**Team Members:**

**Samuel Lum**

**Luan Dinh**

**Daniel Waters**

**Abstract**

**Title:**

Otto the Robot

**Problem:**

Allow the user to remotely control a 3D printed Otto Robot using an Android app and a Bluetooth serial connection. The user is given options through the app to move forward, backwards, left, right, or play sound. The robot will sense objects in front of it, and refuse to move forward if something is within 5cm in front of it.

**Techniques used:**

1. Print the pieces of the robot using a 3D printer.
2. Wire electrical circuits to function correctly with an Arduino Mini microcontroller.
3. Allow the user to connect/disconnect to a Bluetooth radio in the Otto Robot through an Android application.
4. Once a command is received from the user, control the movements of the robot by 4 servo motors placed inside Otto.
5. Detect the distance in front of the robot using an ultrasonic sensor, and decide to move forward based on returned distance.
6. Allow Otto to make sound using a connected buzzer.
7. The Android application was written in Java using Android Studio, and the Arduino sketch was written in C/C++ using Arduino IDE.

**Team Members:**

**Samuel Lum**

**Luan Dinh**

**Daniel Waters**

**Project: Otto the Robot**

**Objectives:**

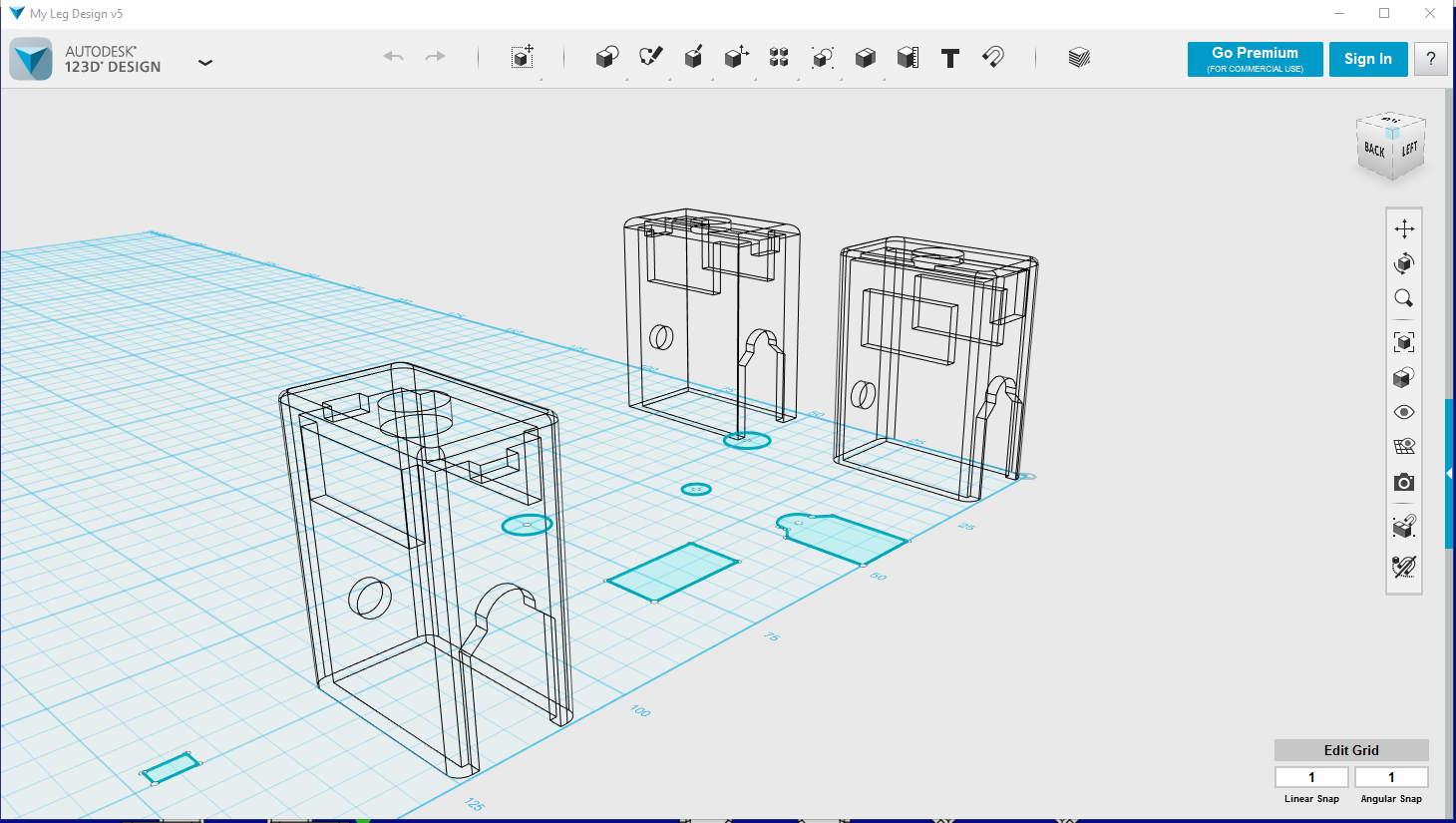
* Quick explanation of what the Otto the Robot is.
* Give a quick overview of how a 3D printer works.
* Explain how the Android application works.
* Use a GUI for the user to communicate with Otto
* Techniques used:
  + 3D design a new leg to connect correctly
  + 3D printed the parts
  + GUI for the Android application
  + Serial Bluetooth connection between phone and Arduino



Otto the Robot is:

