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.

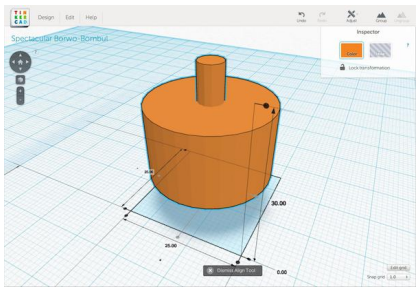
5. 3D MODELLING: THE BASICS

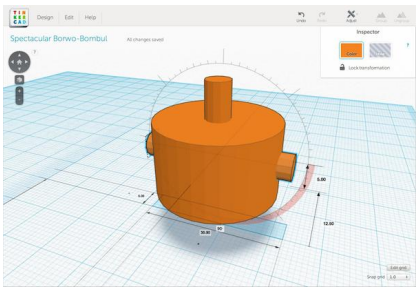
Ok, so we have a cylinder with the dimensions set in the previous step. Now lets get one more cylinder to the workplane.

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.

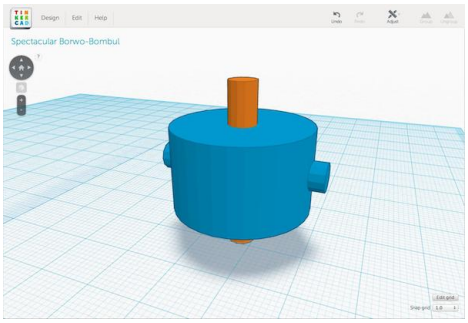


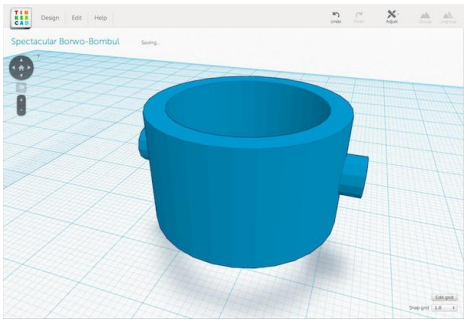


5. Select the horizontal thin cylinder and the big one and group* them together using either keyboard shortcut or command from the top menu bar.

