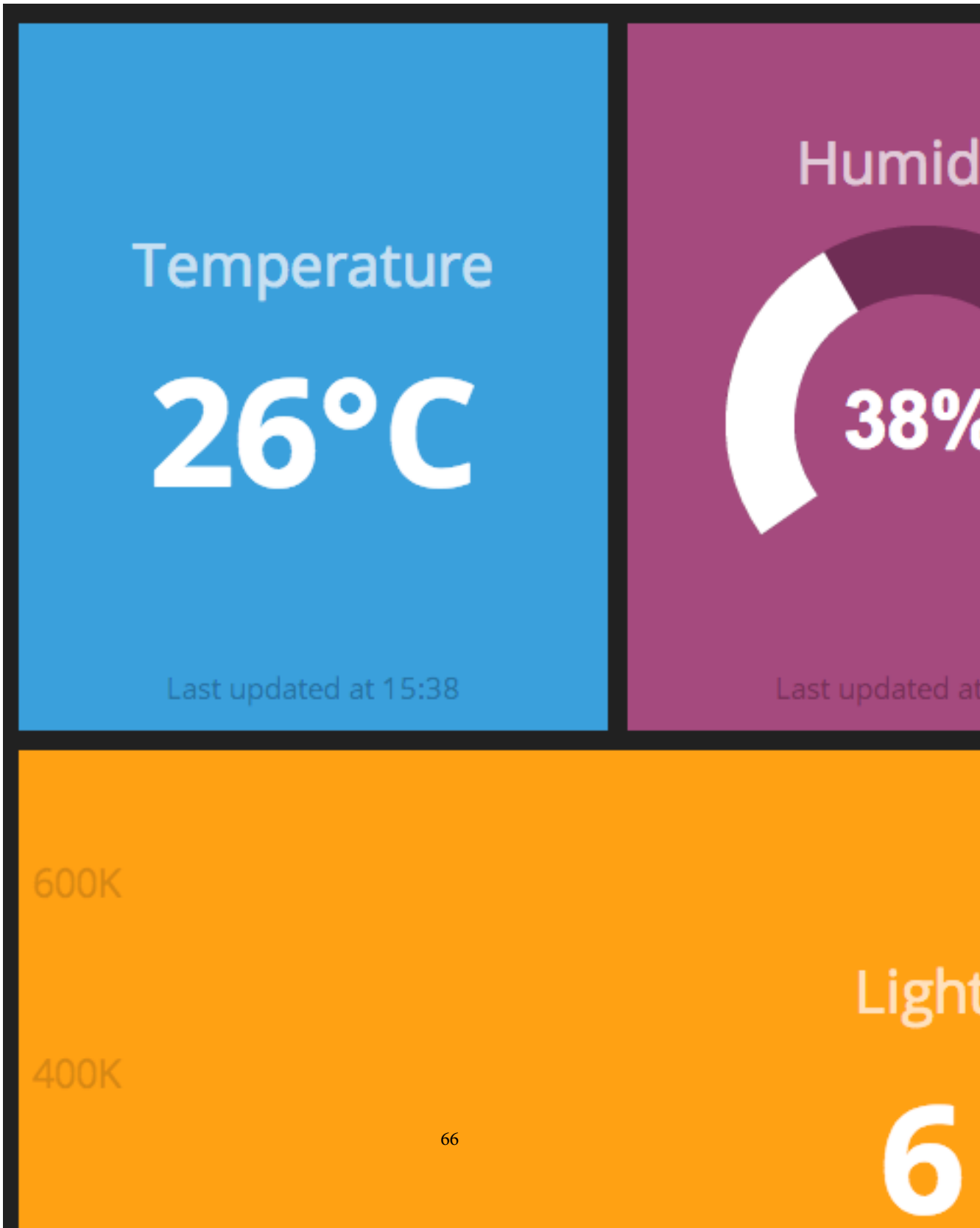
```
    # and assign it to a variable in integer value (0 or 1)
    motion = (serial.readline.strip).to_i
    # if value is equal 1, then...
    if motion == 1
        # inform us about it
        puts “MOTION DETECTED”
    end
# wait one second before the next cycle
sleep(1)
end
```

Now swing your hand in front of the sensor. In downloaded file, `5_alarm.rb` code is more interesting, dig into it!

ALL IN ONE PLACE

Now you know how to use all the sensors, you can use it to create a beautiful web dashboard with all the data.

To make it easier, we've prepared custom dashboard using Dashing. It's super easy to set up and host i.e. on Heroku. Our dashboard looks like this:


Temperature
26°C
Last updated at 15:38
Humid
38%
Last updated at
600K
400K
Light
6

To send data to this dashboard you need to know which ID to use. It should be written on a sticker on your Arduino. If you're not at Makerland, just use `arduino-X` where X is number from 1 to 20.

Now start Ruby script which sends data to the dashboard:

```
ruby 6_dashboard.rb YOUR_PORT ID
```

If there's no error, visit your dashboard at http://makerland-dashboard.herokuapp.com/ID

Dig into `6_dashboard.rb` to see how to send data to Dashing.

> Code for web dashboard is also in ZIP you've downloaded, so feel free to modify it and host wherever you want!

ADDITIONAL TASKS

- Add new commands to ArduinoSensors sketch, like wait certain period of time or read the state of button (ask us for a button ;))
- Open python interactive shell by typing `python` and play with serial live
- Play live with serial using ruby

ADDITIONAL RESOURCES

- https://github.com/meal/makerland-workshop

SOLDER WORKSHOP

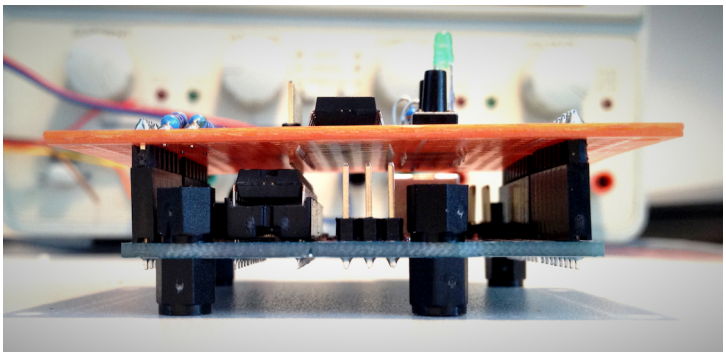


Figure 2: image

OVERVIEW

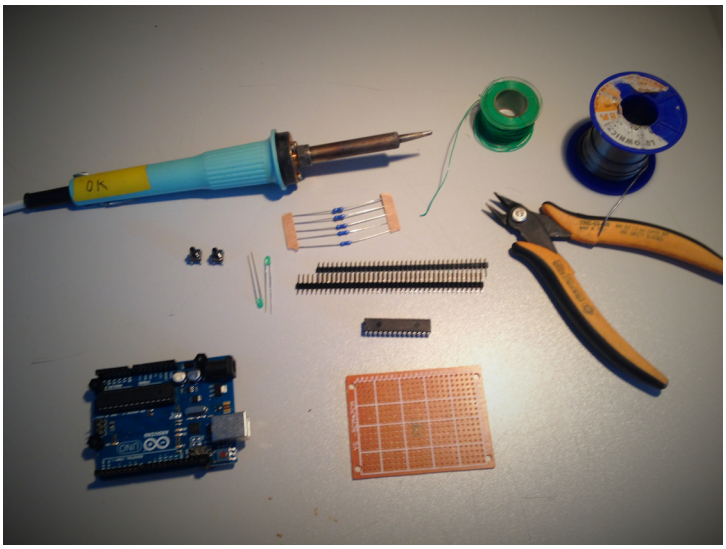
Imagine your Arduino or RaspberryPi. Imagine you are building your dream project and there comes a moment when you have to compromise with the device's functionalities because you have run out of ports in the microcontroller. During this workshop you'll solve your problem. Your task will be to create a board with a MCP23017 port expander. You'll think: a yet another breadboard and plenty of tangled wires. Far from it. During the next hour you'll create something lasting. You'll solder the expander and other elements onto the universal PCB board. And maybe you'll create a completely new shield for the Arduino Uno?