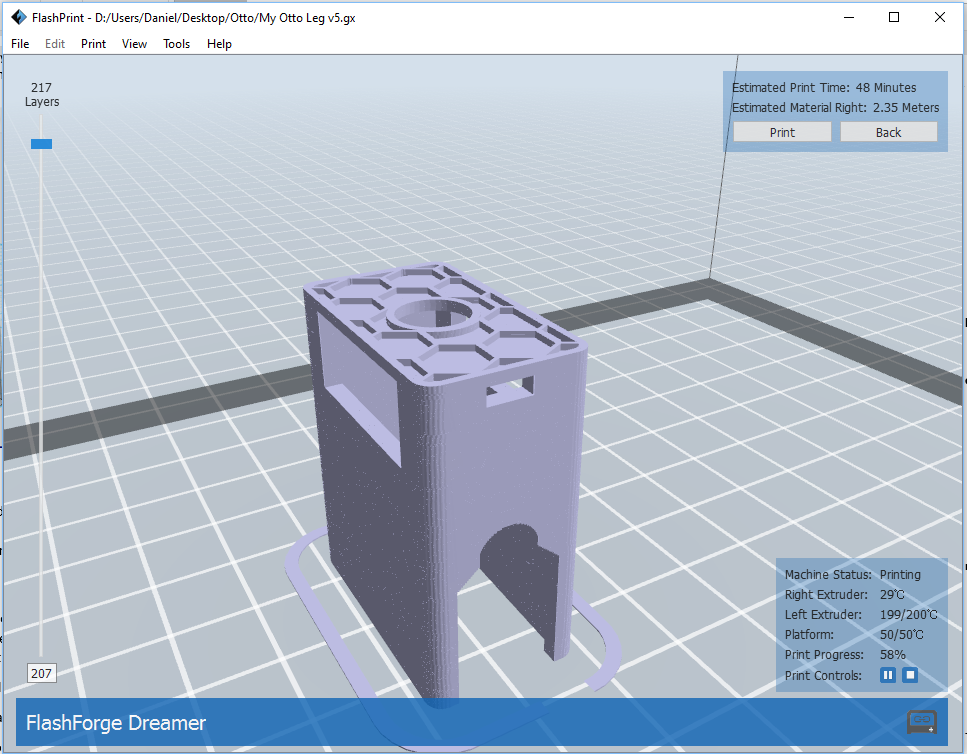
**Otto**, is an open source design for a robot found at <http://otto.strikingly.com>. The website supplies 3D printer design files for the pieces needed to build the robot and electrical wiring diagrams. It also includes a PDF that takes the user step by step in the build process. While the outer shell of Otto is given, the abilities for Otto is left up to the user to design themselves.

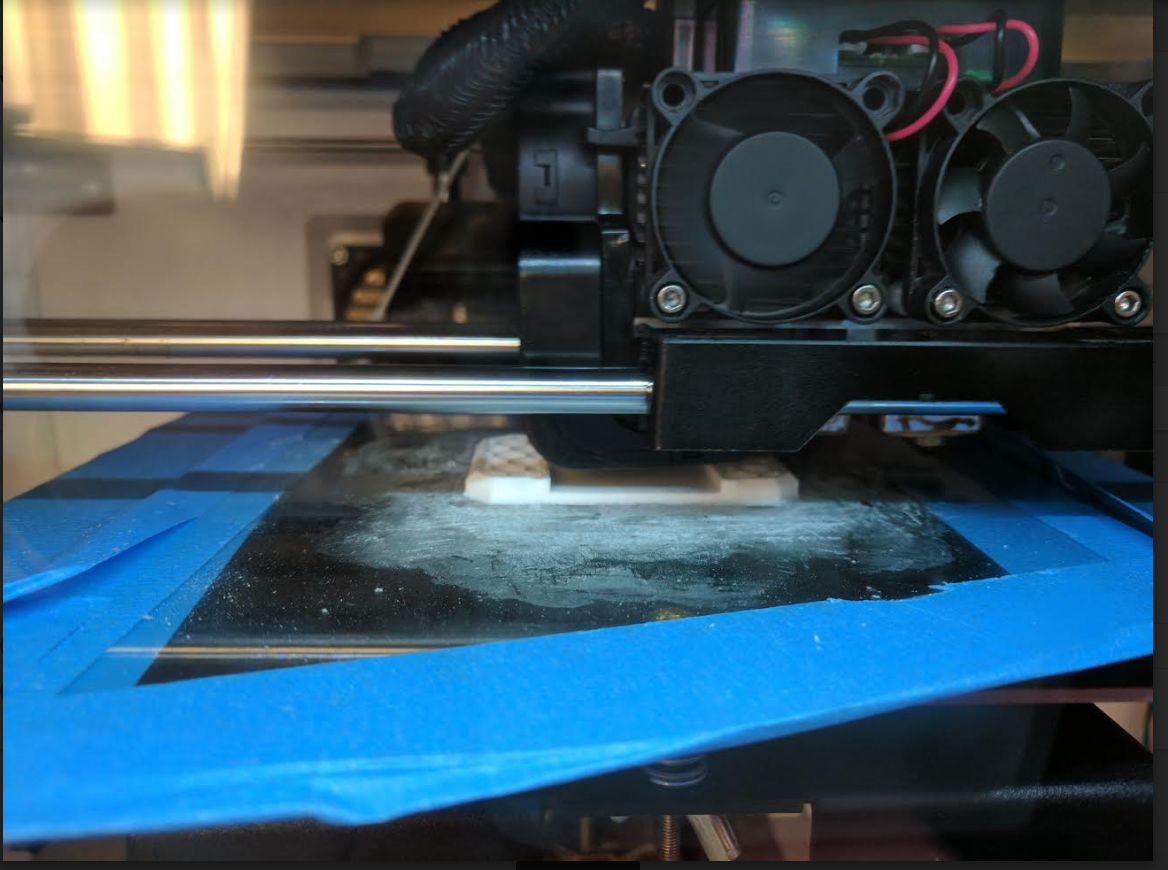
**3D Design**



A 3D printer prints files that are designed in a 3D software such as Autodesk 123D Design. Inside this program a designer uses basic shapes to mold a shape into something that the printer would be able to understand. Example in this picture a simple box has pieces removed from it to form the shape of the newly designed Otto leg that will fit with the servos purchased. In the background are different versions of the leg that did not work correctly.