> 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/

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
```

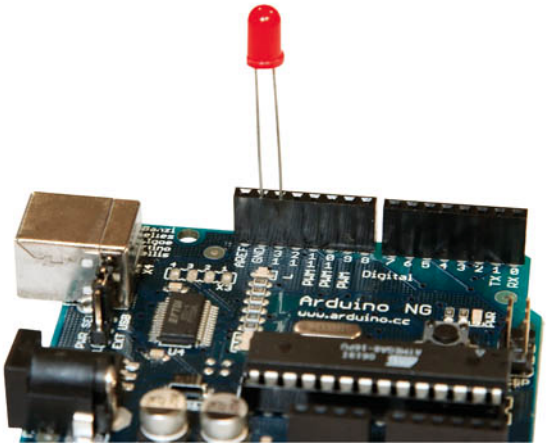
That's it! The program may instruct you to open a web browser to http://localhost:8057/ if it couldn't do so for you automatically. You are now ready to begin the tutorial. Remember, if you run into any trouble, just ask one of the workshop assistants and we'll help you as much as you require!

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!

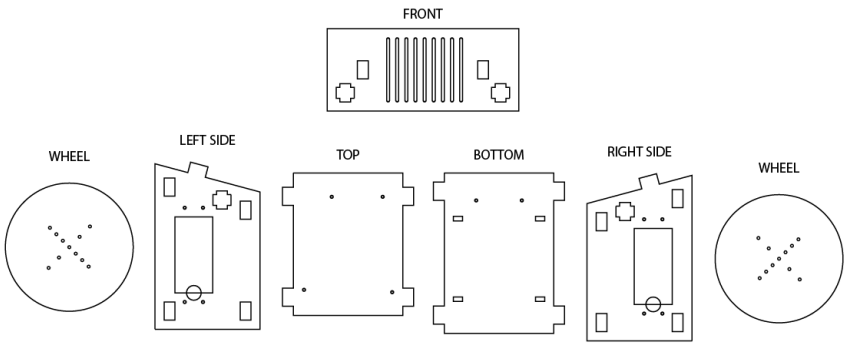
Didn't work? Uh oh. Make sure:

- 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!

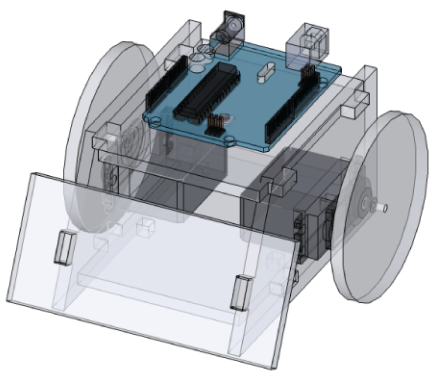
QUEST 2: CONTROLLING A ROBOT

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.

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!

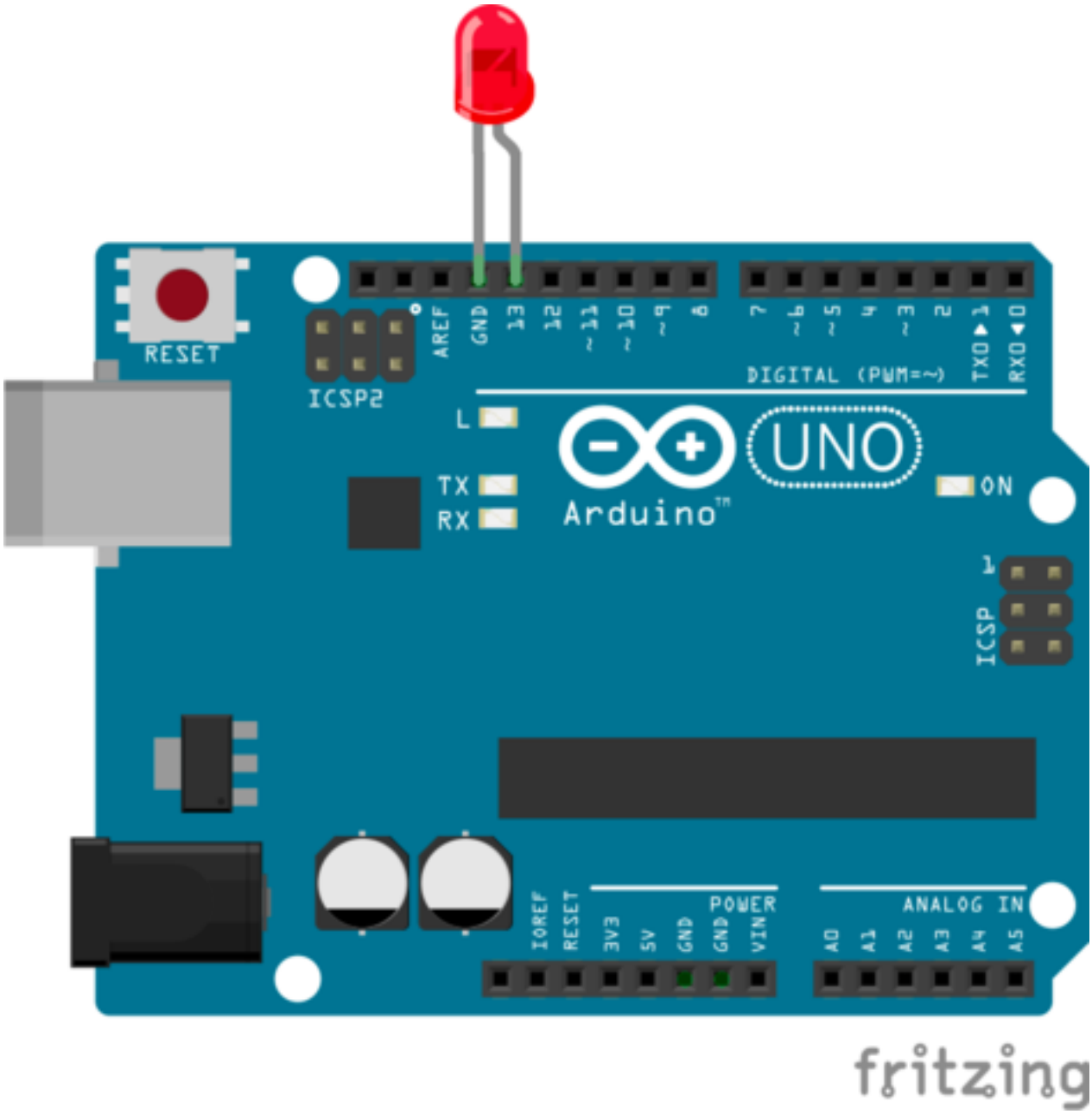
STEP 2: CONNECT AND TEST A SERVO

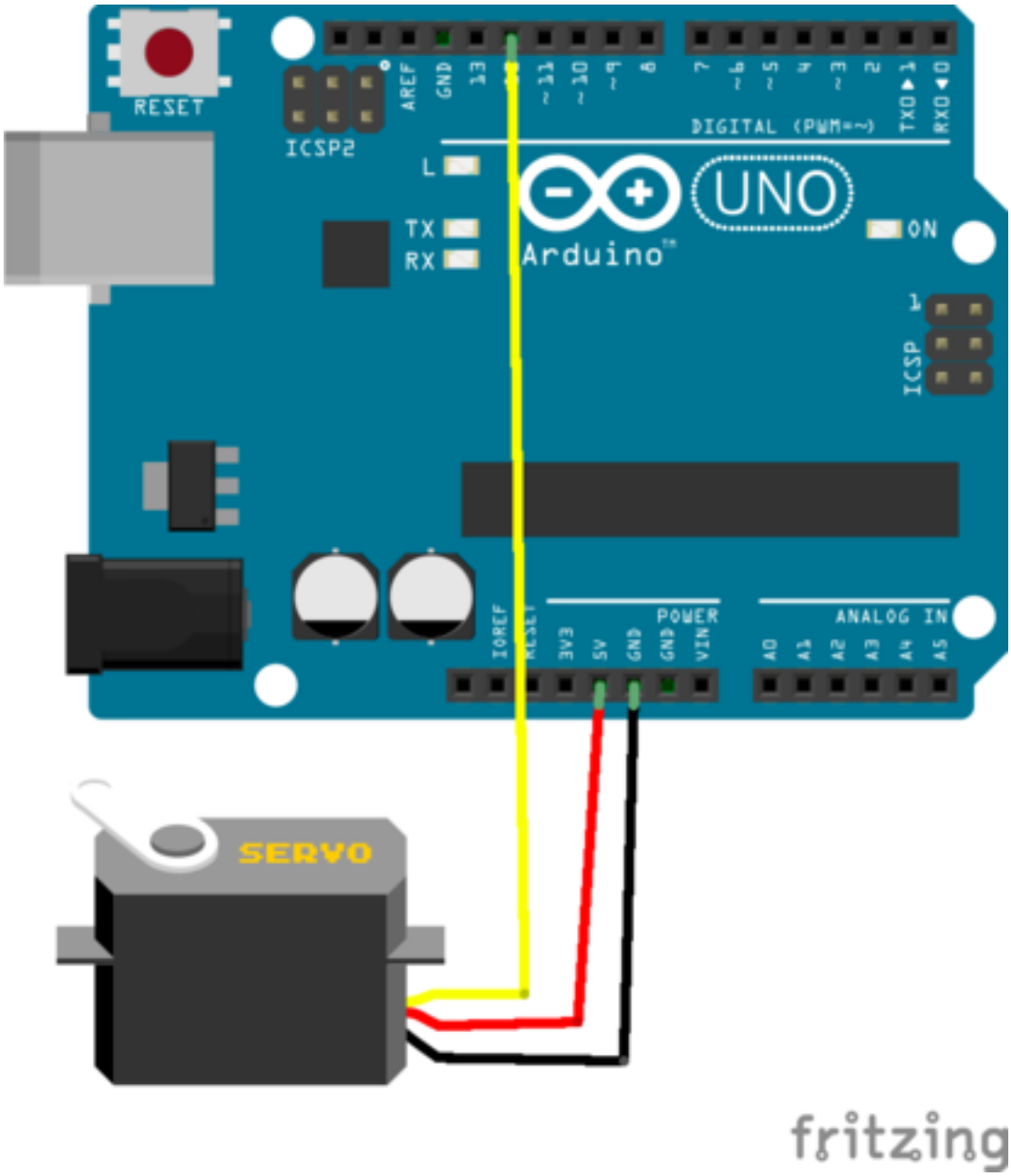
Now that we are sure that you can program your Arduino, we will move on to the fun stuff and connect our servo to our board.

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
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
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.

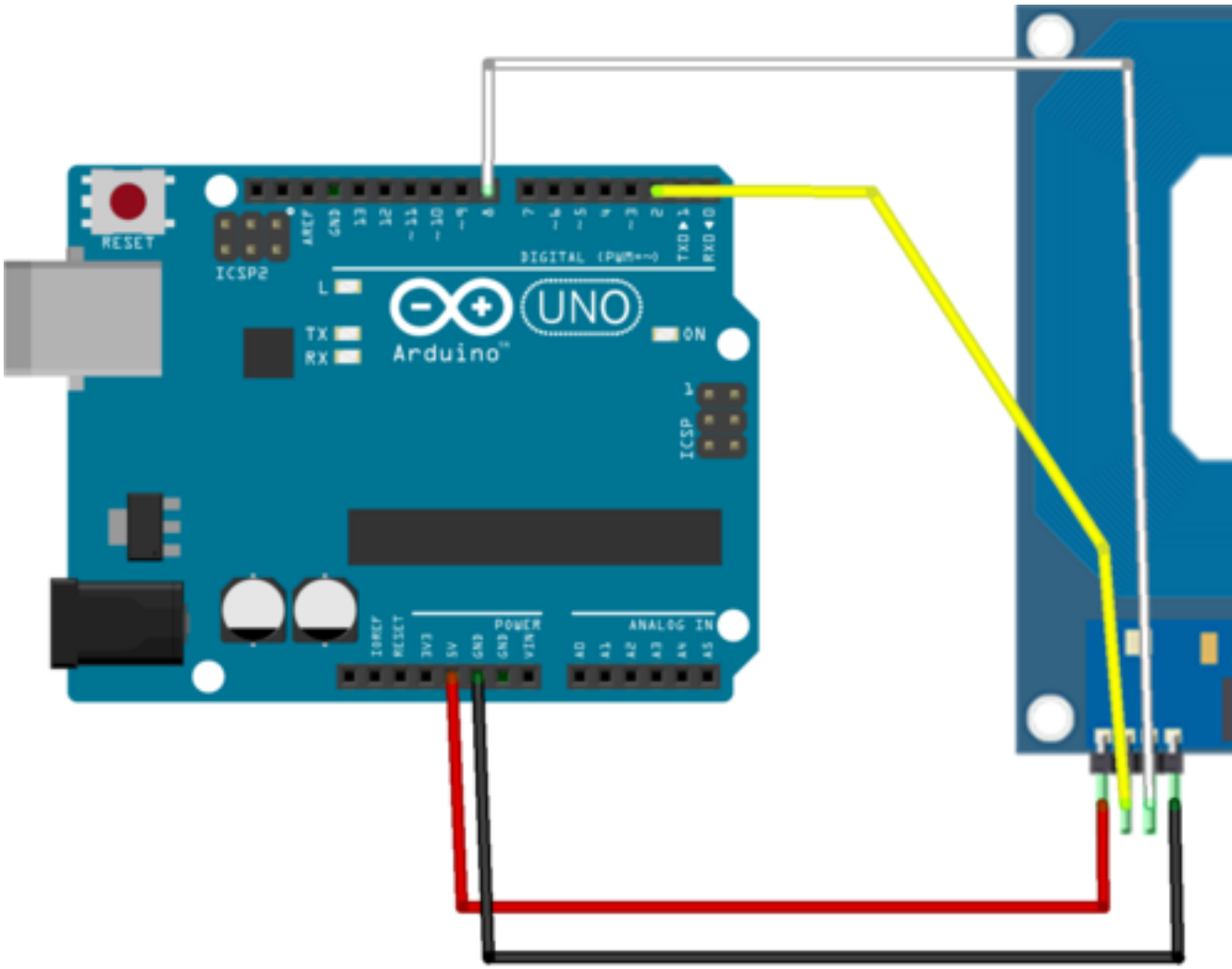
Once you are comfortable with controlling your Servo, we can move on. Additional examples are available here: http://arduino.cc/en/Tutorial/Sweep

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!

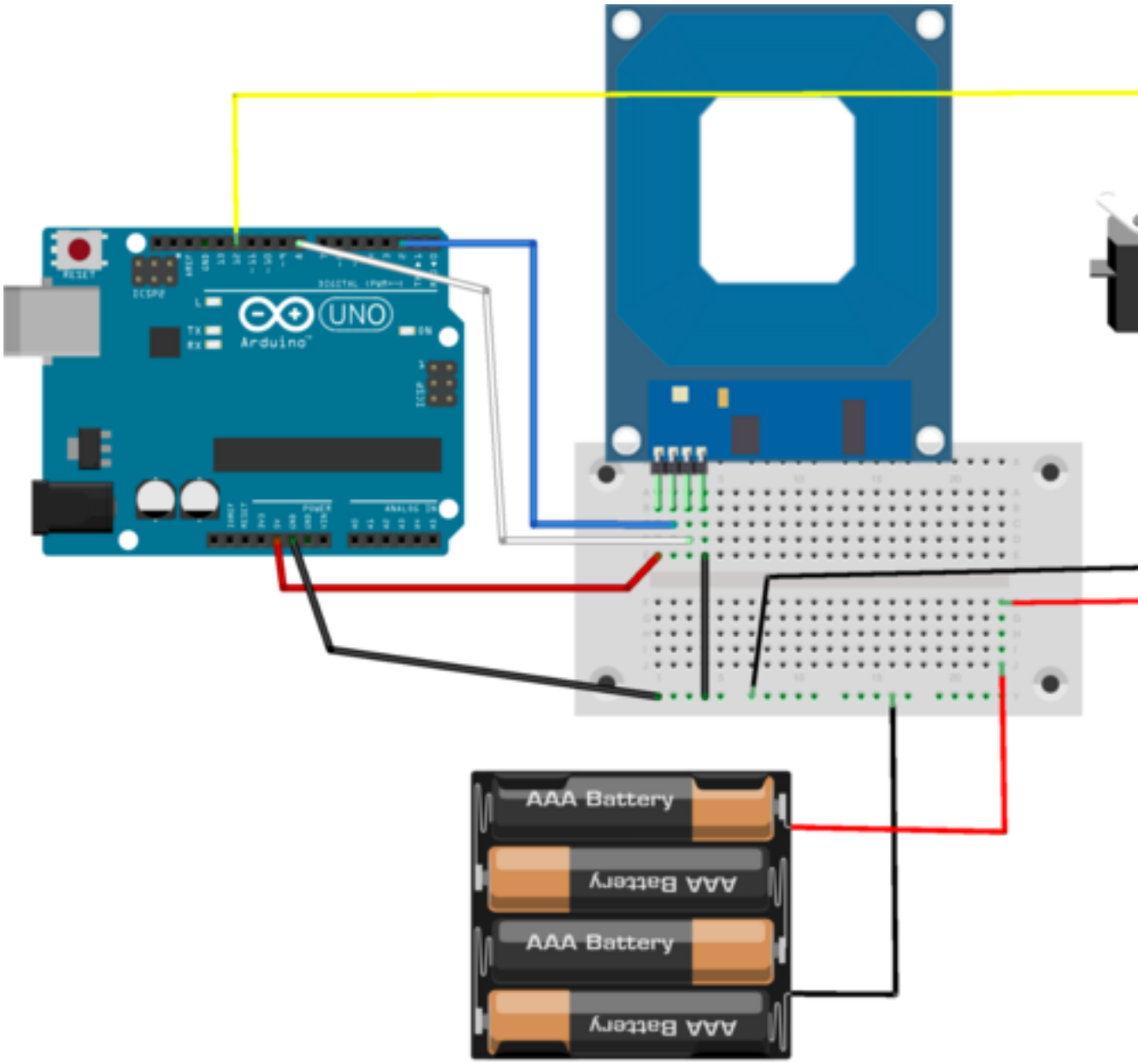
Attach the two metal rods to your servo with screws and washers.

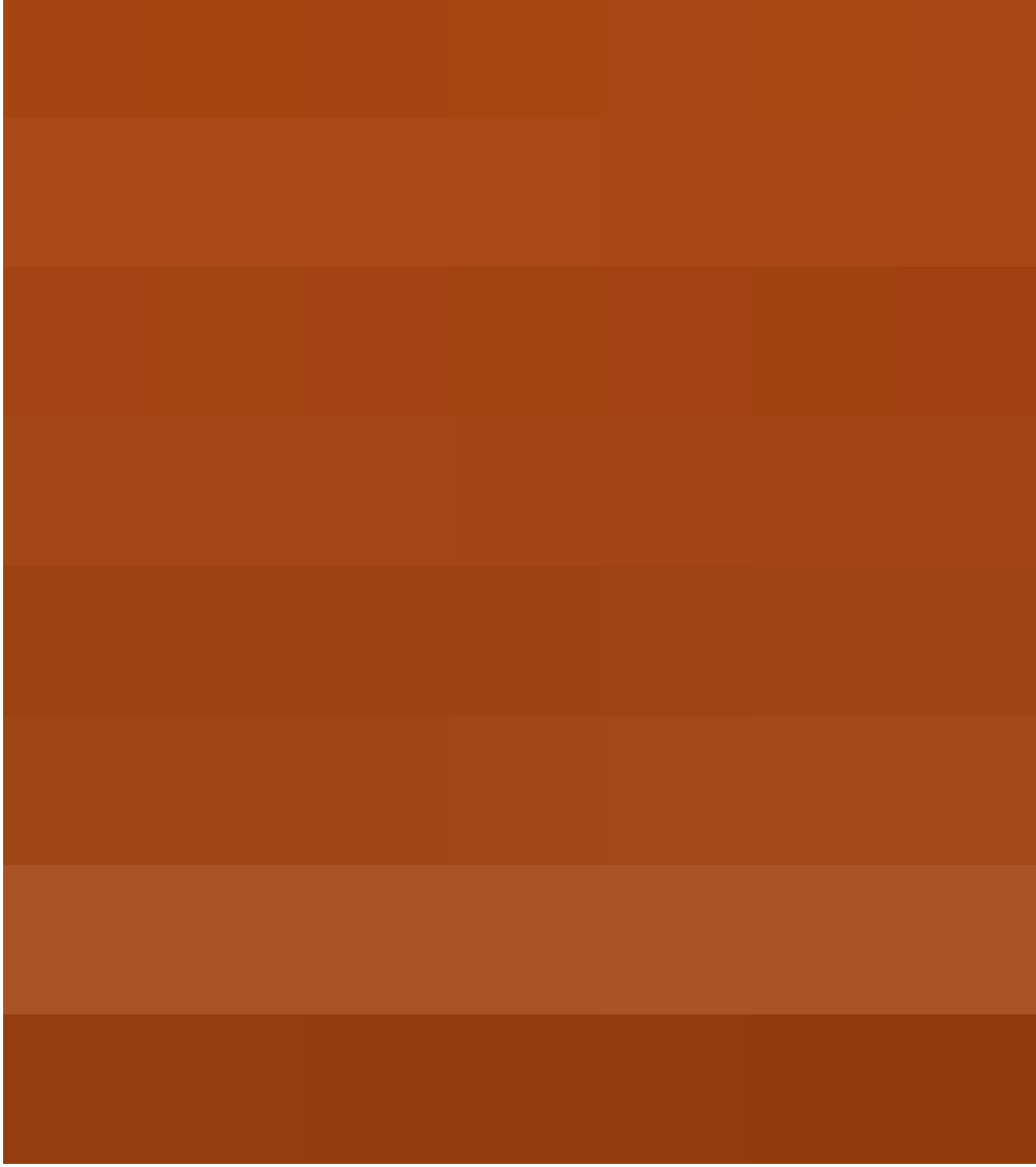
Now use a piece of cardboard (or other stiff material) to make a tighter bond between the servo and the lock:

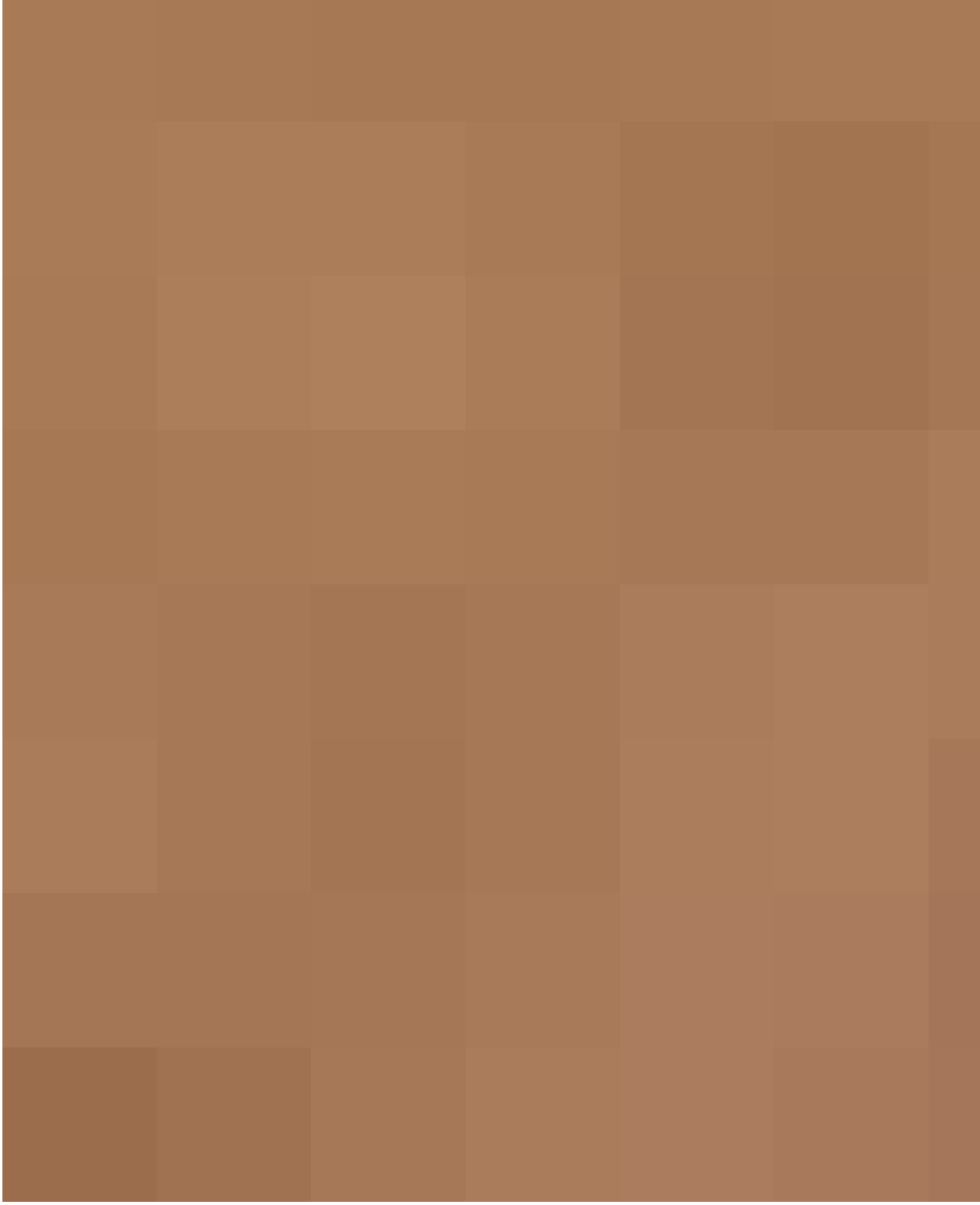
Now you can tape the servo to the deadbolt lock. Make sure it is positioned on the correct side so that the direction that the servo turns in is aligned with the direction that the lock turns in:

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
```