PREPARATIONS

In order to be able to focus on your work, you should prepare your workspace. You need: 1. a soldering iron 2. solder 3. side cutters 4. handles 5. a small bottle of spirit + a cloth 6. a universal board (PCB) 7. a MCP23017 port expander and other elements that your imagination dictates (LEDs, buttons etc). 8. the MCP23017 Datasheet

MAIN FLOW

Your task is to create a port expander for the Arduino Uno. The expander has 16 I/O ports. Add LEDs and buttons to 8 of them. You will be able to read the states of the buttons and control the LEDs through the Arduino Uno, which will communicate with the expander through the I2C bus. You should solder all the elements onto the PCB.



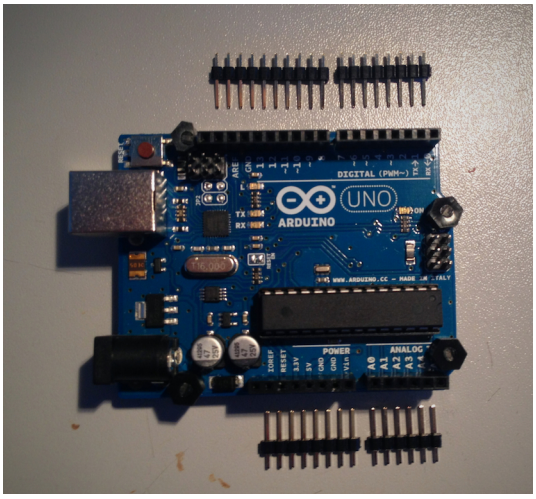


Figure 3: image

1. PLAN YOUR WORK.

If you don't know MCP23017, you must get acquainted with the datasheet. You'll learn from it about the capabilities of the circuit and the distribution of pins. Knowing the distribution of pins and the way of communicating with the expander, you should plan the arrangement of components: the expander, the connectors, LEDs and buttons. If you want to make a shield for the Arduino Uno, you must think of how to deploy the connectors by which you will connect your shield with the Arduino (go to **Additional Tasks**).

2. PREPARATION FOR SOLDERING.

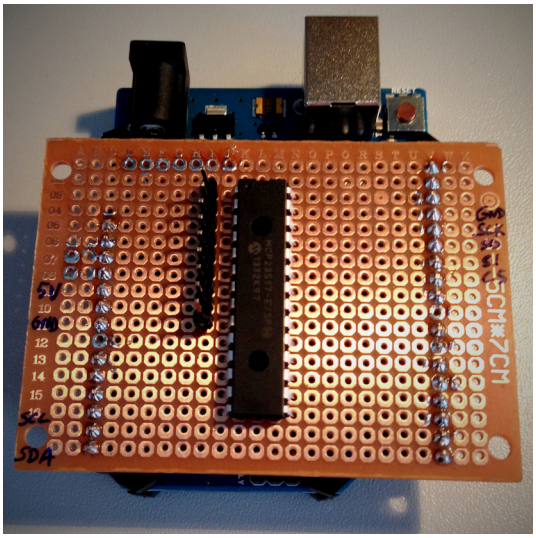


Figure 4: image

Put the first elements into the board. You should solder the elements from below. The most frequent mistake at this stage is to put in too many elements, which are going to fall out of the board when you turn it upside down for soldering.

3. SOLDERING.

Solder the following way: - apply the hot tip of the soldering iron to the leg of the element and the pad on the PCB - heat this spot up for about 3 seconds

- quickly apply solder and flux to the soldered elements. Don't apply solder directly to the iron!
- solder melts, flux removes oxides from metal surfaces, the elements are joined together with a joint
- withdraw the iron and solder
- solder solidifies to form a firm joint of the elements
- cut the ends of long legs (LEDs) right at the joint

