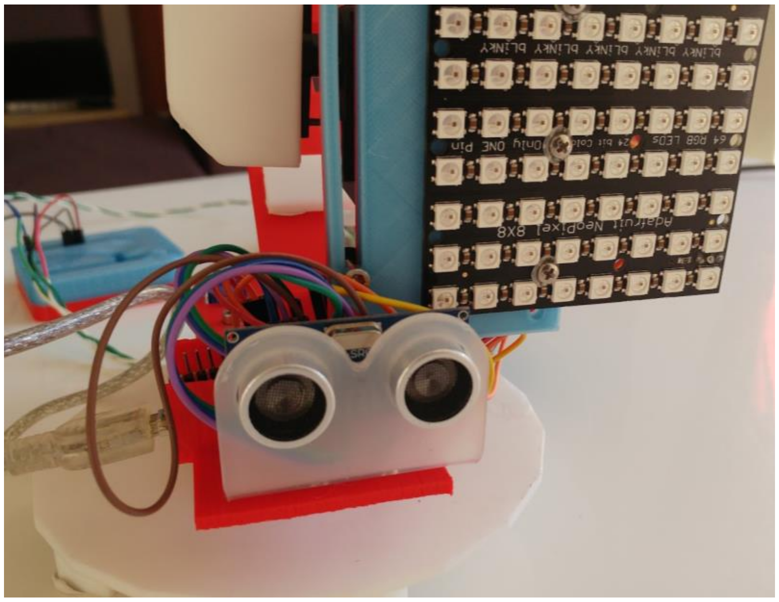
RPL - Computer Engineering. UAB (Barcelona) 2016-2017



Smart Arduino Turret

Automated arduino based catapult

PROJECT SPRINT #6.
DATE: 04 June 2017

Vernon Stanley Albayeros Duarte

Contenido

Sprint #2
Date: 06 May 2017

Smart Arduino Turret

Automated arduino based catapult

I. Project description

This is an Arduino-based stationary robot that scans for targets within a short radius and shoots projectiles. The project uses more than one Arduino boards connected through the I2C protocol to handle various tasks. The robot's chassis is completely 3D printed. The STL files for the parts can be found on the project's thingiverse page here [link]. I also printed a case for the Arduino uno made by thingiverse user ZygmuntW, the files for this case can be found here. The code is hosted in my github repository [link], along the instructions to set your own turret.