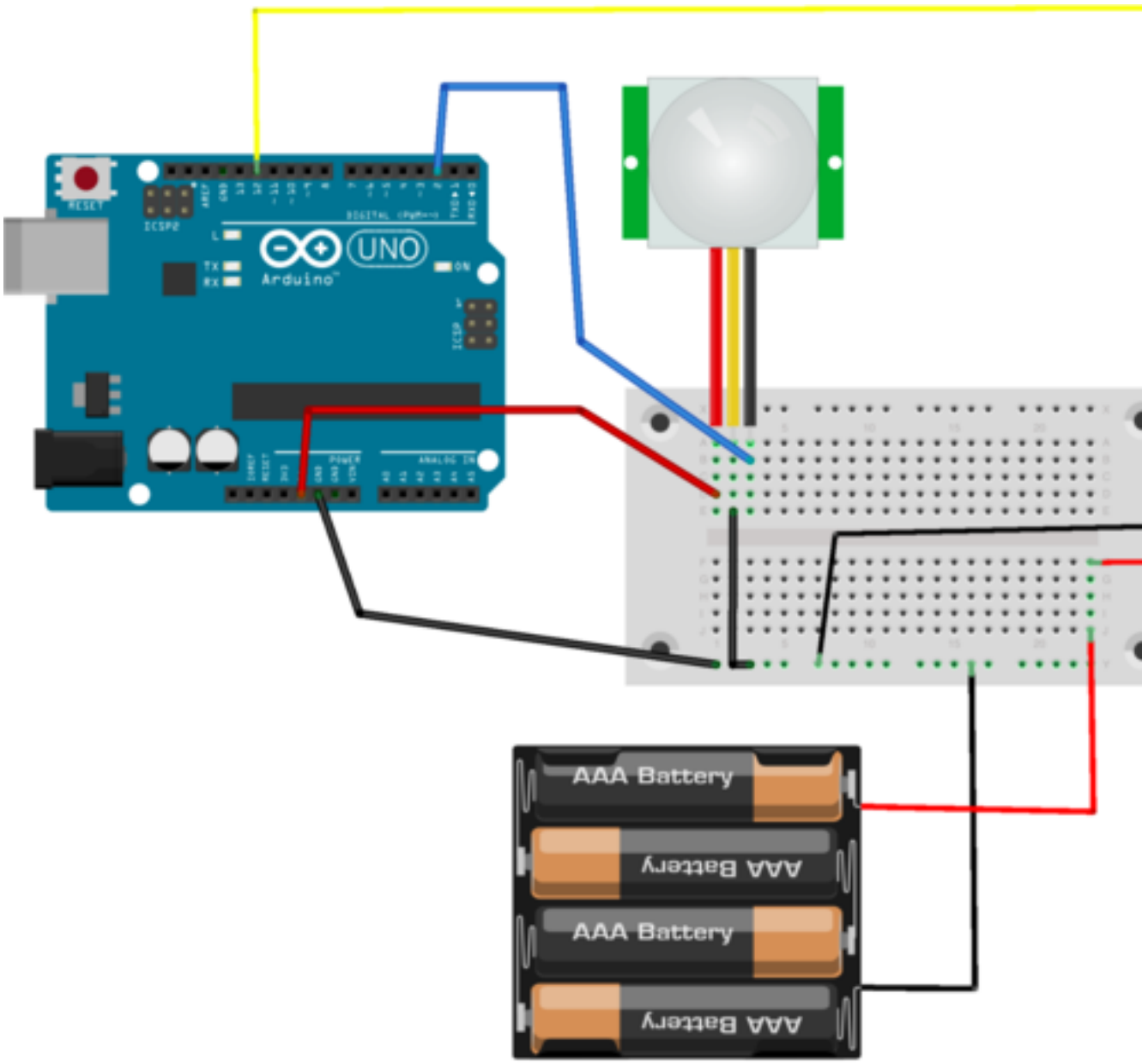
Node-Serialport makes it easy to communicate with your Arduino via a Serial connection from a Node.js program. We will be using it to receive HTTP request for incoming text messages from Twilio, and passing instructions along to the Arduino to lock or unlock your deadbolt.