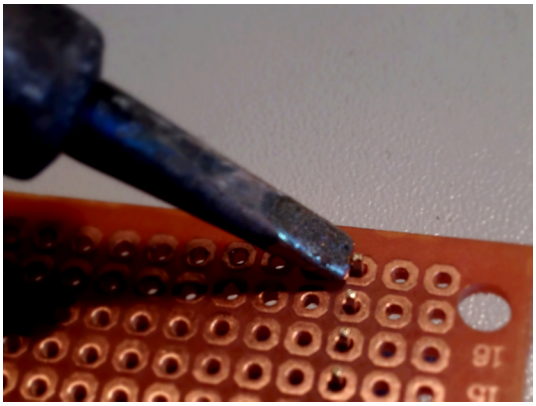


Figure 5: image

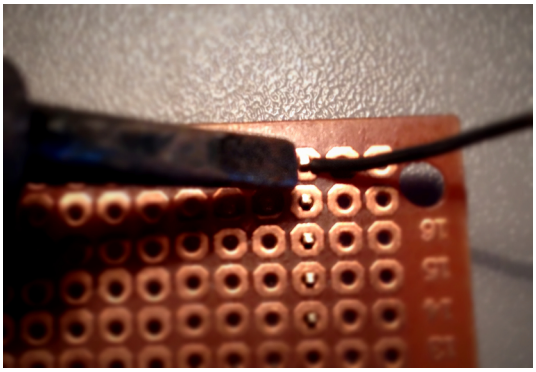


Figure 6: image

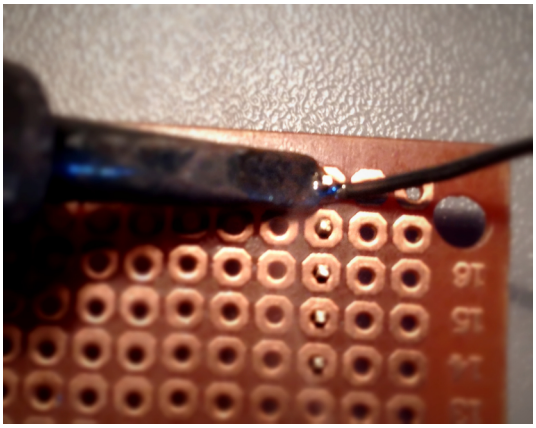


Figure 7: image

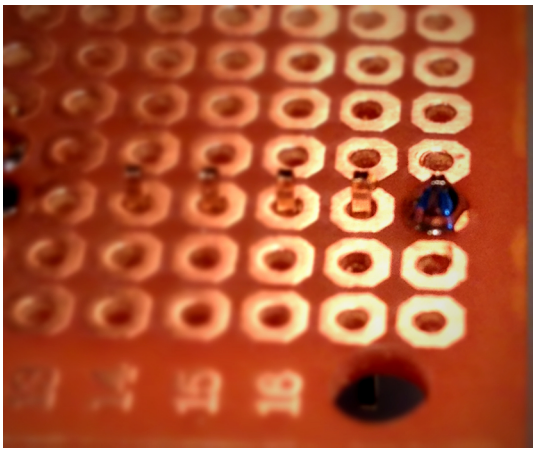


Figure 8: image

1. A well made solder should be shiny and have a nice shape. Holes in the joint or a matte color are signs of a poorly made soldering. The elements were probably dirty (dust or glue) or greasy. Before you start soldering, it's a good idea to clean the elements with a spirit-soaked cloth.
2. Not to overheat the components, try not to heat the legs of the elements up for too long. You'll come up with an optimum time as you practice.
3. Clothing from artificial fabrics electrifies us. An electric charge accumulates on the surface of our bodies. It might even be a few kilovolts relative to the ground. If you touch an element (the expander or a LED) while electrified, there will be an electrostatic discharge and you will damage it. In order to prevent this, wear an antistatic wristband during soldering. If this is not possible, touch a grounded piece in your hereabouts (e.g. the railing of the stairs) before you start.
4. Flux contained in the solder is designed to remove oxides from the surface of the soldered metal. Oxides are formed through the reaction of metal with air. They can cause the so-called *cold joint*. The elements are mechanically connected but do not conduct electricity. Therefore apply solder to the hot parts but not to the tip of the soldering iron. The soldering iron is always hotter and will cause the flux to evaporate before it works.
5. Put the soldered elements into the PCB and then turn it upside down for soldering. If you do not want such items as a resistor or a LED to fall out, you can gently bend their legs out when you insert them into the board.

ADDITIONAL TASKS

Do you feel good with your skills? If so let's get to the next step if not keep on experimenting or ask us for some help.

1. To connect the Arduino Uno with your PCB as you would do with a shield, you have to:

- plan the location of connectors through which you will connect to the Arduino Uno board. You should push your shield into the Arduino Uno.
- connect the MCP23017 expander with the appropriate pin connectors to the Arduino Uno. Pay attention to:

Figure 9: image

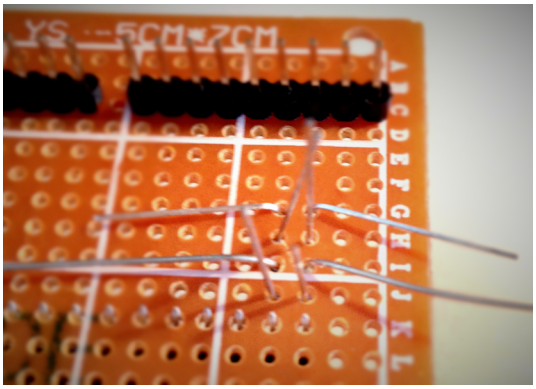


Figure 10: image

power supply and the I2C bus.

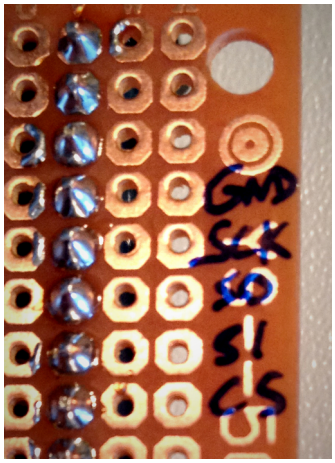


Figure 11: image

2. A port expander is used to ensure that your microprocessor system has more available ports than a microcontroller does. It is not only LEDs and buttons that you can connect to the system plugs. You can connect any other element which works with the digital (two-state) port of the microcontroller. Through the expander you can control a transistor, a buzzer, a tilt sensor, a servo driver. You can even connect an LCD display. You can get out of the box!

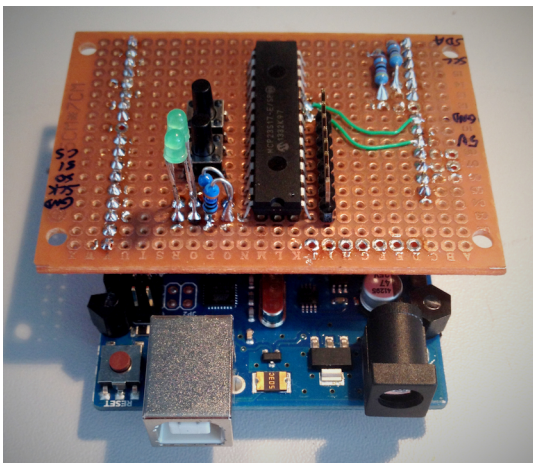


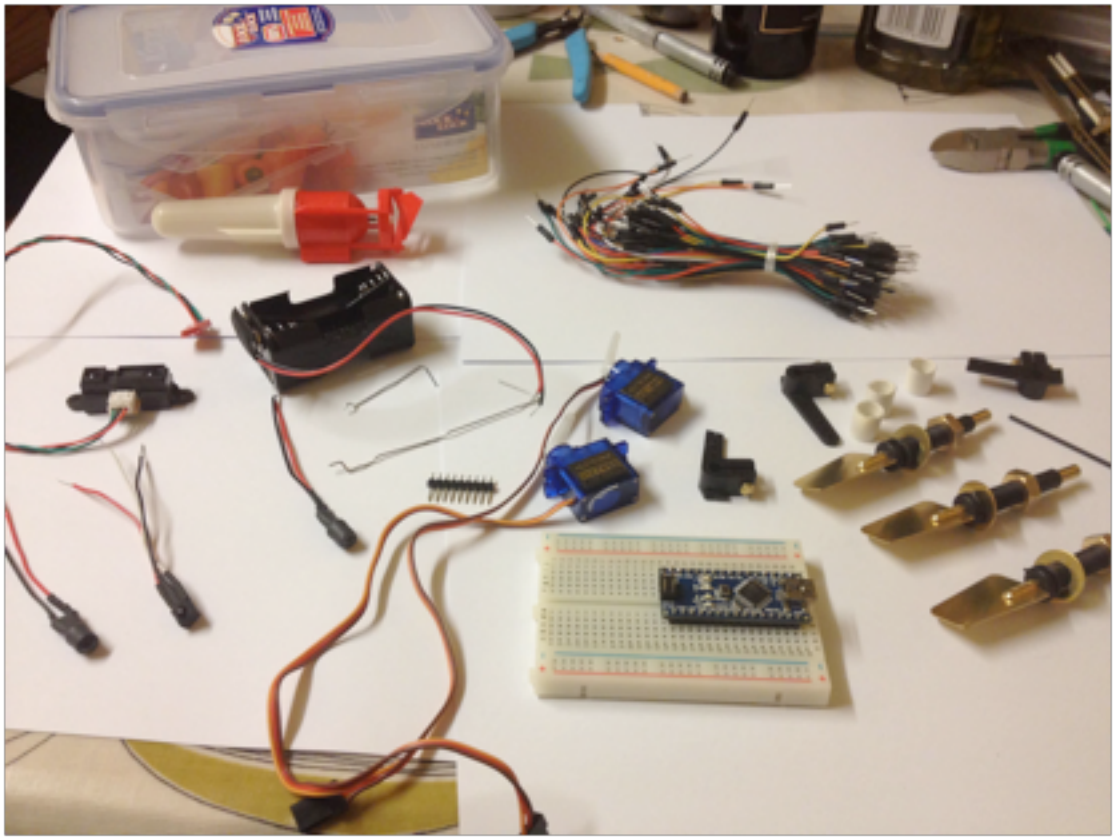
Figure 12: image

ADDITIONAL SOURCES

Want to learn from masters? Check this links below.