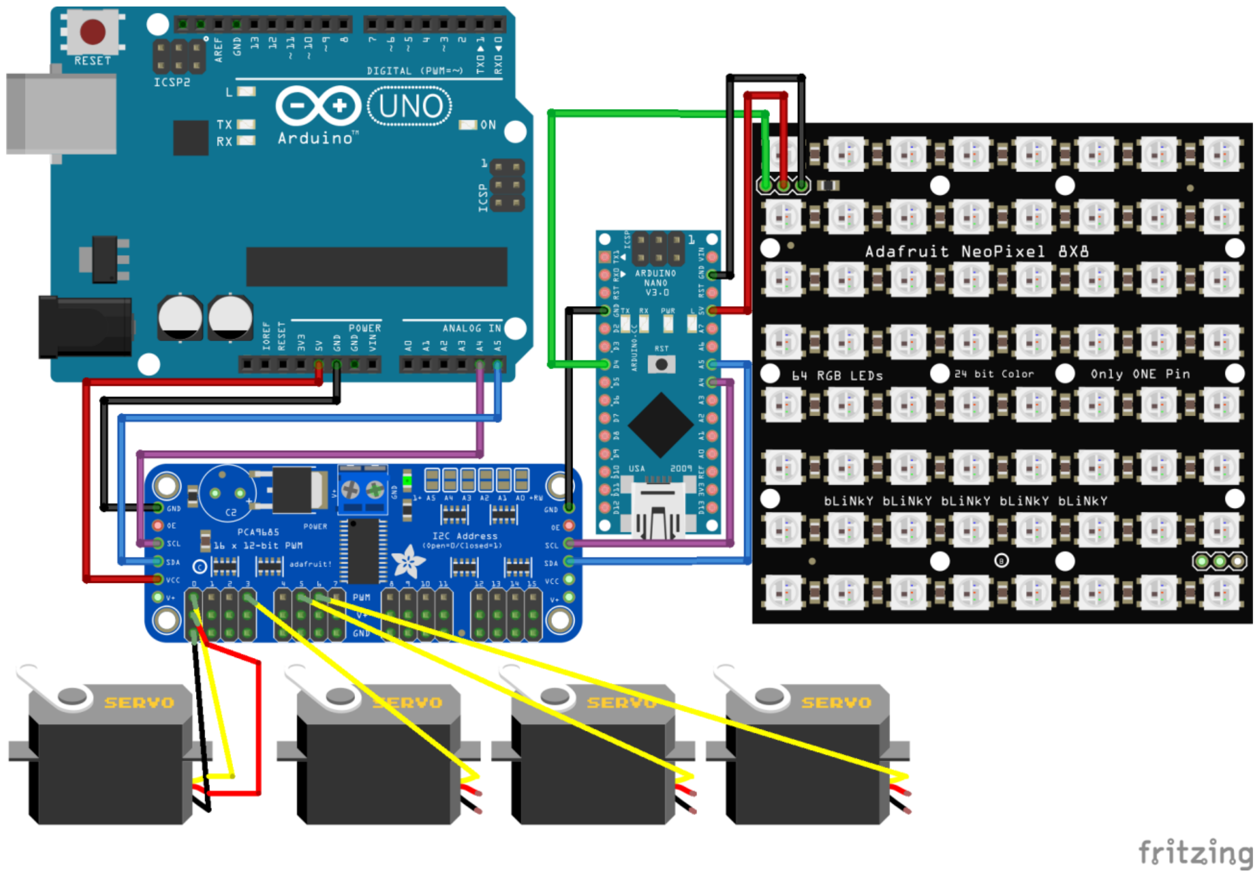
II. Electronic components

This is the list of the used components:

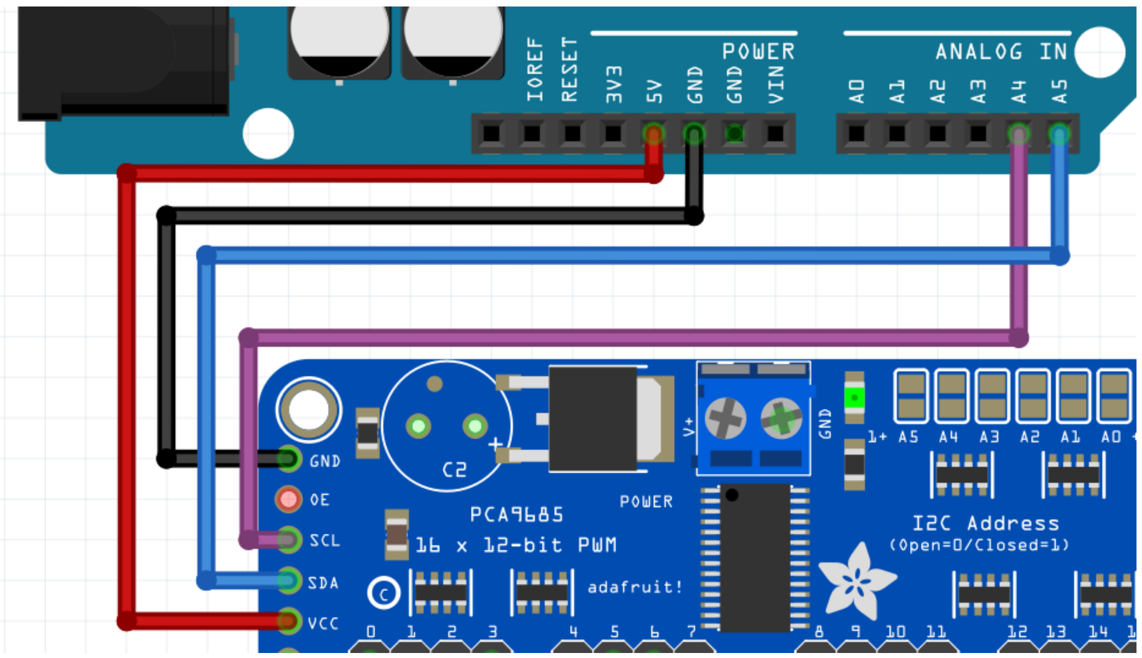
- Arduino Nano
- Arduino UNO R3
- HC-SR04 ultrasonic sensor
- MG996 Servomotor (2x)
- SG90 Servomotor (2x)
- 8x8 WS2812B LED matrix
- M3 x 10 screws
- M3 x 15 screws
- M3 nuts
- M3 brass nuts (for embedding in the larger holes on the base)
- M2 x 10 screws (to attach the servomotors to the arm, base and tray)
- 5V power source capable of giving out more than 4A