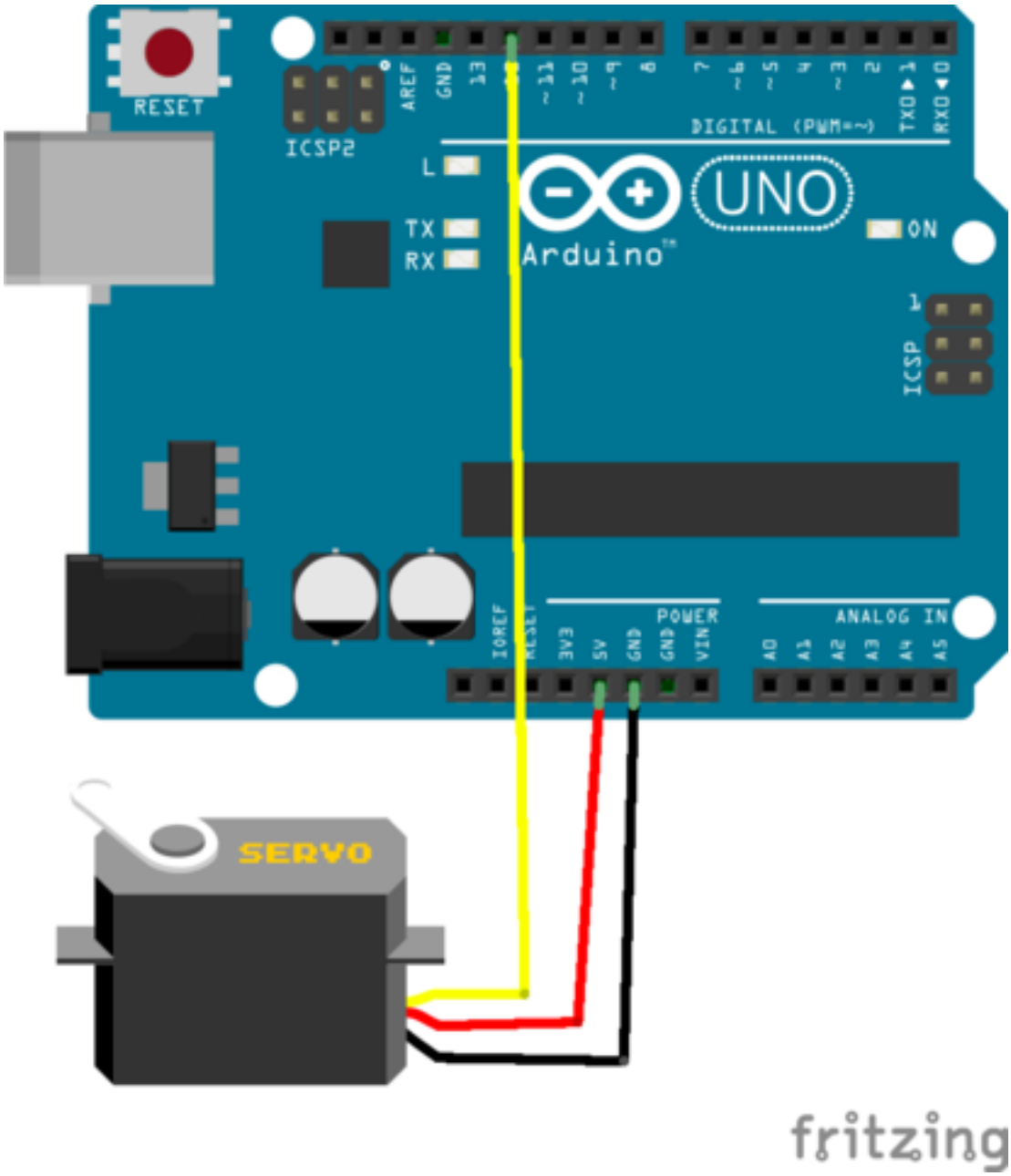
Express is a simple node.js web framework. And the twilio module makes it easy to communicate with the Twilio API.

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
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

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!

SO SMART WATCH

OVERVIEW

Create your own multifunctional smart watch. The LilyPad Arduino is a microcontroller board designed for wearables and e-textiles. It can be sewn to fabric and similarly mounted power supplies, sensors and actuators with conductive thread.

During this workshop you will learn how to sew and connect whole elements. Check how to use basic controllers to create various functionality.

Have you ever heard about pomodoro technique? When whole leds are bright, that means you are very busy and nobody should disturb you. When leds are off, buzzer sound tells you that this is brake time. This management method uses a timer to break down work into intervals and it is based on the idea that frequent breaks can improve mental agility.

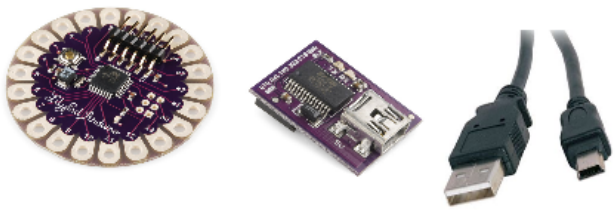
Blinking leds measuring our time is the first goal. Secondly we will upgrade our circuit to create light piano instrument and memory game.

PREPARATIONS

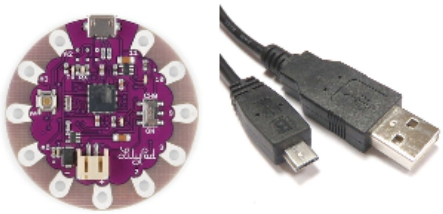
> We have two different Lilypad boards available: Arduino LilyPad FTDI and Arduino LilyPad USB. The difference is that Arduino LilyPad USB is simpler: it connects directly with computer by micro-USB cable and have less pins. Arduino LilyPad FTDI needs to be connected through FTDI Breakout (drivers are needed). > Feel free to choose one of them and follow appropriate instruction.

Make sure that you have collected:

- Arduino LilyPad FTDI, FTDI Breakout and mini USB cable



- or Arduino LilyPad USB and micro USB cable
- Battery Holder
- Cell Battery
- Buzzer




Lithium Cell
CR1225
3V

- LED - 3 pcs

- Photoresistor - 3 pcs

- Conductive thread

- Normal thread
- Needle
- Press-stud 2 pcs
- prepared material base

Arduino LilyPad FTDI

Arduino LilyPad USB

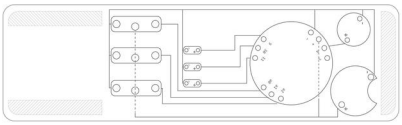
INSTALLING AND CONFIGURING THE ARDUINO SOFTWARE

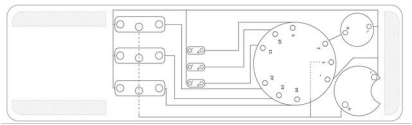
http://arduino.cc/en/Guide/HomePage

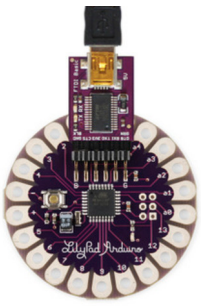
LAUNCHING THE LILYPAD ARDUINO APPLICATION ON WINDOWS

Get the latest version of arduino environment from the http://arduino.cc/en/Main/Software

Attach the FTDI board to your LilyPad Arduino board. Attach one end of the USB cable to your FTDI board and the other end to a USB port on your computer.







Select your board. You'll need to select the entry in the **Tools > Board** menu – *Lilypad Arduino w/ ATmega328*

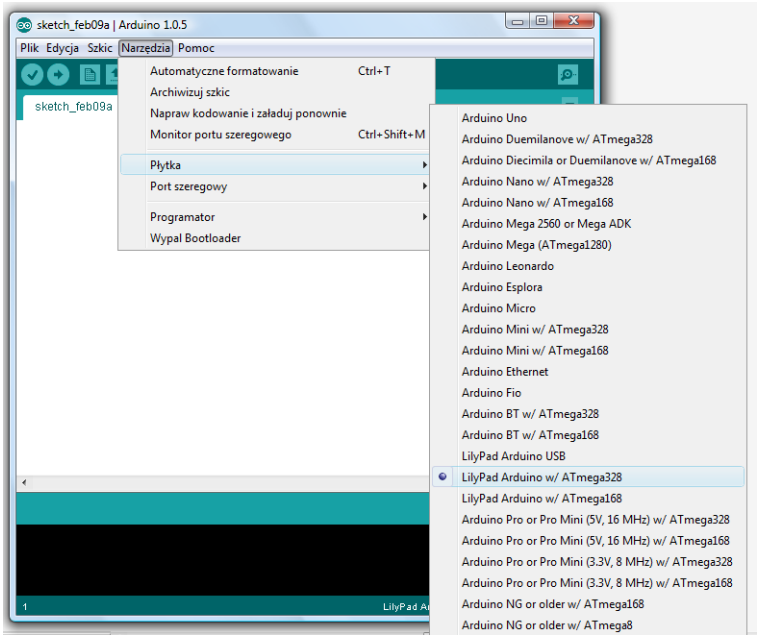
INSTALL THE DRIVERS

You will need to install the drivers for the FTDI board that enables you to program the LilyPad Arduino. You need to download the latest version of the drivers from the http://www.ftdichip.com/Drivers/VCP.htm

If your computer is using Windows 7, Windows Vista, Windows XP, or an earlier operating system, see this website to determine whether you should use the x86 (32 bit) or x64 (64 bit) drivers. If your computer is using Windows 8 or a later operating system, you should download the x64 (64 bit drivers). When the download is finished, extract all of the contents of the *.zip* file to a location you will remember.

Follow the guide on the FTDI website: http://www.ftdichip.com/Support/Documents/InstallGuides.htm, that corresponds to your operating system to complete the installation process.

You'll need to restart your computer after installing the drivers.

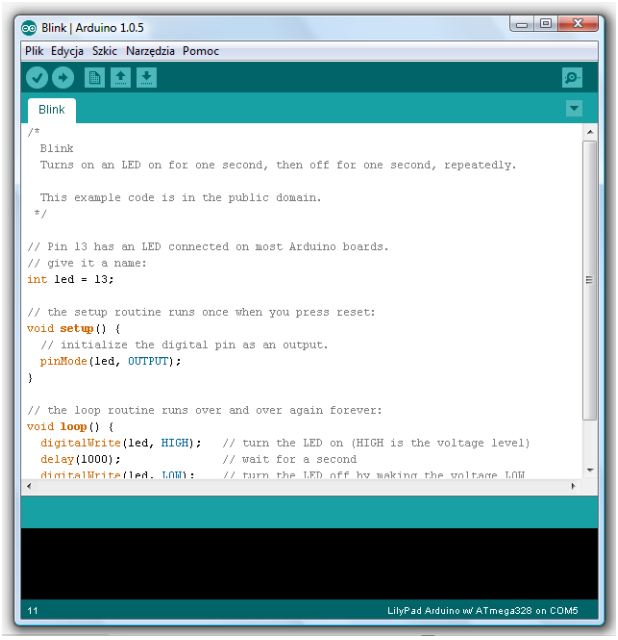


Select your serial port. Select the serial device of the Arduino board from the Tools > Serial Port menu. This is likely to be **COM3** or higher (**COM1** and **COM2** are usually reserved for hardware serial ports).

Upload the program. Open the LED blink example sketch: **File > Examples > 1.Basics > Blink**.

Now, simply click the "Upload" button in the environment (arrow ->). Wait a few seconds - you should see the leds on the board flashing. If the upload is successful, the message "Done uploading." will appear in the status bar.

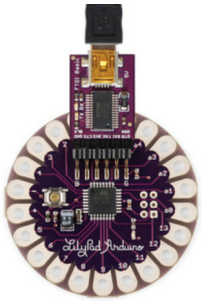
A few seconds after the upload finishes, you should see the pin 13 (L) LED on the board start to blink (in orange). If it does, congratulations! You've gotten LilyPad up-and-running.



LAUNCHING THE ARDUINO APPLICATION ON MAC OSX

Get the latest version of LilyPad Arduino environment from the http://arduino.cc/en/Main/Software

Attach the FTDI board to your LilyPad Arduino board. Attach one end of the USB cable to your FTDI board and the other end to a USB port on your computer.