http://www.youtube.com/watch?v=vIT4ra6Mo0s http://www.aaroncake.net/electronics/solder.htm http://www.sciencebuddies.o fair-projects/project_ideas/Elec_primer-solder.shtml http://www.wikihow.com/Solder-(Electronics) # Autonomous AUV Workshop

REQUIREMENTS



1. 1 x 1litre Lock & lock container pre drilled
2. 1 x Arduino Nano V3.0
3. 2 x Micro servo motors with horns
4. 3 x Micro rudder assemblies
5. 1 x submarine motor
6. 1 x 400 breadboard
7. Selection Male to Male jumper leads

8. 1 x IDE header 9 pins
9. 2 x Large paper clips
10. 3 x Plastic spacers
11. 2 x Vishay IR LED's TSAL6400
12. 1 x Vishay IR detector TSOP4838
13. 1 x Sharp GP2D12 distance sensor with connector
14. 1 x thin rod 2mm diameter
15. 1 x Battery pack holder 4AA batteries
16. Bags of coins for ballast

CONSUMABLES

1. Silicon sealant
2. Coloured wires 0.7mm single strand
3. Heat shrink
4. Plasticine
5. Double sided tape
6. AA batteries (5 per AUV)

TOOLS / EQUIPMENT

1. Laptops with Arduino IDE installed
2. Mini-USB cables to load sketches onto Arduino Nano
3. Servo motor tester
4. Small flat screw driver
5. Snips
6. Wire strippers
7. 13mm Spanner
8. Multimeter

INTRODUCTION

In this workshop we are going to build an AUV, the main body is a Lock & Lock container normally usually used for keeping food in. Propulsion to drive our vehicle will be provided by a submarine motor capable of being submerged to shallow depths. An Arduino Nano will control navigation. Dive planes will be used to regulate depth and a rudder for turning. Infrared sensors will act as the eyes of the AUV and will be used for obstacle avoidance and bottom of the pool detection.

THE BUILD

The main body of the AUV is going to use a Lock & Lock container normally used for storing food. It has a built in seal and is both airtight and watertight. Three holes have been pre-drilled into the container for installation of the dive planes and rudder. The dive planes allow the AUV to submerge very much like the fins on a shark and the rudder controls the direction of travel.

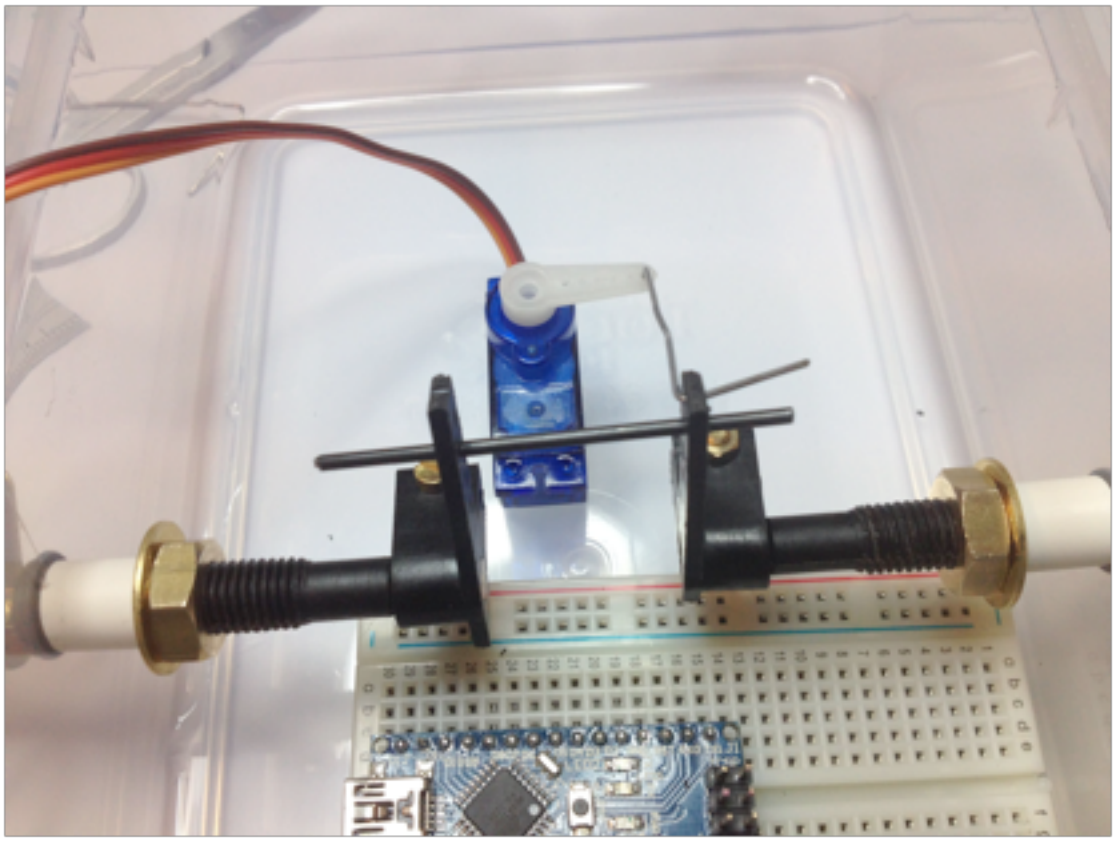
Lets get started by assembling the servo motors, dive planes and rudder this will also provide us with our first opportunity to test how waterproof our system is.