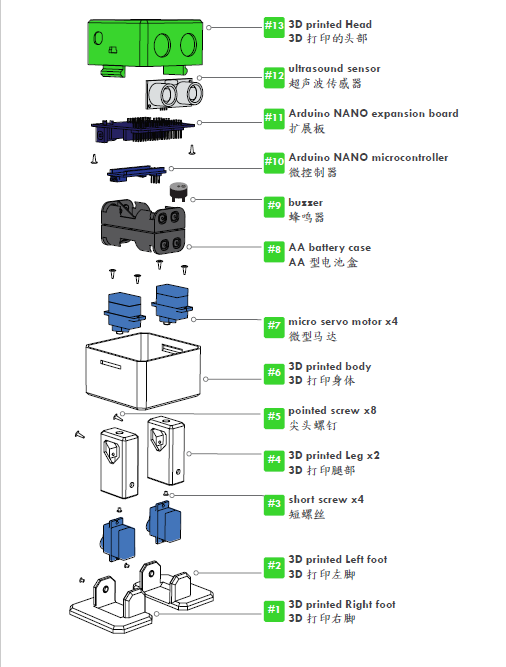
Once the figure is complete, it is exported to another software program that takes the 3D object and slices it into 100’s or 1000’s of tiny layers that the printer will use to build the object layer on layer. The program that I am using here is called FlashForge FlashPrint. For this simple leg at a standard quality it slices it down to 217 layers that needs to be printed.



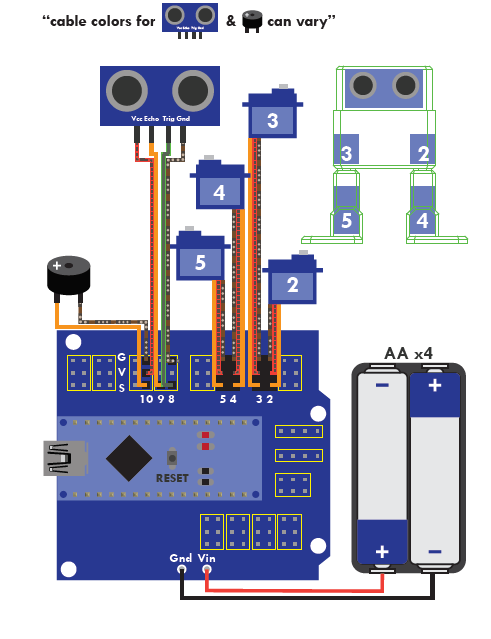


Finally, the printer understands the layers, and builds the object layer upon layer heating up plastic and printing them with small droplets of plastic.

**Otto the Robot - Parts Schematic**

Parts and Wiring Diagram for the Otto Robot

**Otto the Robot – Wiring Diagram**



**Parts Needed:**

**1x - Arduino Nano R3 Bluetooth embedded -** [**https://www.dfrobot.com/index.php?route=product/product&product\_id=1122**](https://www.google.com/url?q=https://www.google.com/url?q%3Dhttps://www.dfrobot.com/index.php?route%253Dproduct/product%2526product_id%253D1122%26amp;sa%3DD%26amp;ust%3D1477184990057000%26amp;usg%3DAFQjCNEKTpgmWJndd0WFRKf9GHzIdCvTMQ&sa=D&ust=1477184990073000&usg=AFQjCNFn6XbLA51Xuj65Qrap0tamuYk8sw)

**1x - Arduino Nano Shield I/O -**

[**https://www.amazon.com/Solu-Expansion-Sensor-Arduino-Compatible/dp/B00UBEHJUO/ref=sr\_1\_1?ie=UTF8&qid=1463706846&sr=8-1&keywords=Arduino+NANO+Shield+I%2FO**](https://www.google.com/url?q=https://www.google.com/url?q%3Dhttps://www.amazon.com/Solu-Expansion-Sensor-Arduino-Compatible/dp/B00UBEHJUO/ref%253Dsr_1_1?ie%253DUTF8%2526qid%253D1463706846%2526sr%253D8-1%2526keywords%253DArduino%252BNANO%252BShield%252BI%25252FO%26amp;sa%3DD%26amp;ust%3D1477184990058000%26amp;usg%3DAFQjCNF9IKG_SzKIB6bwvJT9f6koBEpQsA&sa=D&ust=1477184990073000&usg=AFQjCNEnJDjROvTJA7vTkAUB1cH5erVClg)

**1x - USB-A to Micro USB Cable -**

[**https://www.amazon.com/gp/product/B012A1ML6W/ref=oh\_aui\_detailpage\_o02\_s00?ie=UTF8&psc=1**](https://www.google.com/url?q=https://www.google.com/url?q%3Dhttps://www.amazon.com/gp/product/B012A1ML6W/ref%253Doh_aui_detailpage_o02_s00?ie%253DUTF8%2526psc%253D1%26amp;sa%3DD%26amp;ust%3D1477184990059000%26amp;usg%3DAFQjCNH3zjUvan4TNBtP3k6qlCdaCycngQ&sa=D&ust=1477184990073000&usg=AFQjCNGHQj67zGD1UO1rnAD3TlwiGaLGSw)

**1x - HC-SR04 Ultrasound Sensor -**

[**https://www.amazon.com/gp/product/B004U8TOE6/ref=oh\_aui\_detailpage\_o03\_s01?ie=UTF8&psc=1**](https://www.google.com/url?q=https://www.google.com/url?q%3Dhttps://www.amazon.com/gp/product/B004U8TOE6/ref%253Doh_aui_detailpage_o03_s01?ie%253DUTF8%2526psc%253D1%26amp;sa%3DD%26amp;ust%3D1477184990060000%26amp;usg%3DAFQjCNFkso1mr9lOPCX5DlcGTcIGxcjSXw&sa=D&ust=1477184990074000&usg=AFQjCNE32bQsNvbRtGYWgVnL_eIF3bq3qQ)