INSTALL THE FTDI DRIVERS

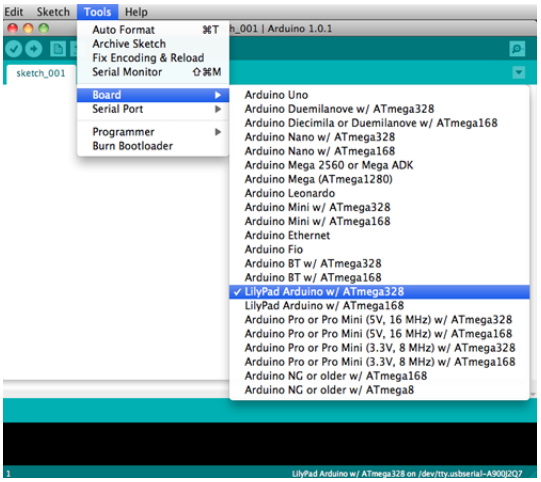
You will need to install the drivers for the FTDI board that enables you to program the LilyPad Arduino. You need to download the latest version of the drivers from the FTDI website: http://www.ftdichip.com/Drivers/VCP.htm

Scroll down to the middle of this page, where there is a table listing drivers for different computers. Click on the most recent driver that is compatible with your computer. If your computer is running OS 10.5 or higher you should use

the x64 (64-bit) drivers. If you're running OS 10.4 you should use the x32 (32-bit) drivers. Otherwise, you should use the PPC (PowerPC) drivers. To determine which OS your computer has, click on the apple icon in the upper right hand corner of your screen and click on "About This Mac" menu item.

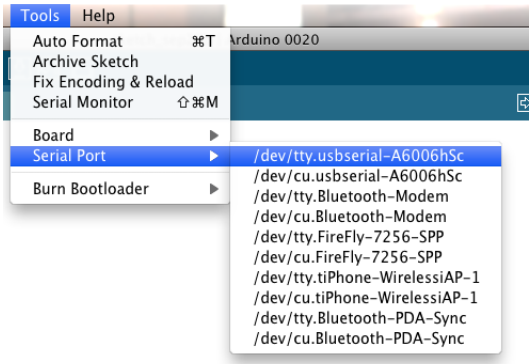
SELECT YOUR BOARD

You'll need to select the entry in the **Tools > Board** menu that corresponds to your LilyPad Arduino.



SELECT YOUR SERIAL PORT

Select the correct serial port from the **Tools > Serial Port** menu.



Upload the program. Open the LED blink example sketch: **File > Examples > 1.Basics > Blink**.

Now, simply click the "Upload" button in the environment (arrow ->). Wait a few seconds - you should see the leds on the board flashing. If the upload is successful, the message "Done uploading." will appear in the status bar.

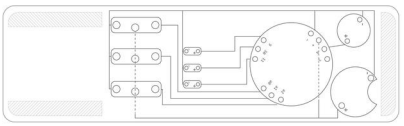
MAIN FLOW

Lets start with stop-watch function.

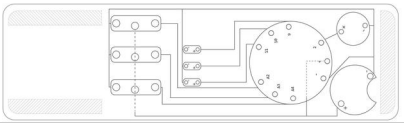
First of all we have to prepare our circuit. We will sew a bit, but don't worry – everything is prepared, there is not much left.

This is our basis.

Arduino LilyPad FTDI



Arduino LilyPad USB



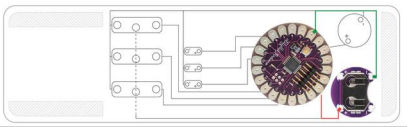
Whole elements and thread paths are printed on. All you have to do is to follow the instructions. Each piece of LilyPad have adhesive tape at the back side to help you place Lilypad elements on the material. Before you sew anything, stick it.

To make your watch durable, sew elements using normal thread and then use conductive thread to connect pins. When sewing, it is very important that the different threads never overlap one another.

To make sawing easier, whole thread paths are dotted.

Just to help you a bit, LilyPad board and battery holder are already sewed and connected.

Arduino LilyPad FTDI



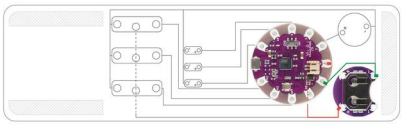
Arduino LilyPad USB

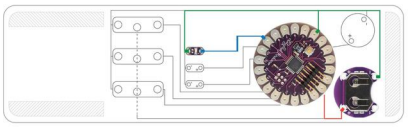
Stick first LED on the proper place (start from the top one). Note the POSITIVE (+) and NEGATIVE (-) ends of the LED lights. To keep the sewing clean, make sure the negative ends are all pointed leftwards, and the positive ends are pointed rightwards. Sew (+) with pin 9 (blue line) on LilyPad board and (-) with ground (-) (green line).

Arduino LilyPad FTDI

Arduino LilyPad USB

Congratulation! Your first circuit is done! Lets test it. Upload again Blink programme. Don't forget to change pin number to 9. Does it work?





Let's go further. Sew another two leds. Connect positive ends to pins 10 and 11. Don't forget to ground them (green line) – you can sew whole negative ends together.

Arduino LilyPad FTDI

Arduino LilyPad USB

Try to test it now. Modificate "Blink" program by adding new leds. (If you need help, go to SmartWatch folder and run "Blink_three leds" file).

Let's add sound. Stick buzzer in the right bottom corner. Sew negative end with existing ground path. Positive end connect:

- with pin 4 at FTDI board
- with pin 3 at USB board

Arduino LilyPad USB

Check if it's connected correctly. Go to SmartWatch folder and run `Simple_tone` file. If you hear tones, your buzzer is connected correctly.

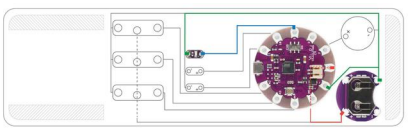
It seems that we have completed our circuit, so let's start with Pomodoro watchstop programme. Our goal is to successively fade out leds to communicate time intervals. When whole leds are bright, that means you are very busy and nobody should disturb you. When whole leds are off, buzzer sound tells you that this is time for a brake.

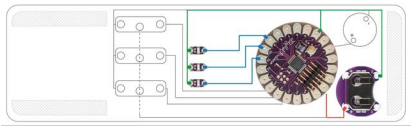
Or just cook ideal soft-boiled egg…

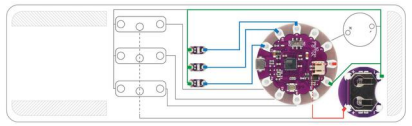
Define led and buzzer pin connection:

```
const int ledPin1 = 9;
const int ledPin2 = 10;
const int ledPin3 = 11;
const int buzzerPin = 4;
```

Define our pomodore cycle time in seconds. The three LEDs will output the the progress of the cycle by slowly dimming







down, one after another:

```
const int pomodoroCycleLenght = 1*60;
```

Define the lenght of the tone (in seconds) generated when the pomodoro cycles ends.

```
const int buzzingTime = 1;
```

Define the frequency of the tone generated when the pomodoro cycles ends.

```
const int buzzingFrequency = 2093;  // C7
```

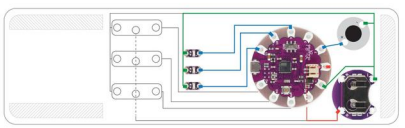
Define controlls the “base” time of the board. A value of 1000 means 1 program second coresponds to 1 real second. You can change this value for debug purpouses.

```
const int MILLIS_IN_SECOND = 1000;
```

Afterwards perform setup of the board. This is called by the Arduino framework.

```
void setup() {
  //We set all used pins to output mode.
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
  pinMode(buzzerPin, OUTPUT);
}
```

Generates a tone (square wave) on the provided `speakerPin`. The tone will last for `timeInMilliseconds` and have the `frequencyInHertz`.



```
void beep (unsigned char speakerPin, int frequencyInHertz, long timeInMilliseconds) {
  int x;
  long halfCycleDelay = (long) (1000000 / frequencyInHertz / 2);
  long cycleRepeats = (long) ((timeInMilliseconds * 1000) / (halfCycleDelay * 2));
  for (x = 0; x < cycleRepeats; x++) {
    digitalWrite(speakerPin,HIGH);
    delayMicroseconds(halfCycleDelay);
    digitalWrite(speakerPin,LOW);
    delayMicroseconds(halfCycleDelay);
  }
}

int time = 0;
```

Next is the main loop method. Called over and over again by the Arduino framework.

```
void loop() {
 const int timePerLED = pomodoroCycleLenght / 3; /* each LED presents the passing of one third of the pomodoro cy
```

For each led we first calculate the time relative to it's part of the pomodoro cylce and constrain it to the range `[0, timePerLED]`. We then map this value to a brightness level.

```
 analogWrite(ledPin1, map(constrain(time, 0, timePerLED), 0, timePerLED, 255, 0)); //set LED1 (first part of po
 analogWrite(ledPin2, map(constrain(time - timePerLED, 0, timePerLED), 0, timePerLED, 255, 0)); //set LED2 (se
 analogWrite(ledPin3, map(constrain(time - timePerLED*2, 0, timePerLED), 0, timePerLED, 255, 0)); //set LED3 (
```

We now need to wait for exactly one second.

If this is the end of the pomodoro cycle we will emit a buzz for 1000ms. The methods blocks for this time. Otherwise we will simply use the delay function.

```
  if(time >= pomodoroCycleLenght - buzzingTime){
    beep(buzzerPin, buzzingFrequency, MILLIS_IN_SECOND);
  } else {
    delay(MILLIS_IN_SECOND);
  }
```

Increment the time by one second, and limit it to the pomodoro cycle lenght.

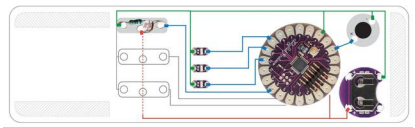
```
  time = (time + 1) % pomodoroCycleLenght;
}
```

It's time to wear your new smart watch! Get things done, manage your time: plan, track, and visualize your tasks.

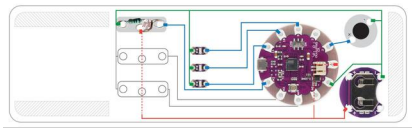
SECOND FLOW

If you still have some enthusiasm left, try to upgrade your smart stopwatch. We will create musical instrument by adding some analog inputs. For this purpose add photoresistors to our circuit. We will generate sound by fading out brightness.

Arduino LilyPad FTDI



Arduino LilyPad USB



Connect photoresistor to:

– analog pin A0 at FTDI board – analog pin A2 at USB board

Ground it and additionally sew with power pin (+) on Lilypad board as shown above.

Did you link it rightly? (go to `SmartWatch` folder and run `Fade_leds` file)

```
int photocellPin = 0;
int photocellReading=10;

int LEDpin1 = 9;
int LEDpin2 = 10;
int LEDpin3 = 11;

int LEDbrightness;

void setup(void) {

  Serial.begin(9600);
}

void loop(void) {

  photocellReading = analogRead(photocellPin);
  Serial.println( photocellReading);
  LEDbrightness = map(1024-photocellReading, 0, 1023, 0, 255);
```

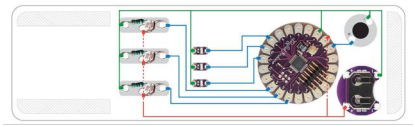
```
  Serial.println(LEDbrightness);
  analogWrite(LEDpin1, LEDbrightness);
  analogWrite(LEDpin2, LEDbrightness);
  analogWrite(LEDpin3, LEDbrightness);

  delay(100);
}
```

While getting dark leds are getting brighten. Don't you think this is great signaling for your night bike rides?

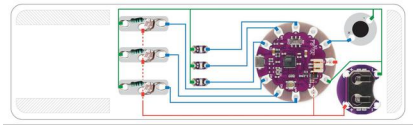
Finish keyboard for our smart watch instrument by Add two left photoresistor. Ground them (green path), connect with pin (A1 and A2) and with power.

Our smart watch instrument keyboard is finished.