ASSEMBLING SERVO MOTORS, DIVE PLANES AND RUDDER

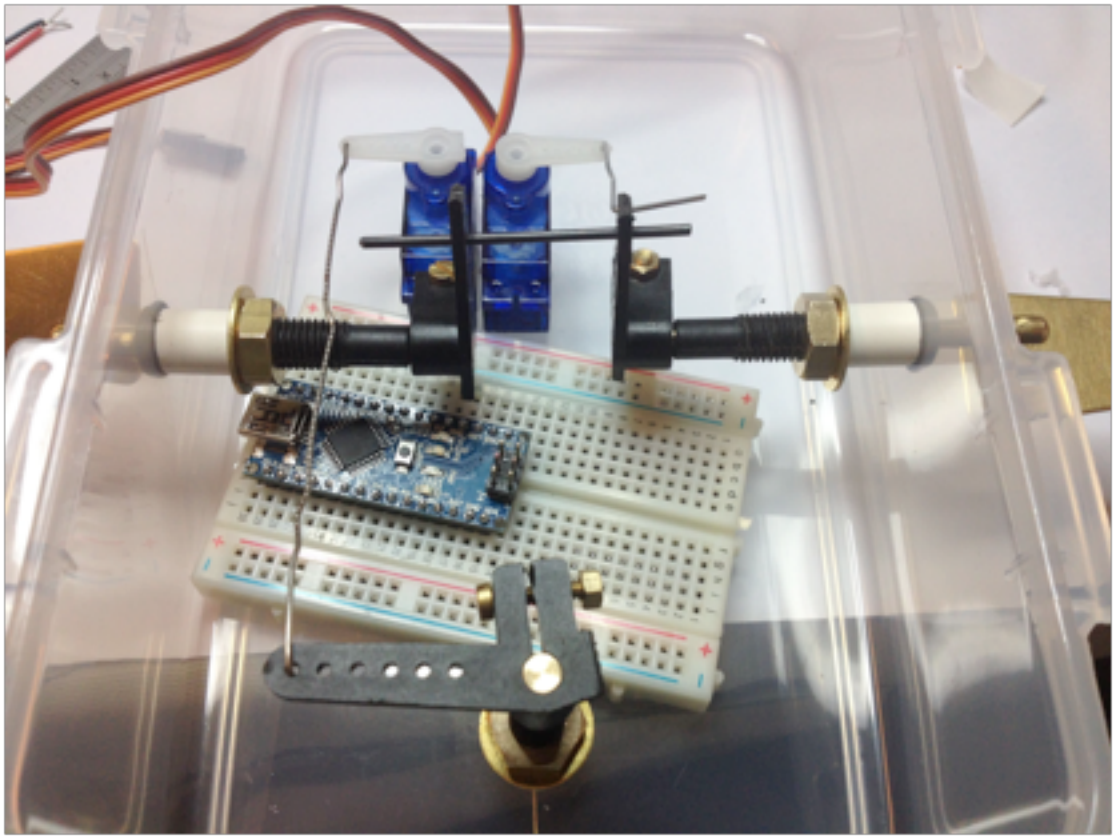
1. Assemble dive planes / rudder into container with provided spacers, washers and 13mm nuts. Take care not to over tighten the nut as this will distort the O-ring and be a possible point of water entry.

2. At this stage it is important to test for leaks. Place the lid on the container and gently lower the complete assembly into the pool. Check carefully for any leaks of water. If leaks are found they can be cured by injecting a small amount of silicone sealant into the end of the rudder shafts. After adding any silicone sealant repeat the test for water tightness. Keep testing until you have a watertight container.

3. Attach the dive plane horns to the dive plane shafts.
4. Line up the dive planes parallel to the bottom of the AUV.
5. Centre servo body and stick down with double sided tape.
6. Centre servo at 90 degrees using provided servo motor tester.
7. Attach horn to servo, see picture below for orientation.
8. Add link between servo and dive planes using a paperclip.
9. Tighten screws on dive planes.
10. Connect dive planes together with short metal rod.



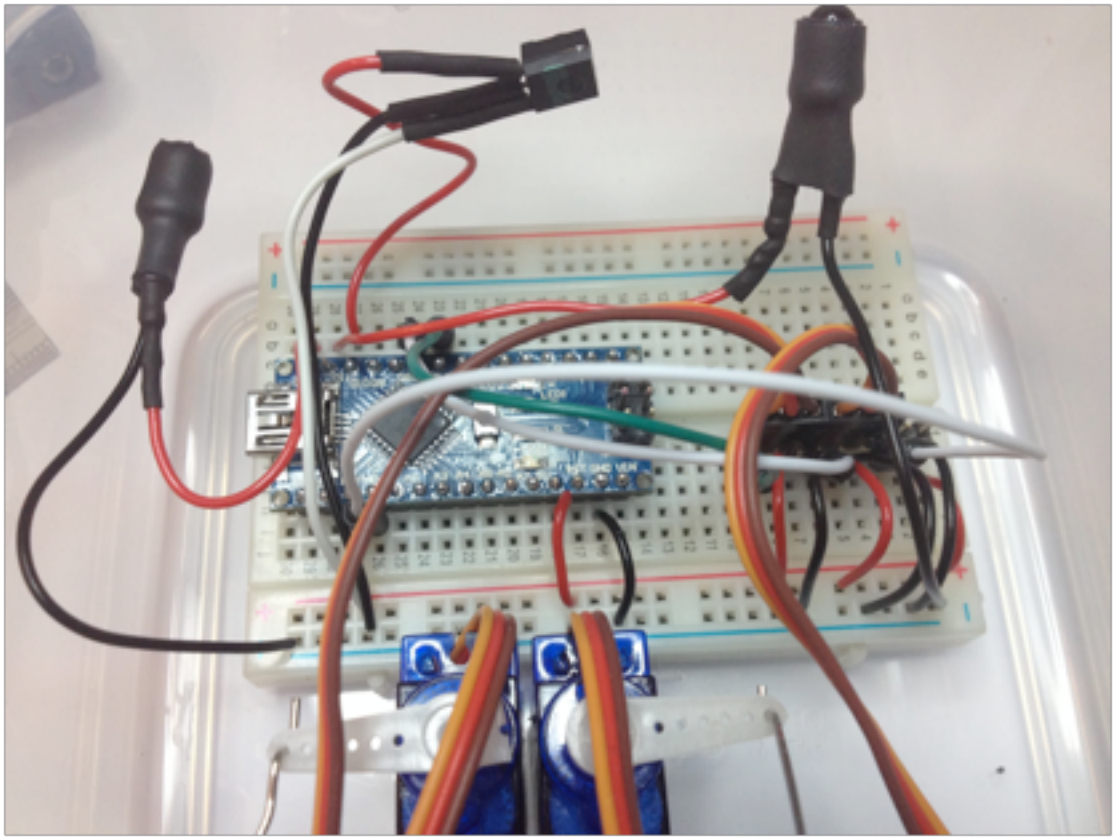
11. Test movement of dive planes using servo motor tester.
12. Install rudder into AUV body using spacer, washer and nut.
13. Attach rudder horn to rudder shaft.
14. Centre servo at 90 degrees using servo tester.
15. Fit servo horn to servo, see picture below for orientation.
16. Add link between servo and rudder assembly using paperclip.
17. Test rudder assembly using servo tester.



Now that we have the mechanical components assembled we can move onto adding the electronics components. Starting with two IR LED's and IR receiver. These are used by the AUV for obstacle avoidance. Each IR LED is switched on and off alternately by the Arduino Nano at a frequency of 38khz, the same as used in a television remote control. The IR receiver can detect infrared light at a frequency of 38khz so will detect any light reflected back from an obstacle in the path of the AUV. The Arduino sketch recognises which LED is switched 'on' so can take action to avoid the obstacle.

CONNECTING IR LED'S AND RECEIVER

1. Place breadboard with the Arduino Nano installed at the front of the AUV
2. Connect IR LED's and IR receiver to the breadboard. See picture below and circuit diagram.