**4x - Mini Servo SG90 9g -**

[**https://www.amazon.com/gp/product/B01608II3Q/ref=oh\_aui\_detailpage\_o03\_s01?ie=UTF8&psc=1**](https://www.google.com/url?q=https://www.google.com/url?q%3Dhttps://www.amazon.com/gp/product/B01608II3Q/ref%253Doh_aui_detailpage_o03_s01?ie%253DUTF8%2526psc%253D1%26amp;sa%3DD%26amp;ust%3D1477184990061000%26amp;usg%3DAFQjCNHuSwTF9lRWxEb-zCN-XNy2eGt7uQ&sa=D&ust=1477184990074000&usg=AFQjCNFTNDjmNKmmFB9VLz8Qb4yDObQ5wg)

**1x - Buzzer -**

[**https://octopart.com/1536-adafruit+industries-61582119**](https://www.google.com/url?q=https://www.google.com/url?q%3Dhttps://octopart.com/1536-adafruit%252Bindustries-61582119%26amp;sa%3DD%26amp;ust%3D1477184990062000%26amp;usg%3DAFQjCNFm4peGaZDmLa8tgEDlzYLoiwA7VA&sa=D&ust=1477184990074000&usg=AFQjCNEhM_89GUcAlJ2j3Ua_zi4R5p94ng)

**6x - Female/Female Jumper Wires -**

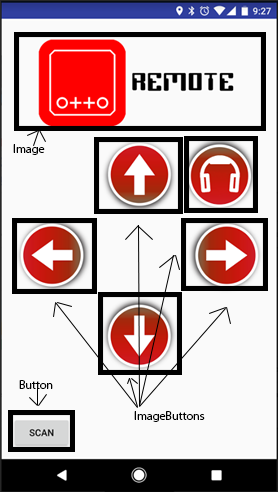
[**https://octopart.com/266-adafruit+industries-28486657**](https://www.google.com/url?q=https://www.google.com/url?q%3Dhttps://octopart.com/266-adafruit%252Bindustries-28486657%26amp;sa%3DD%26amp;ust%3D1477184990063000%26amp;usg%3DAFQjCNGFNb7EBEjWObLrkJKwIMXw-9AAGg&sa=D&ust=1477184990074000&usg=AFQjCNGbob6RAghFXr6efi41nj4VaVAr6A)

**4xAA Battery Holder -**

<http://www.ebay.com/itm/122100190919?_trksid=p2057872.m2749.l2649&ssPageName=STRK%3AMEBIDX%3AIT>

**Android Application Design**

Android applications are built with Java with some additional requirements needed to run on an Android device. Since the application requires user interaction, a GUI for the user is always required. These layout files are written in XML (Extensible Markup Language) and used by the Android application to display widgets for the user to interact with.



**activity\_otto\_remote.xml**

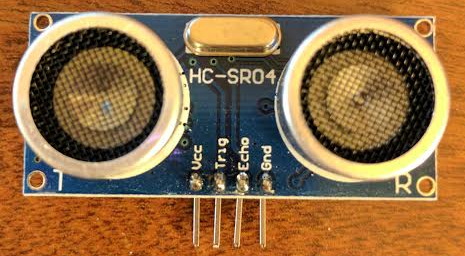
<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:focusable="true"  
 android:focusableInTouchMode="true"  
 android:paddingBottom="@dimen/activity\_vertical\_margin"  
 android:paddingLeft="@dimen/activity\_horizontal\_margin"  
 android:paddingRight="@dimen/activity\_horizontal\_margin"  
 android:paddingTop="@dimen/activity\_vertical\_margin"  
 tools:context=".OttoRemote"**>  
  
 <**Button  
 android:id="@+id/buttonScan"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_alignParentBottom="true"  
 android:layout\_alignParentLeft="true"  
 android:text="Scan"**/>  
  
 <**ImageButton  
 android:id="@+id/upButton"  
 android:layout\_width="@dimen/button\_width"  
 android:layout\_height="@dimen/button\_height"  
 android:layout\_below="@+id/logo"  
 android:layout\_centerHorizontal="true"  
 android:background="@drawable/up\_selector"  
 android:scaleType="centerCrop"**/>  
  
 <**ImageButton  
 android:id="@+id/rightButton"  
 android:layout\_width="@dimen/button\_width"  
 android:layout\_height="@dimen/button\_height"  
 android:layout\_alignParentRight="true"  
 android:layout\_below="@id/upButton"  
 android:background="@drawable/right\_selector"  
 android:scaleType="centerCrop"**/>  
  
 <**ImageButton  
 android:id="@+id/downButton"  
 android:layout\_width="@dimen/button\_width"  
 android:layout\_height="@dimen/button\_height"  
 android:layout\_below="@id/rightButton"  
 android:layout\_centerHorizontal="true"  
 android:background="@drawable/down\_selector"  
 android:scaleType="centerCrop"**/>  
  
 <**ImageButton  
 android:id="@+id/leftButton"  
 android:layout\_width="@dimen/button\_width"  
 android:layout\_height="@dimen/button\_height"  
 android:layout\_alignParentLeft="true"  
 android:layout\_below="@id/upButton"  
 android:background="@drawable/left\_selector"  
 android:scaleType="centerCrop"**/>  
  
 <**ImageButton  
 android:id="@+id/musicButton"  
 android:layout\_width="@dimen/button\_width"  
 android:layout\_height="@dimen/button\_height"  
 android:layout\_toRightOf="@id/upButton"  
 android:layout\_below="@id/logo"  
 android:background="@drawable/music\_selector"  
 android:scaleType="centerCrop"**/>  
  
 <**ImageView  
 android:id="@+id/logo"  
 android:layout\_width="300dp"  
 android:layout\_height="160dp"  
 android:layout\_alignParentTop="true"  
 android:layout\_centerHorizontal="true"  
 android:scaleType="centerInside"  
 android:src="@drawable/otto\_logo"**/>  
  
</**RelativeLayout**>

The company that produces the Bluno supplies a starter application that contains the necessary library files to connect and send serial data to the Bluno board. The java file, **OttoRemote.java**, connects the layout to the logic by using **OnClickListeners** to wait for users to interact with the, and then by extending the **BlunoLibrary** calls the method **serialSend(STRING),** that will send a character to the Bluno. Otto is then able to use a switch statement to act out a command based on the received character.