Arduino LilyPad FTDI



Arduino LilyPad USB



Time to code:

```
// pin definitions
const int photo0Pin = 0;

const int led0Pin = 9;
const int led1Pin = 10;
const int led2Pin = 11;

const int buzzerPin = 4;

const int photoMin = 50;  // the value for a fully covered photo cell
const int photoMax = 160;  // the value for a fully uncovered photo cell

void setup(void) {
  // We'll send debugging information via the Serial monitor
  Serial.begin(9600);
```

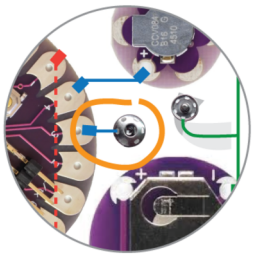
```
  // setup output pins
  pinMode(led0Pin, OUTPUT);
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);

  // setup input pins
  pinMode(photo0Pin, INPUT);
}

// the photo cell sensor has major noise. We will use a variation of a moving window average.
int photoAvg = 100;
const int photoAvgSamples = 5;

const int sampleTime = 10;

void loop(void) {
  // read the photo cell value (ADC read value has a range of 0-1024)
  int photo0value = analogRead(photo0Pin);

  // normalize the raw sensor value. 0 is a covered sensor. 100 and above is a uncovered sensor.
  photo0value = map(max(photo0value, photoMin), photoMin, photoMax, 0, 100);

  // calculate our moving avarage variant
  photoAvg = (photoAvg * (photoAvgSamples-1) + photo0value) / photoAvgSamples;

  // output debuging information
  Serial.println(photoAvg);

  // we are only interested in values under 100 (a somewhat covered sensor)
  if(photoAvg < 100) {
    digitalWrite(led0Pin, HIGH);
    // we map our range 0->100 to a range of 6000->2000 (note the inversion).
    beep(buzzerPin, map(photoAvg, 0, 100, 6000, 2000), sampleTime);
  } else {
    digitalWrite(led0Pin, LOW);
    delay(sampleTime);
  }
}

// Generates a tone (square wave) on the provided speakerPin. The tone will last for timeInMilliseconds and have
```

```
void beep (unsigned char speakerPin, int frequencyInHertz, long timeInMilliseconds) {
  int x;
  long halfCycleDelay = (long) (1000000 / frequencyInHertz / 2);
  long cycleRepeats = (long) ((timeInMilliseconds * 1000) / (halfCycleDelay * 2));
  for (x = 0; x < cycleRepeats; x++) {
    digitalWrite(speakerPin, HIGH);
    delayMicroseconds(halfCycleDelay);
    digitalWrite(speakerPin, LOW);
    delayMicroseconds(halfCycleDelay);
  }
}
```

I'm sure your creativity will allow you to develop more ideas using this circuit. How about memory game? Rules: remember and repeat sequences of flashing lights. Game starts with one light. In every round additional light is added to sequence. Try to write it by yourself or go to SmartWatch folder and run `Memory` file.

But, it is not finished yet. Why not to add button to switch between two programs?

We will use press-stud. Top part is already connected. Your job is to sew bottom part near pin 3 and connect it.



Try to modify code a bit – it's simple! If little help is needed go to SmartWatch folder and run "Switch" file.

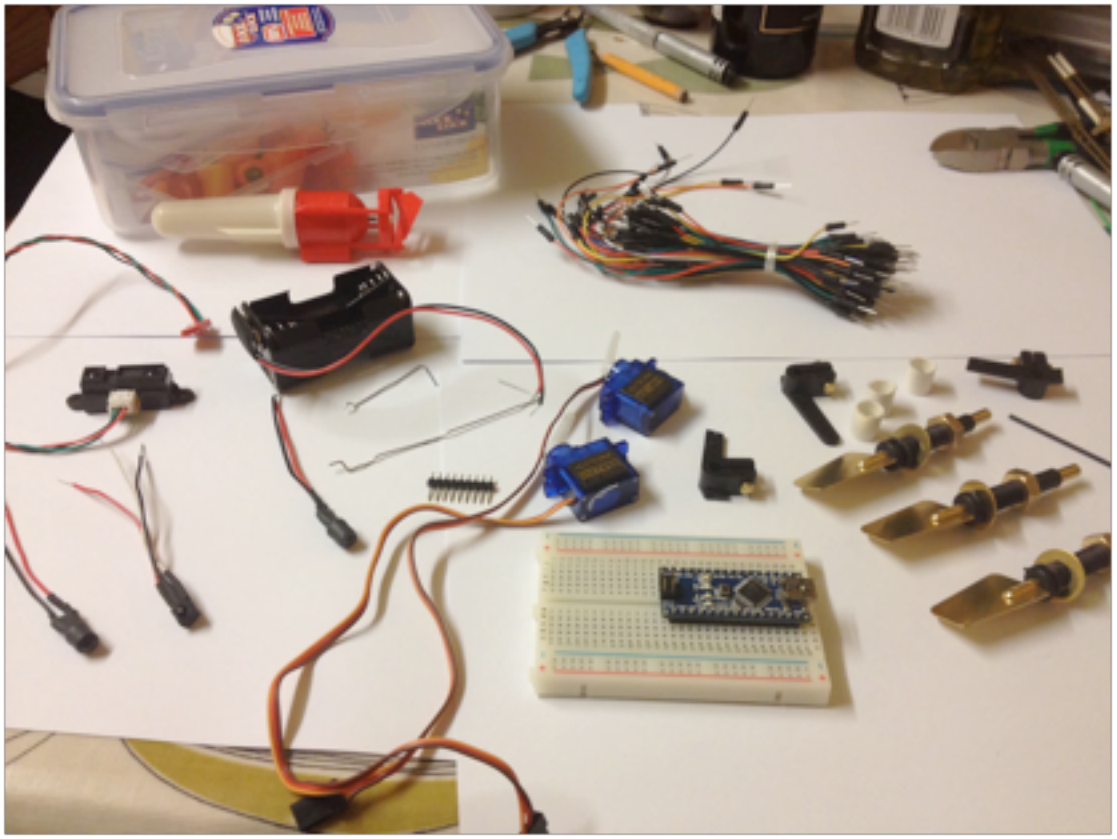
ADDITIONAL SOURCES

Here is a list of links to specifications, tutorials and inspirations:

- Lilypad arduino specyfication http://arduino.cc/en/Main/arduinoBoardLilyPad
- Lilypad tutorials http://lilypadarduino.org/
- Wearable inspirations http://learn.adafruit.com/category/flora # 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

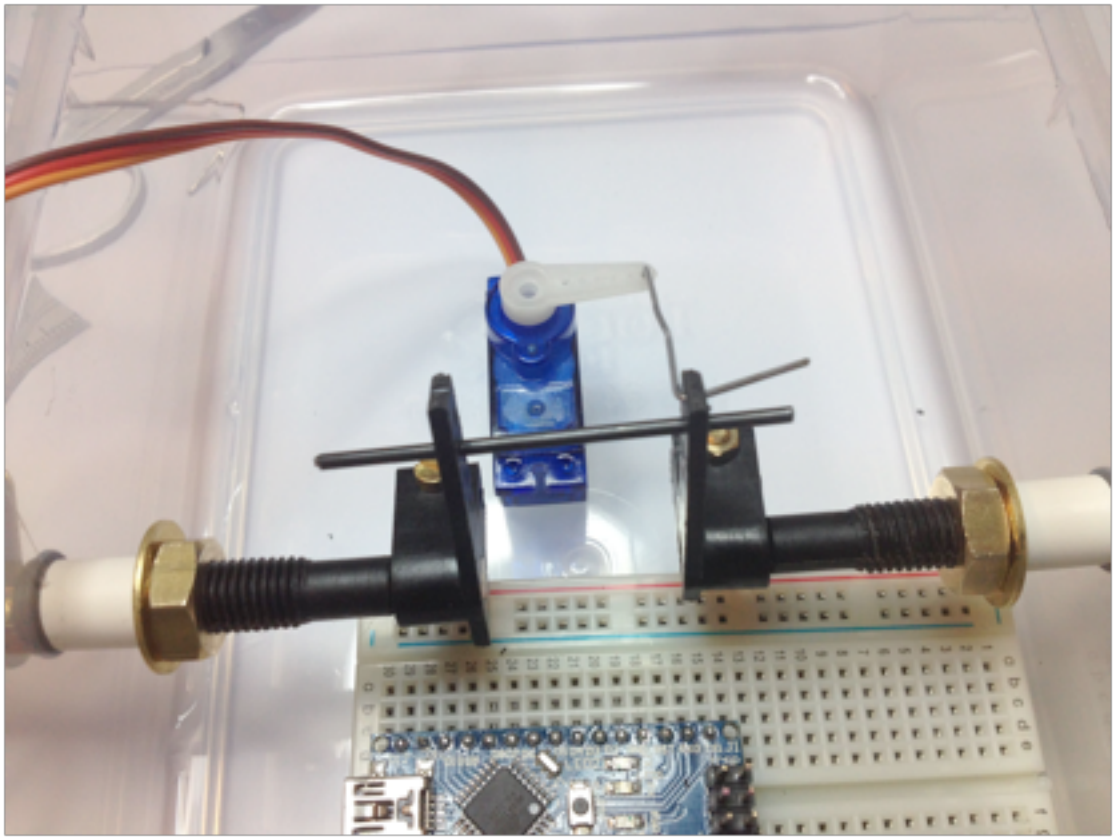
The main body of the AUV is going to use a Lock & Lock container normally used for storing food. It has a built in seal and is both airtight and watertight. Three holes have been pre-drilled into the container for installation of the dive planes and rudder. The dive planes allow the AUV to submerge very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

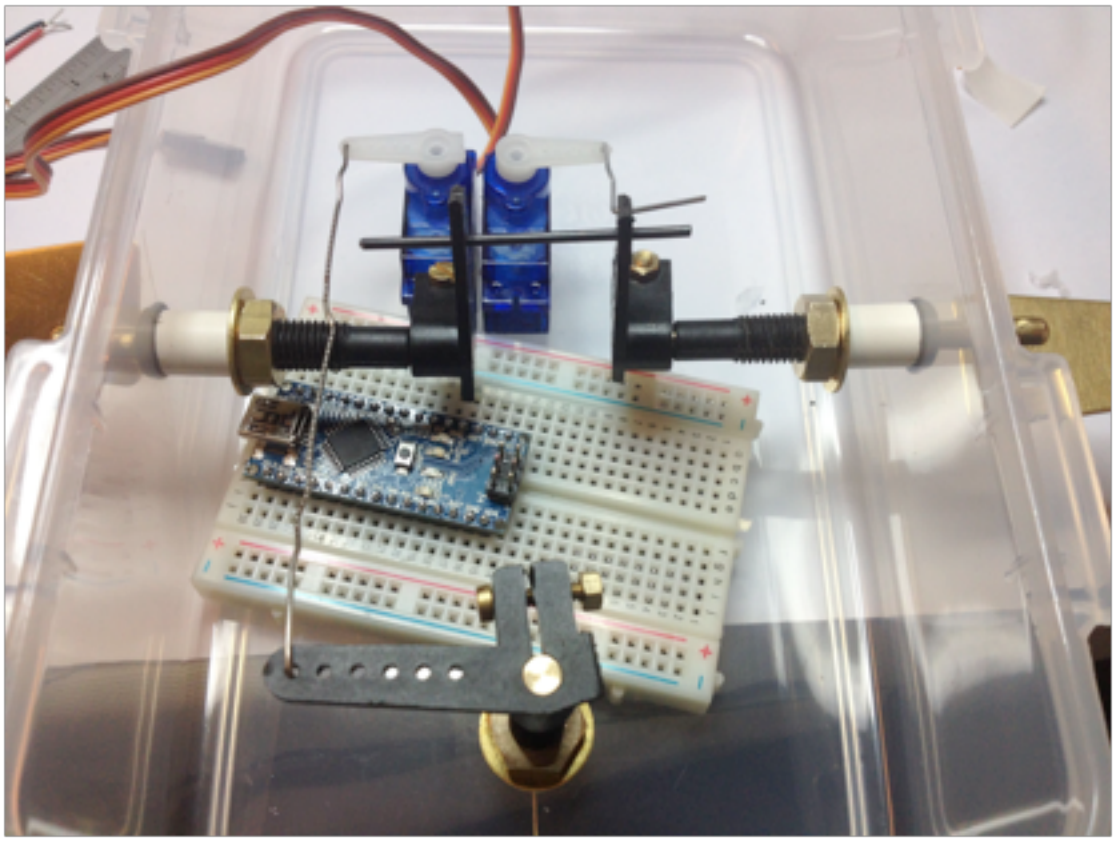
1. Assemble dive planes / rudder into container with provided spacers, washers and 13mm nuts. Take care not to over tighten the nut as this will distort the O-ring and be a possible point of water entry.