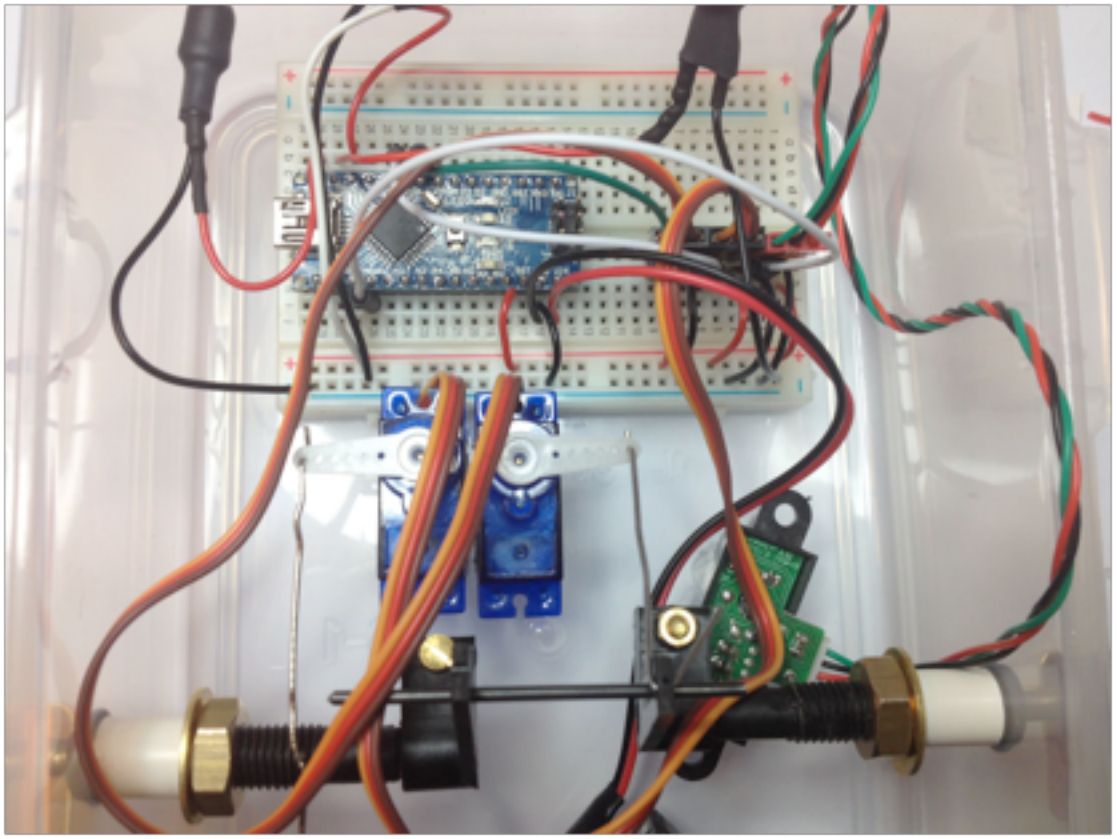


3. Connect the two servos to the breadboard using the piece of IDE header

We are next going to add a sensor to detect when the AUV is near the bottom of the pool. We are going to use infrared again but as we want a more accurate measurement we are going to use a Sharp GP2D12 distance sensor, the sensor can accurately measure distances between 10 and 80 cm.

CONNECTING UP THE SHARP GP2D12

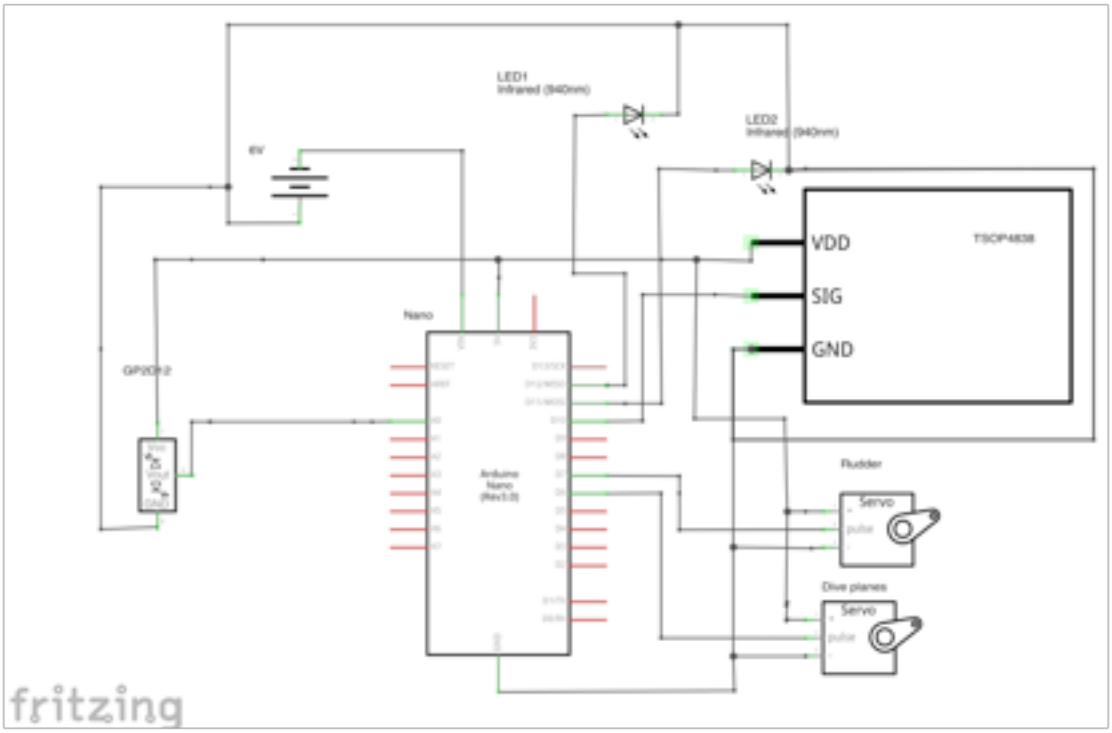
Connect the sharp GP2D12 sensor to the breadboard using the remaining three pins on the IDE header and use the male jumpers to connect to the Arduino Nano. Use the circuit diagram from below for reference. With the distance sensor wired up we can now move onto fixing the IR LED's, IR receiver and the Sharp GP2D12 to the box for this we are going to use plasticine which is readily mouldable but adheres nicely to the plastic.