III. Scheme

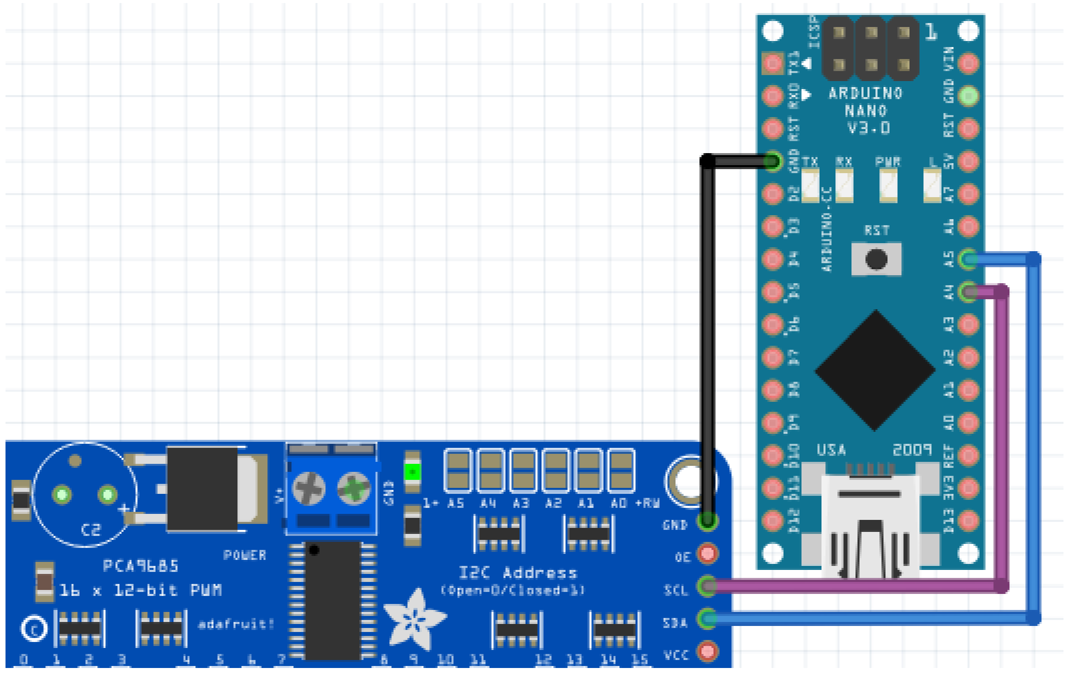
- Global scheme. Servomotor connections have been simplified to only show the signal wire to reduce visual clutter in the picture. All of the servomotors should have their GND and VIN signals plugged into their respective slots in the PWM driver.