**OttoRemote.java**

**package** com.watersjournal.ottoremote;  
  
**import** android.content.Intent;  
**import** android.os.Bundle;  
**import** android.view.View;  
**import** android.view.View.OnClickListener;  
**import** android.widget.Button;  
**import** android.widget.EditText;  
**import** android.widget.ImageButton;  
**import** android.widget.TextView;  
  
**public class** OttoRemote **extends** BlunoLibrary {  
 **private** Button **buttonScan**;  
 **private** Button **buttonSerialSend**;  
 **private** EditText **serialSendText**;  
 **private** TextView **serialReceivedText**;  
 **private** ImageButton **buttonUp**;  
 **private** ImageButton **buttonDown**;  
 **private** ImageButton **buttonLeft**;  
 **private** ImageButton **buttonRight**;  
 **private** ImageButton **buttonMusic**;  
 **private long startTime**;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **startTime** = System.*currentTimeMillis*();  
  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_otto\_remote***);  
  
 onCreateProcess(); *//onCreate Process by BlunoLibrary* serialBegin(115200); *//set the Uart Baudrate on BLE chip to 115200  
  
 // Button to move Otto forward* **buttonUp** = (ImageButton) findViewById(R.id.***upButton***);  
 **buttonUp**.setOnClickListener(**new** OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **long** diff = System.*currentTimeMillis*() - **startTime**;  
 *// Only send the command if the last one was over a second ago  
 // To allow Otto to finish the movement* **if** (System.*currentTimeMillis*() > **startTime** + 1000) {  
 serialSend(**"F"**);  
 **startTime** = System.*currentTimeMillis*();  
 }  
 }  
 });  
  
 *// Button to move Otto backward* **buttonDown** = (ImageButton) findViewById(R.id.***downButton***);  
 **buttonDown**.setOnClickListener(**new** OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **long** diff = System.*currentTimeMillis*() - **startTime**;  
 *// Only send the command if the last one was over a second ago  
 // To allow Otto to finish the movement* **if** (System.*currentTimeMillis*() > **startTime** + 1000) {  
 serialSend(**"B"**);  
 **startTime** = System.*currentTimeMillis*();  
 }  
 }  
 });  
  
 *// Button to move Otto left* **buttonLeft** = (ImageButton) findViewById(R.id.***leftButton***);  
 **buttonLeft**.setOnClickListener(**new** OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **long** diff = System.*currentTimeMillis*() - **startTime**;  
 *// Only send the command if the last one was over a second ago  
 // To allow Otto to finish the movement* **if** (System.*currentTimeMillis*() > **startTime** + 1000) {  
 serialSend(**"L"**);  
 **startTime** = System.*currentTimeMillis*();  
 }  
 }  
 });  
  
 *// Button to move Otto right* **buttonRight** = (ImageButton) findViewById(R.id.***rightButton***);  
 **buttonRight**.setOnClickListener(**new** OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **long** diff = System.*currentTimeMillis*() - **startTime**;  
 *// Only send the command if the last one was over a second ago  
 // To allow Otto to finish the movement* **if** (System.*currentTimeMillis*() > **startTime** + 1000) {  
 serialSend(**"R"**);  
 **startTime** = System.*currentTimeMillis*();  
 }  
 }  
 });  
  
 *// Button to have Otto play some music* **buttonMusic** = (ImageButton) findViewById(R.id.***musicButton***);  
 **buttonMusic**.setOnClickListener(**new** OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **long** diff = System.*currentTimeMillis*() - **startTime**;  
 **if** (System.*currentTimeMillis*() > **startTime** + 1000) {  
 serialSend(**"M"**);  
 **startTime** = System.*currentTimeMillis*();  
 }  
 }  
 });  
  
 *//initial the button for scanning the BLE device* **buttonScan** = (Button) findViewById(R.id.***buttonScan***);   
 **buttonScan**.setOnClickListener(**new** OnClickListener() {  
  
 @Override  
 **public void** onClick(View v) {  
 buttonScanOnClickProcess();*//Alert Dialog for selecting the BLE device* }  
 });  
 }  
  
 **protected void** onResume(){  
 **super**.onResume();  
 System.***out***.println(**"BlUNOActivity onResume"**);  
 onResumeProcess(); *//onResume Process by BlunoLibrary* }  
  
 @Override  
 **protected void** onActivityResult(**int** requestCode, **int** resultCode, Intent data) {  
 onActivityResultProcess(requestCode, resultCode, data); *//onActivityResult Process by BlunoLibrary* **super**.onActivityResult(requestCode, resultCode, data);  
 }  
  
 @Override  
 **protected void** onPause() {  
 **super**.onPause();  
 onPauseProcess(); *//onPause Process by BlunoLibrary* }  
  
 **protected void** onStop() {  
 **super**.onStop();  
 onStopProcess(); *//onStop Process by BlunoLibrary* }  
  
 @Override  
 **protected void** onDestroy() {  
 **super**.onDestroy();  
 onDestroyProcess(); *//onDestroy Process by BlunoLibrary* }  
  
 *//Once connection state changes, this function will be called* @Override  
 **public void** onConectionStateChange(connectionStateEnum theConnectionState) {  
 **switch** (theConnectionState) { *//Four connection states* **case *isConnected***:  
 **buttonScan**.setText(**"Connected"**);  
 **break**;  
 **case *isConnecting***:  
 **buttonScan**.setText(**"Connecting"**);  
 **break**;  
 **case *isToScan***:  
 **buttonScan**.setText(**"Scan"**);  
 **break**;  
 **case *isScanning***:  
 **buttonScan**.setText(**"Scanning"**);  
 **break**;  
 **case *isDisconnecting***:  
 **buttonScan**.setText(**"isDisconnecting"**);  
 **break**;  
 **default**:  
 **break**;  
 }  
 }  
  