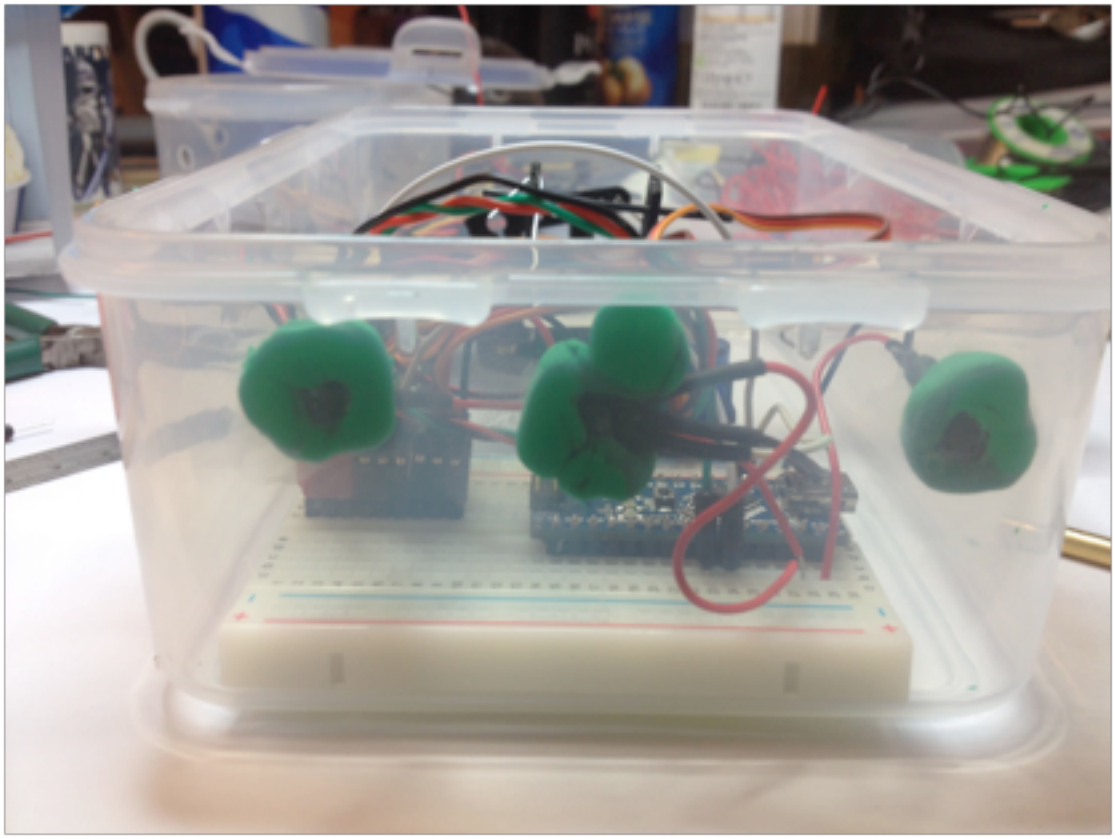


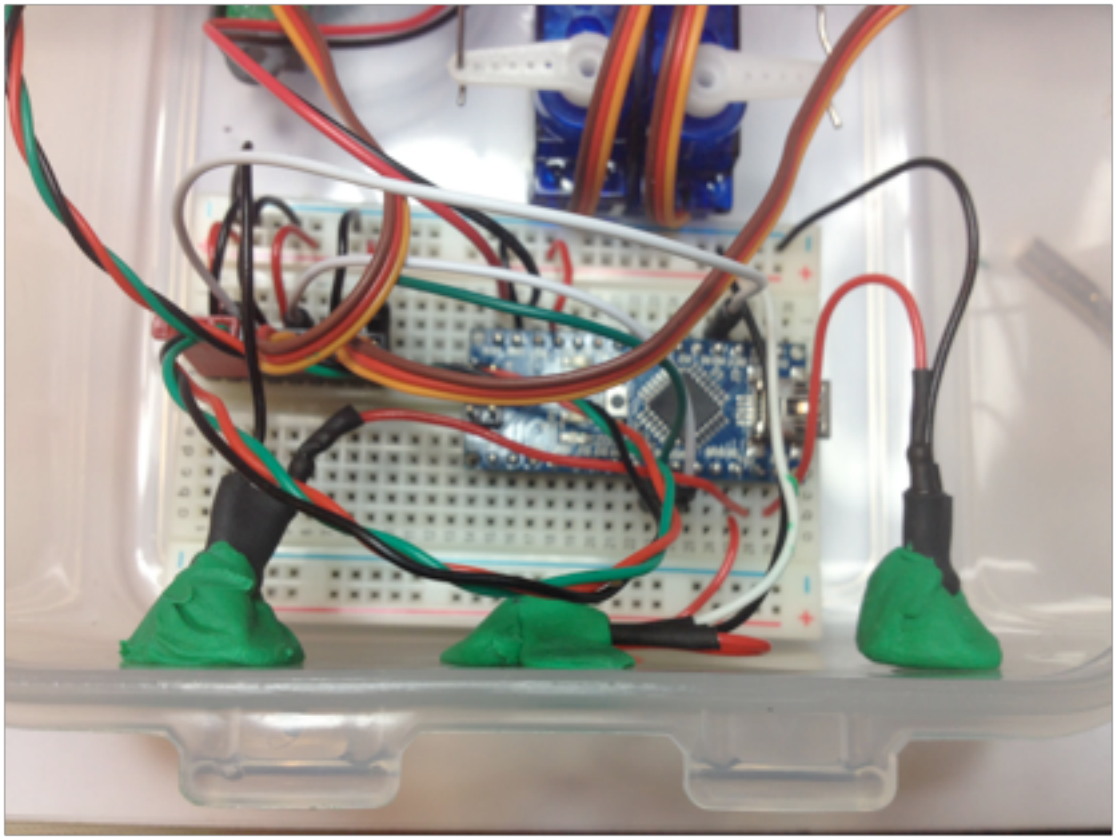
AUV SCHEMATIC

ATTACHING THE IR SENSORS

1. Take a small piece of plasticine and mould it round the lens of each LED leaving a hole at the front then press onto the plastic container.
2. Do the same for the IR receiver


LED1
Infrared (940nm)
LED2
Infrared (940nm)
9V
VDD
SIG
GND
TSOP4838
Nano
Arduino
Nano
(Rev3.0)
Rudder
Servo
Dive planes
Servo
fritzing



3. 
4. Similarly attach the Sharp GP2D12 to the bottom of the AUV

The last thing we need to add to the inside of the AUV is the battery pack this connects to the Vin and GND of the Arduino Nano.