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.



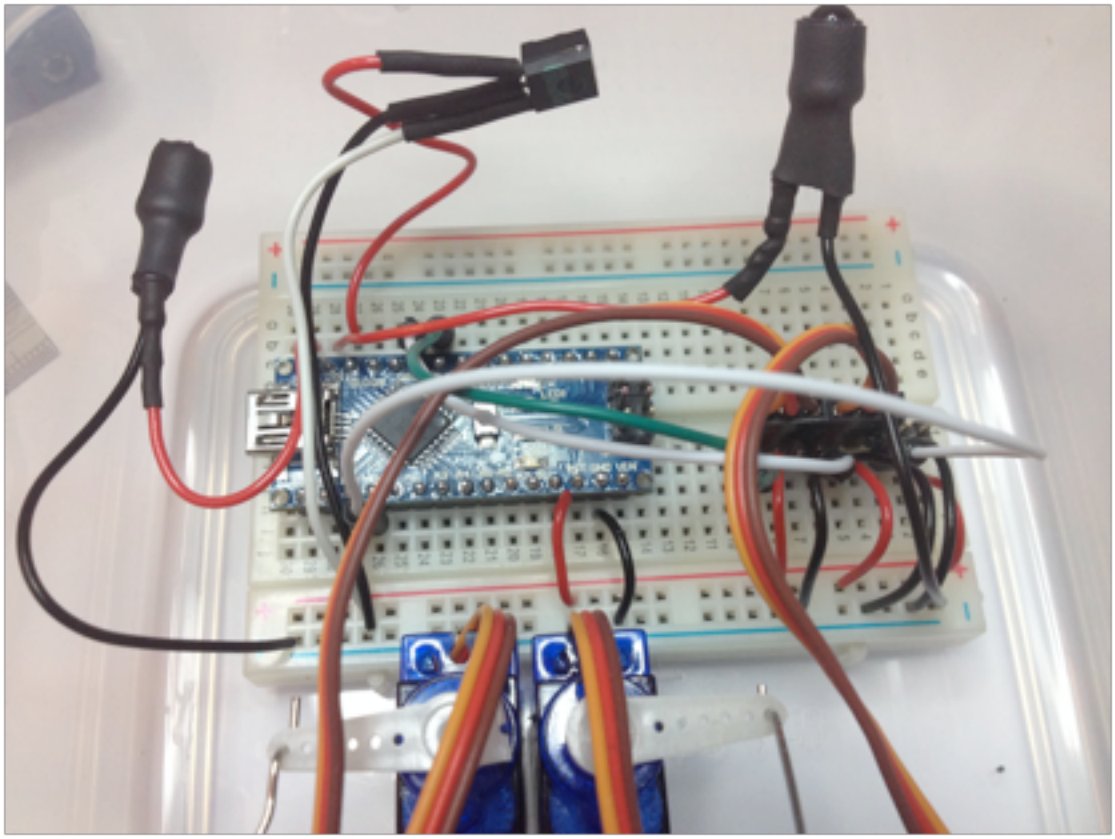
17. Test rudder assembly using servo tester.

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

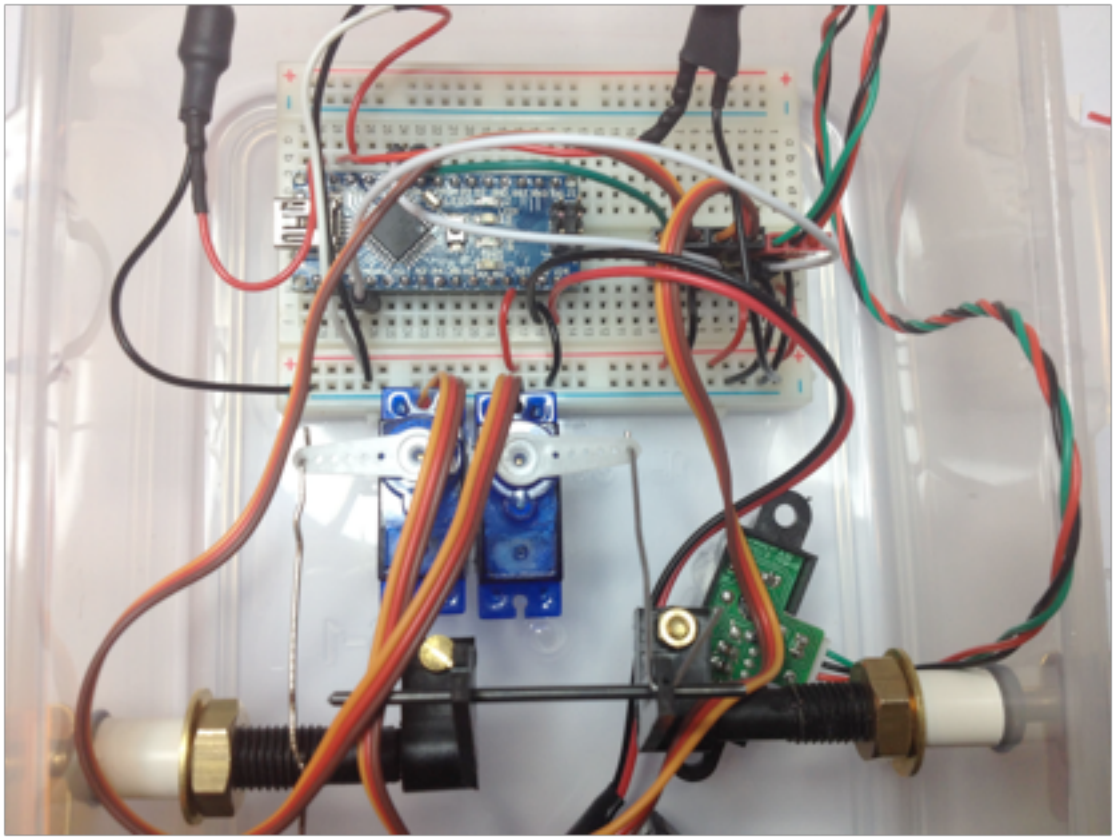
We are next going to add a sensor to detect when the AUV is near the bottom of the pool. We are going to use infrared



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

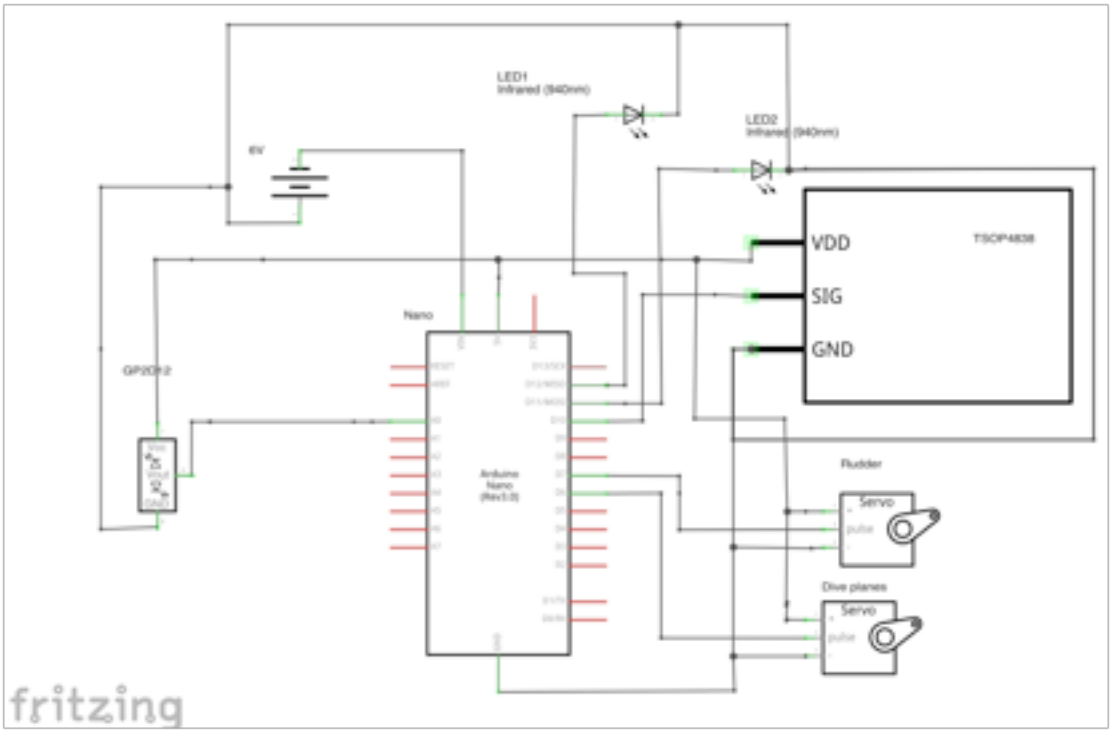
Connect the sharp GP2D12 sensor to the breadboard using the remaining three pins on the IDE header and use the male jumpers to connect to the Arduino Nano. Use the circuit diagram from below for reference. With the distance sensor wired up we can now move onto fixing the IR LED's, IR receiver and the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