 *//Once connection data received, this function will be called* @Override  
 **public void** onSerialReceived(String theString) {   
  
 }  
}

**Using Ultrasonic Sensor for Avoidance**



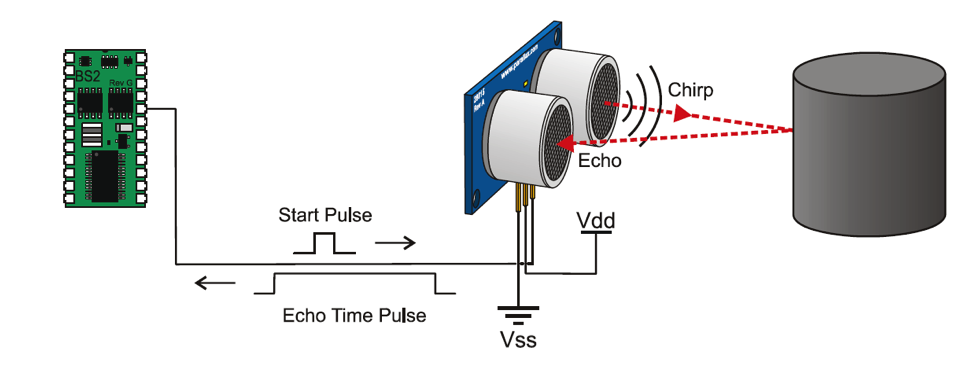
**What is an Ultrasonic Sensor?**

**Active ultrasonic sensors** generate high-frequency sound waves and evaluate the echo which is received back by the sensor, measuring the time interval between the sending signal and the receiving the echo is used to determine the distance to an object.

For more information see:

https://en.wikipedia.org/wiki/Ultrasonic\_transducer

**Demonstration of an ultrasonic sensor.**



An ultrasonic sensor works just like how a bat or dolphin uses echo location. When the “Trig” side of the sensor receives 5Volts (HIGH), it begins to send out sound waves that are too high for humans to hear. After a very short period, about 10 microseconds, the sound waves are shut off. This tells the sensor to now begin to hear for the return of the sound waves through the “Echo” side of the sensor.

Arduino has a built-in function called **pulseIn** that will give the exact time difference between when the function is called and when the designated pin turns either HIGH or LOW. This calculated time is then used to calculate the distance traveled by the sound waves, using the simple formula, **Distance = Time \* Velocity (Of Sound Waves).**

**Code:**

**safeDistance Method**

**bool safeDistance() {**

**long duration, distance;**

**digitalWrite(trigPin, LOW); // Make sure not sending any sound**

**delayMicroseconds(2); // Clears the speaker for 2 microseconds**

**digitalWrite(trigPin, HIGH); // Start sound waves**

**delayMicroseconds(10); // Run for 10 microseconds**

**digitalWrite(trigPin, LOW); // Turn off sound waves**

**duration = pulseIn(echoPin, HIGH); // Set the echo pin to wait till receive 5V**

**distance = (duration/2) \* 0.03435; // Calculate the distance by formula**

**// (Distance = Speed \* Time)**

**// Time / 2 because do not want the time from // the object and back**

**// We only want the time to the object.**

**// 0.03435 is the speed of sound in CM/S^2**

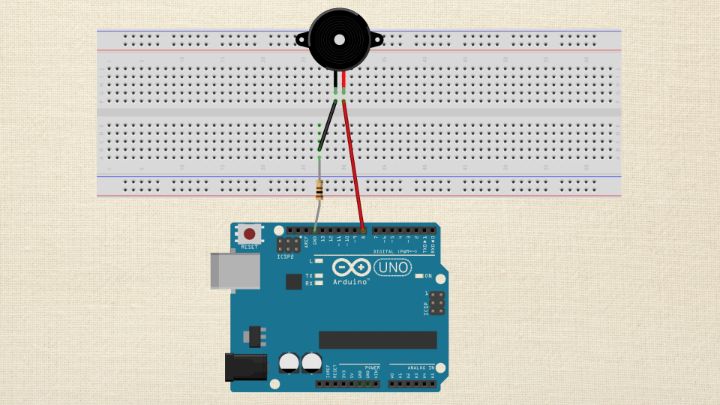
**return (distance > 5);**

**}**

**Tone**

**Hardware:**

Piezo Speaker: uses a material that’s piezoelectric, it can changes shape when you apply electric to it. By adhering a piezo-electric disc to a thin metal plate, and the applying electricity, we can bend the metal back and forth, which in turn creates noise.



Otto robot (already build in)

**Tone() function**

**Syntax**

Tone needs 2 arguments, but it can take 3

Tone(pin, frequency)

Tone(pin, frequency, duration)

**Parameters**

\*Pin Number: the pin that to geberate the tone

\*Frequency: it is specified in hertz. Hertz are cycles per second. Unsigned int

\*Duration: how long you want a tone to play. It is in milliseconds.

**Return**

Nothing

Here is an example of tone() function

Tone(10, 1000, 500)

Pin number: 10

Frequency: 1000

Duration: 500

**Notes:** The frequency is an unsigned integer and can take value up to 65,535-but if you are trying to make tones for a human ear, then values between 2000-5000 are where our ears are most tunes.

If you want to break a tone you can use **delay() function**

Tone() function uses one of the built in times on the Arduino’s micro-controller.

Tone() function works independently of the delay() function.

**Tone() function limits**

Cannot generate a tone lower than 31hz. You can pass values 31 and less to the tone() function, but it does not mean you will get a good representation of it.