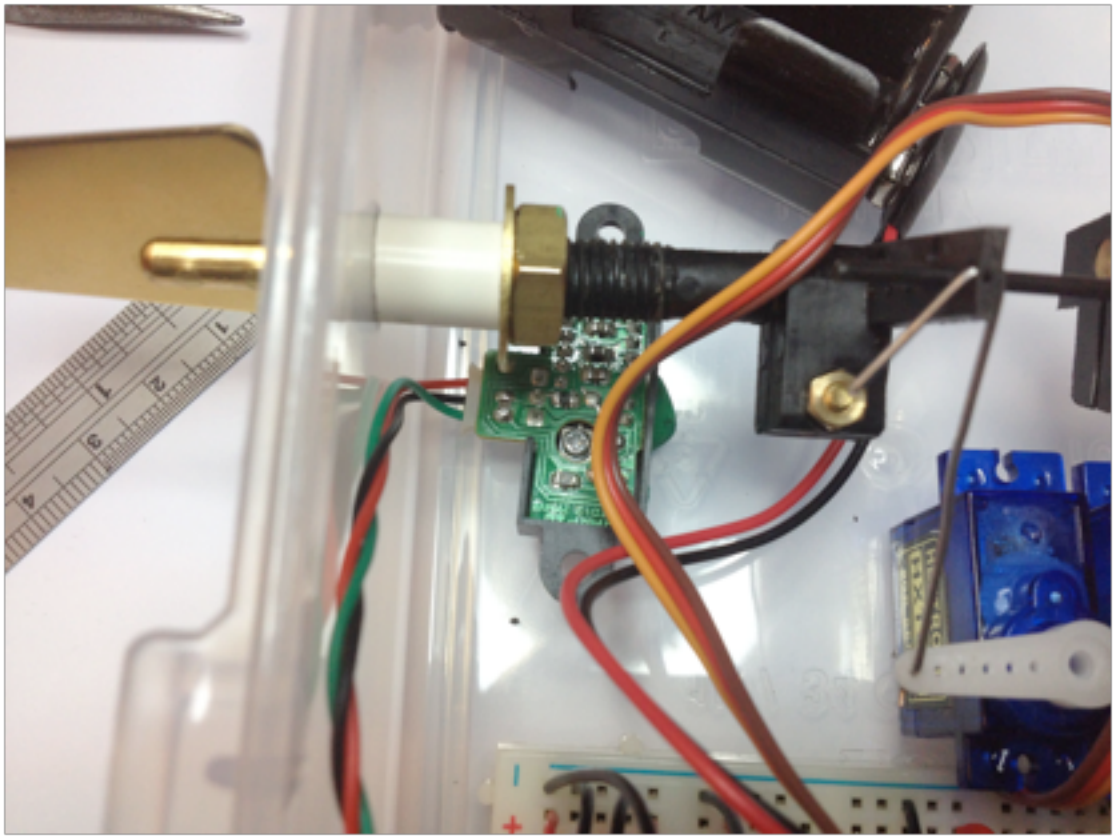
INSTALLING THE BATTERY PACK

1. Install the battery pack into the AUV.
2. The battery pack provides 6V and connects to the Vin and GND of the Arduino Nano.

We now need to check the operation of the servo motors and sensors.

TESTING

1. Use your hand as an obstacle place it in front of the forward facing IR sensors and check that the rudder moves appropriately
2. Place your hand underneath the AUV and note that the dive planes should move in relation to how far your hand is away.
3. If the servos don't move as expected then check all your wiring and repeat the tests until every thing is working.
4. When everything is working disconnect the battery pack and proceed to the next stage

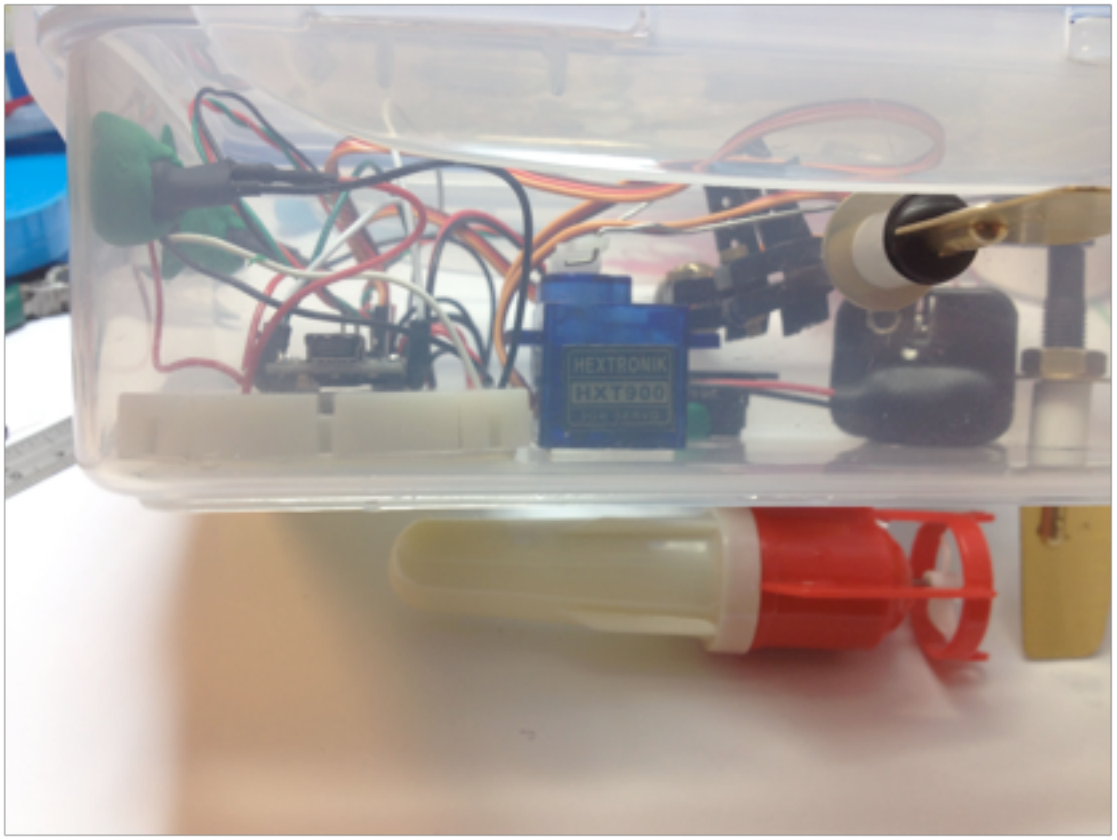


That completes the internal electronic and mechanical components of the AUV. We want to make a quick check that everything is still watertight. Pop on the lid of the AUV and gently submerge it into the pool do a visual check for leaks and then take it out and dry it.

To make the AUV move through the water we are going to use a submarine motor as used on a toy submarine. The motor sticks on the bottom of the AUV using a suction cap and is powered by its own internal AA battery.

PROPULSION

1. Attach the rubber suction cap to the submarine motor.
2. Stick it to the bottom of the AUV lining up the propeller with the rudder.



That is the main parts of the AUV completed but if you place it into the pool you will notice it is nicely buoyant and will float quite happily on top of the water. We want a vehicle that can go underwater so we need to add ballast to make it very slightly positively buoyant.

ADDING BALLAST

1. Add small bags of coins to the bottom of the inside of the AUV

2. Make sure the coin bags won't impede any of the movements of the rudder or dive planes.
3. Try to keep the weight spread out so when the AUV is in the water it will be level.
4. After adding some weight try putting the AUV into the pool and see how buoyant it is.
5. Keep repeating adding or subtracting weight until the AUV is just positively buoyant and nicely trimmed in the pool

You can tell when the AUV is just positively buoyant by giving it a gentle push on the top with your finger; the AUV should sink a little way and then slowly float back to the surface.

That completes the build process it's now time to try the AUV on its maiden voyage.

MAIDEN VOYAGE

1. Connect internal battery pack to the breadboard
2. Check everything is working properly
3. Switch on submarine motor
4. Place into the pool and let go.

Congratulations you have built your underwater AUV, it should be moving around the pool avoiding obstacles and slowly moving up and down in the water.

REFERENCES

1. How to use infrared receiver sensors for collision avoidance http://letsmakerobots.com/node/29634
2. Arduino http://arduino.cc
3. Fritzing http://fritzing.org