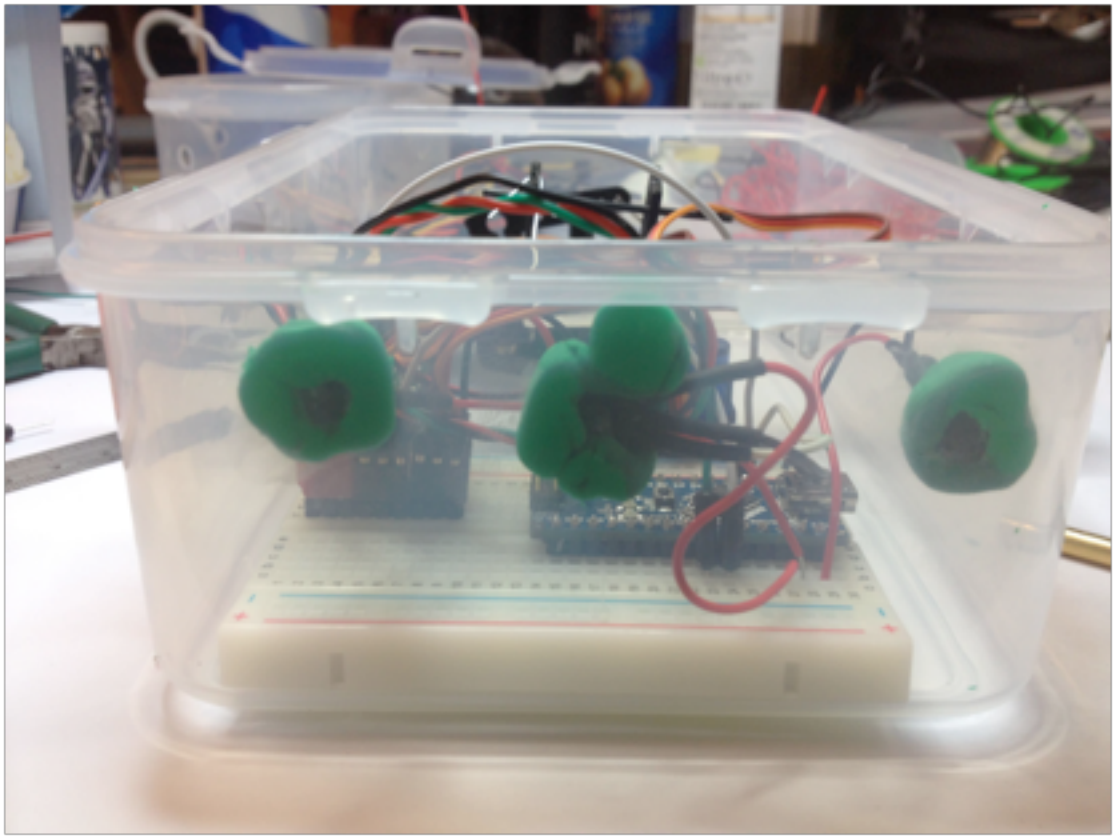


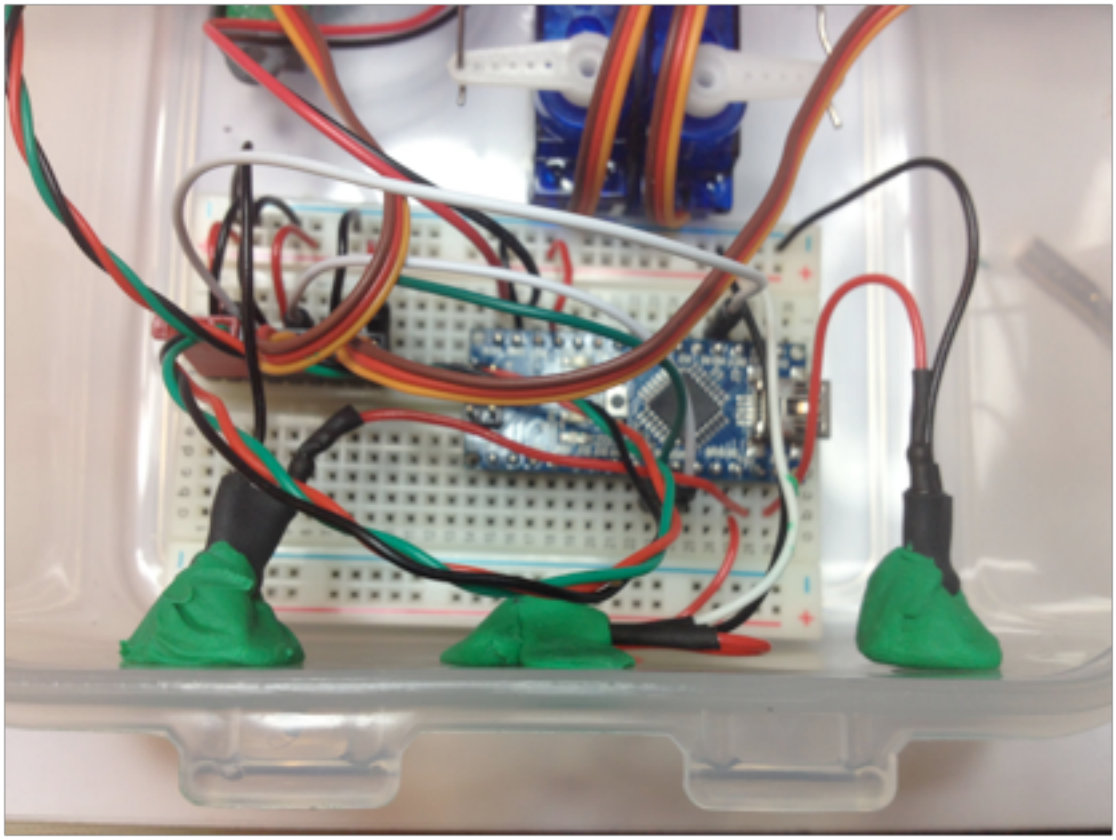
AUV SCHEMATIC

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
GP2Y0A2
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.

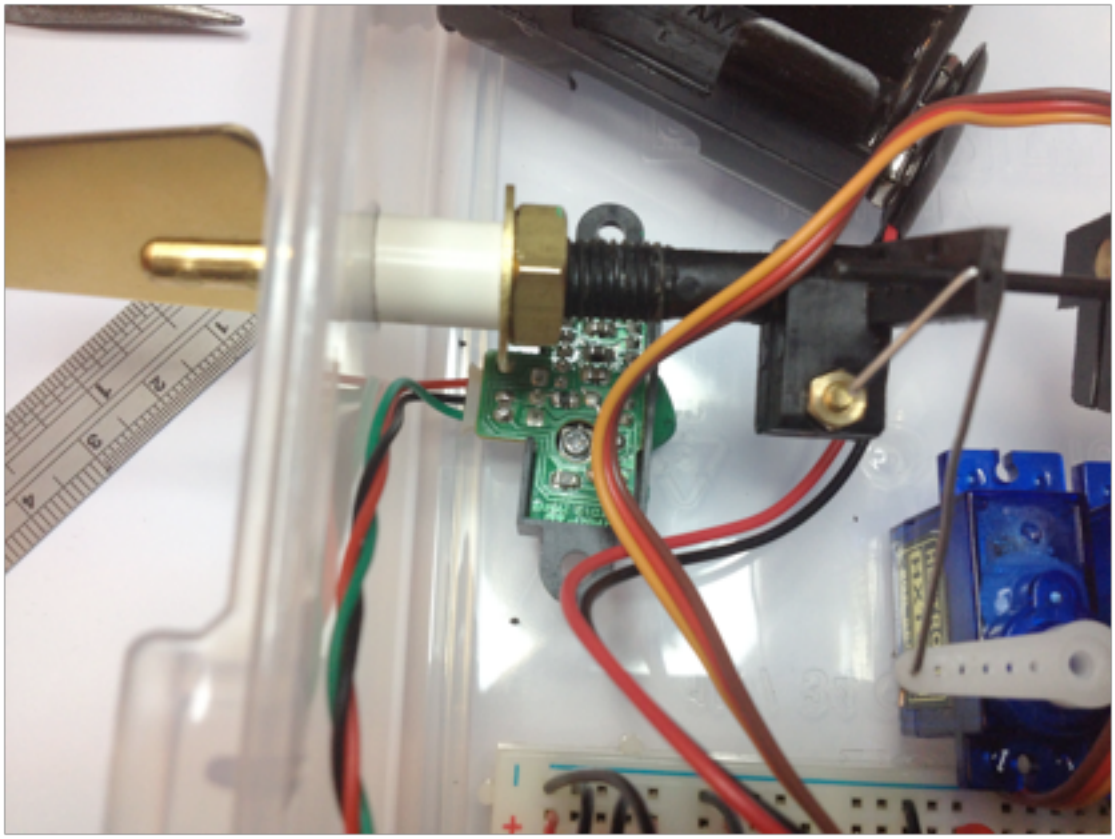
INSTALLING THE BATTERY PACK

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

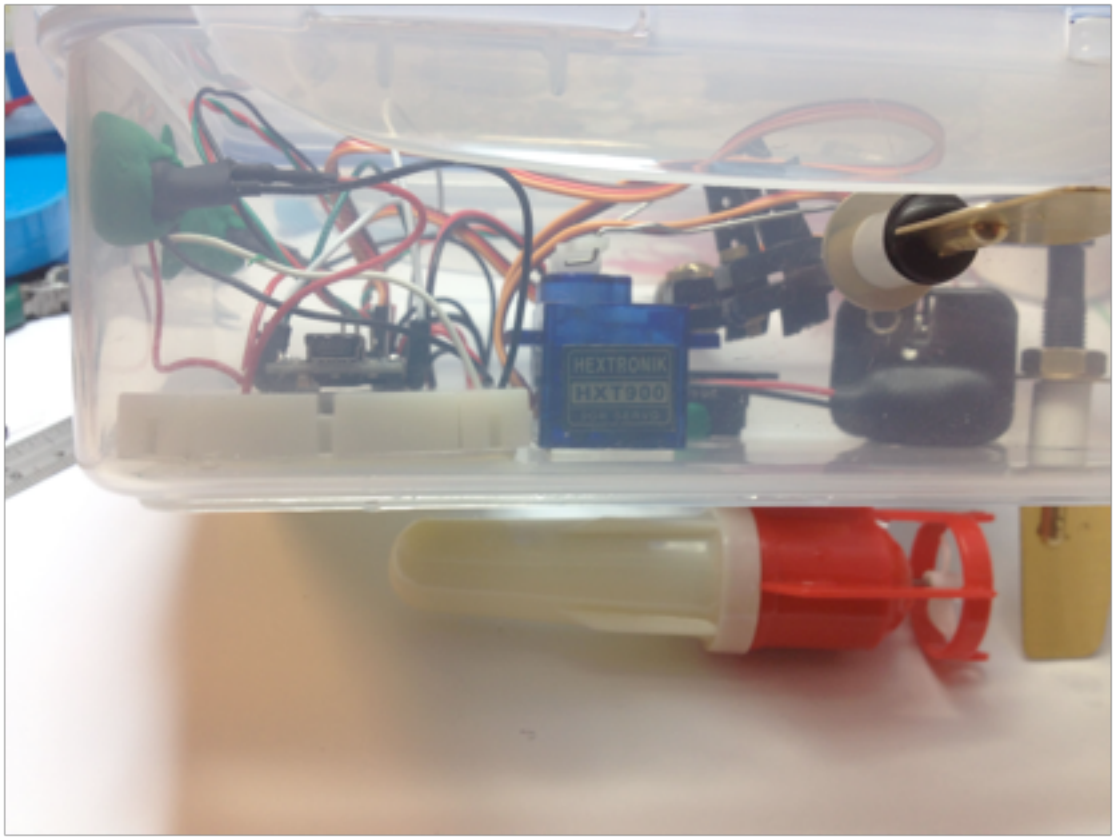


That completes the internal electronic and mechanical components of the AUV. We want to make a quick check that everything is still watertight. Pop on the lid of the AUV and gently submerge it into the pool do a visual check for leaks and then take it out and dry it.

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