Tone() function cannot be used by two separated pins at the same time

void playMusic() {

for(int i=0; i < MAX\_COUNT; i++) {

tone\_ = melody[i];

beat = 50;

duration = beat \* tempo; // Set up timing

playTone();

// A pause between notes...

delayMicroseconds(pause);

}

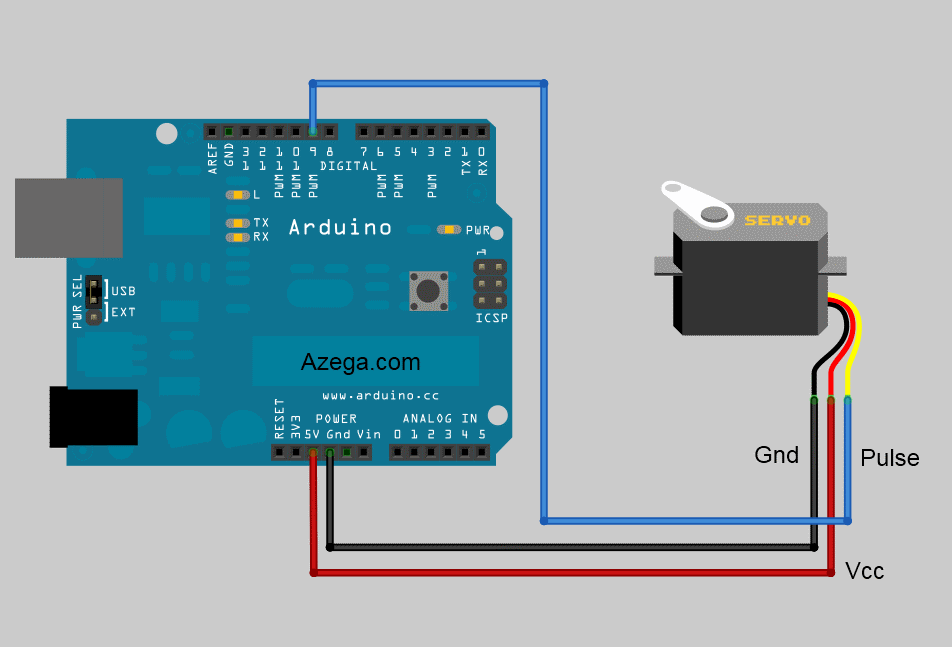
}

**Servomotors for Walking**

What is a Servomotor?

A servo is motor that has a rotor that can be controlled between 0 and 180 degrees. The precise angle, velocity and acceleration of the servomotor can be defined. Our servomotors are attached via three wires: power (Vcc), ground (Gnd) and signal (Pulse). The functions provided in the default Arduino library to pair the motor are attach(), write(), detach().

**Demonstration of Servomotor wiring configuration**



The attach() and detach() function pairs one servo motor with a signal pin on the microcontroller. When the pin outputs a signal, it sends a sinusoidal wave through the signal wire. The program written adjusts the 4 dimensions of a wave:

Amplitude (degrees), Period (mS) , Phase (radians) and Offset (degrees)

A signal emitted from the microcontroller tells each servo what angle the servo should move, in what order and for how long. Each servomotor represents one joint in robot’s leg. The collective effort of the four servomotors propels the robot in the desired direction.

**Otto\_Arduino.ino**

**#include <Servo.h>**

**#include "Oscillator.h"**

**#include <EEPROM.h>**

**#define N\_SERVOS 4**

**//-- First step: Configure the pins where the servos are attached**

**/\***

**---------------**

**| O O |**

**|---------------|**

**YR 3==> | | <== YL 2**

**---------------**

**|| ||**

**|| ||**

**RR 5==> ----- ------ <== RL 4**

**|----- ------|**

**\*/**

**#define TRIM\_RR 7**

**#define TRIM\_RL 4**

**#define TRIM\_YR 4**

**#define TRIM\_YL -7**

**//OTTO.setTrims(-7,-4,-4,7);**

**#define PIN\_RR 5**

**#define PIN\_RL 4**

**#define PIN\_YR 3**

**#define PIN\_YL 2**

**#define C 2100**

**#define D 1870**

**#define E 1670**

**#define f 1580**

**#define G 1400**

**#define R 0**

**#define INTERVALTIME 10.0**

**Oscillator servo[N\_SERVOS];**

**const int trigPin = 9;**

**const int echoPin = 8;**

**const int speakerOut = 10;**

**// function prototypes**

**void turnLeft(int steps, int T=3000);**

**void turnRight(int steps, int T=3000);**

**void walk(int steps, int T=1000);**

**void backyard(int steps, int T=3000);**

**bool safeDistance();**

**void playMusic();**

**int melody[] = {E,E,E,R,E,E,E,R,E,G,C,D,E,R,f,f,f,f,f,E,E,E,E,D,D,E,D};**

**int MAX\_COUNT = sizeof(melody) / 2; // Melody length, for looping.**

**// Set overall tempo**

**long tempo = 7500;**

**// Set length of pause between notes**

**int pause = 1000;**

**// Loop variable to increase Rest length**

**int rest\_count = 70;**

**// Initialize core variables**

**int tone\_ = 0;**

**int beat = 0;**

**long duration = 0;**

**void setup() {**

**// put your setup code here, to run once:**

**Serial.begin(115200);**

**servo[0].attach(PIN\_RR);**

**servo[1].attach(PIN\_RL);**

**servo[2].attach(PIN\_YR);**

**servo[3].attach(PIN\_YL);**

**servo[0].SetTrim(TRIM\_RR);**

**servo[1].SetTrim(TRIM\_RL);**

**servo[2].SetTrim(TRIM\_YR);**

**servo[3].SetTrim(TRIM\_YL);**

**pinMode(trigPin, OUTPUT);**

**pinMode(echoPin, INPUT);**

**pinMode(speakerOut, OUTPUT);**

**for(int i=0;i<4;i++) servo[i].SetPosition(90);**

**}**

**// TEMPO: 121 BPM**

**int t=495;**

**void loop() {**

**// put your main code here, to run repeatedly:**

**if(Serial.available()) {**

**char character = Serial.read();**

**switch(character) {**

**case 'F':**

**Serial.println("MOVING OTTO UP");**

**if (safeDistance()) {**

**walk(2, t\*2);**

**} else {**

**tone(10, 440, 1000);**

**}**

**break;**

**case 'B':**

**Serial.println("MOVING OTTO DOWN");**

**tone(13, 800, 100);**

**back(2, t\*2);**

**break;**

**case 'L':**

**Serial.println("MOVING OTTO LEFT");**

**turnLeft(2, t\*2);**

**break;**

**case 'R':**

**Serial.println("MOVING OTTO RIGHT");**

**turnRight(2, t\*2);**

**break;**

**case 'M':**

**Serial.println("PLAYING MUSIC");**

**playMusic();**

**break;**

**}**

**}**

**}**

**void oscillate(int A[N\_SERVOS], int O[N\_SERVOS], int T, double phase\_diff[N\_SERVOS]){**

**for (int i=0; i<4; i++) {**

**servo[i].SetO(O[i]);**

**servo[i].SetA(A[i]);**

**servo[i].SetT(T);**

**servo[i].SetPh(phase\_diff[i]);**

**}**

**double ref=millis();**

**for (double x=ref; x<T+ref; x=millis()){**

**for (int i=0; i<4; i++){**

**servo[i].refresh();**

**}**

**}**

**}**

**void back(int steps, int T){**

**int A[4]= {15, 15, 30, 30};**

**int O[4] = {0, 0, 0, 0};**

**double phase\_diff[4] = {DEG2RAD(0), DEG2RAD(0), DEG2RAD(90), DEG2RAD(90)};**

**for(int i=0;i<steps;i++)oscillate(A,O, T, phase\_diff);**

**}**

**void walk(int steps, int T){**

**int A[4]= {15, 15, 30, 30};**

**int O[4] = {0, 0, 0, 0};**

**double phase\_diff[4] = {DEG2RAD(0), DEG2RAD(0), DEG2RAD(-90), DEG2RAD(-90)};**

**for(int i=0;i<steps;i++)oscillate(A,O, T, phase\_diff);**

**}**

**void turnLeft(int steps, int T){**

**int A[4]= {20, 20, 10, 30};**

**int O[4] = {0, 0, 0, 0};**

**double phase\_diff[4] = {DEG2RAD(0), DEG2RAD(0), DEG2RAD(90), DEG2RAD(90)};**

**for(int i=0;i<steps;i++)oscillate(A,O, T, phase\_diff);**

**}**

**void turnRight(int steps, int T){**

**int A[4]= {20, 20, 30, 10};**

**int O[4] = {0, 0, 0, 0};**

**double phase\_diff[4] = {DEG2RAD(0), DEG2RAD(0), DEG2RAD(90), DEG2RAD(90)};**

**for(int i=0;i<steps;i++)oscillate(A,O, T, phase\_diff);**

**}**

**bool safeDistance() {**

**long duration, distance;**

**digitalWrite(trigPin, LOW); // Make sure not sending any sound**

**delayMicroseconds(2); // Clears the speaker for 2 microseconds**

**digitalWrite(trigPin, HIGH); // Start sound waves**

**delayMicroseconds(10); // Run for 10 microseconds**

**digitalWrite(trigPin, LOW); // Turn off sound waves**

**duration = pulseIn(echoPin, HIGH); // Set the echo pin to wait till receive 5V**

**distance = (duration/2) \* 0.03435; // Calculate the distance by formula (Distance = Speed \* Time)**

**// Time / 2 because do not want the time from the object and back**

**// We only want the time to the object.**

**// 0.03435 is the speed of sound in CM/S^2**

**return (distance > 5);**

**}**

**void playTone()**

**{**

**long elapsed\_time = 0;**

**if(tone\_ > 0) { // if this isn't a Rest beat, while the tone has**

**// played less long than 'duration', pulse speaker HIGH and LOW**

**while(elapsed\_time < duration) {**

**digitalWrite(speakerOut,HIGH);**

**delayMicroseconds(tone\_ / 2);**

**// DOWN**

**digitalWrite(speakerOut, LOW);**

**delayMicroseconds(tone\_ / 2);**

**// Keep track of how long we pulsed**

**elapsed\_time +=(tone\_);**

**}**

**}**

**else**

**{**

**// Rest beat; loop times delay**

**for(int j = 0; j < rest\_count; j++)**

**{**

**delayMicroseconds(duration);**

**}**

**}**

**}**

**void playMusic() {**

**for(int i=0; i < MAX\_COUNT; i++) {**

**tone\_ = melody[i];**

**beat = 50;**

**duration = beat \* tempo; // Set up timing**

**playTone();**

**// A pause between notes...**

**delayMicroseconds(pause);**

**}**

**}**

**Conclusion:**

In this project, it was a totally new experience for us all. None of us had worked with Arduino boards before, and we all had a limited understanding of the electrical circuitry. The online guide was a big help to get us started with the prefabricated 3D models and electrical wiring diagrams. Ultimately, we wanted to alter the design with our code by adding additional features.

We learned several things in creation of this robot including:

1. We learned how servos worked and how to control them by sending different electrical signals to drive the motors.
2. We learned how to create an Android app, and how to work with the Bluno libraries that were included for this Arduino board. We also learned how to control an Arduino through a Bluetooth serial interface.
3. We learned how an Arduino functions and some understanding of electrical circuits.

**Things that we would have done differently:**

1. Better calibrate his movements so that he moves better.
2. Adding additional hardware to Otto like a power switch or lights.

**Issues that we encountered:**

1. Making the pre-designed pieces work was an issue because they did not fit perfectly. This took a lot longer than expected, but we got them to fix after altering them.
2. Biggest issue at the beginning was that the leg model that was provided did not fit the servos that we purchased. This was a slow process because 3D printing each leg took around 30 minutes for each leg. Eventually, we had to design our own leg to make it work correctly.
3. One of the ultrasonic sensors did not function correctly, and we originally thought that our code was wrong. Only by testing with a known good sensor did we finally realize that the sensor was actually bad.