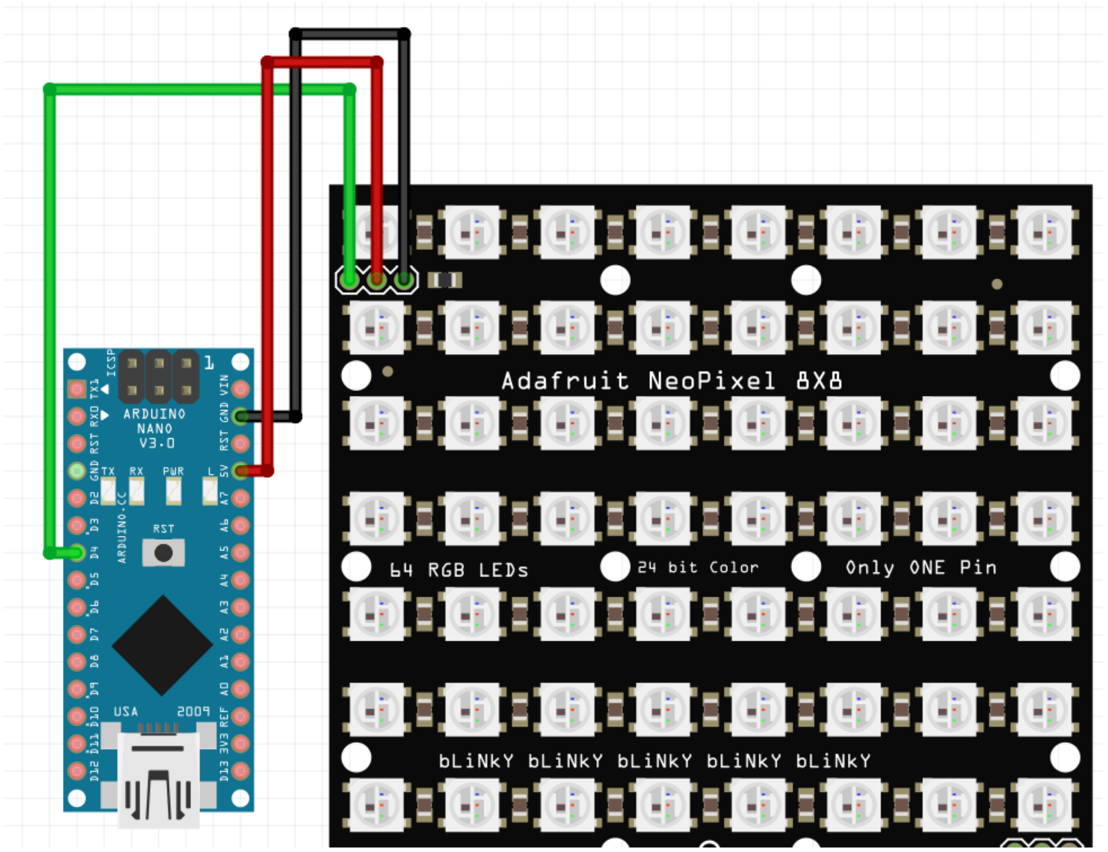
- Arduino Uno – 16 channel PWM driver pinout



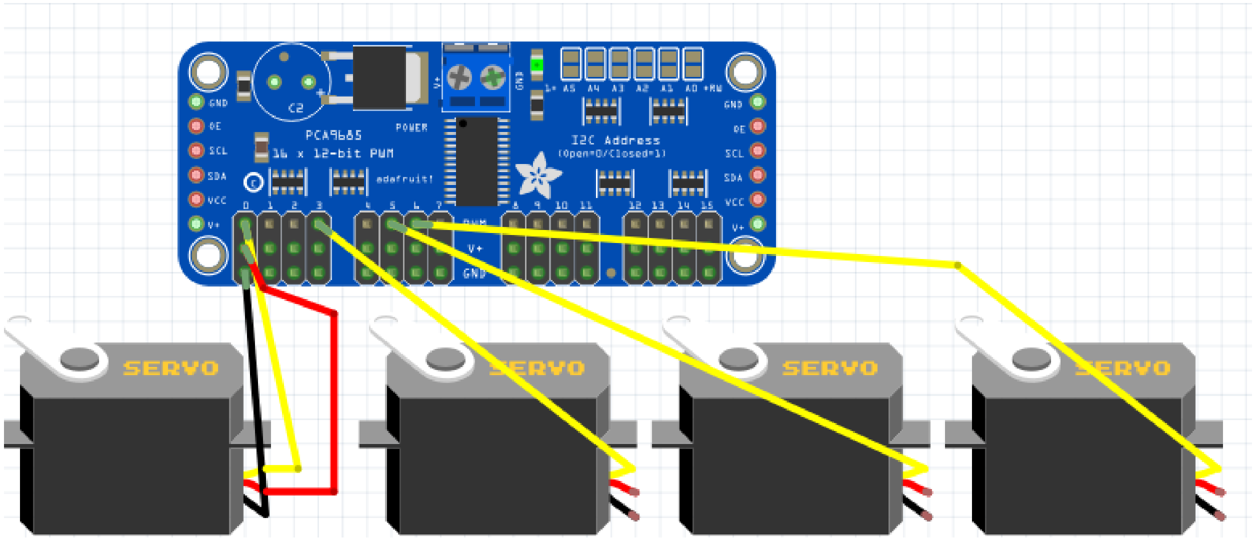
- Arduino nano – 16 channel PWM driver pinout



- Arduino Nano – LED matrix pinout



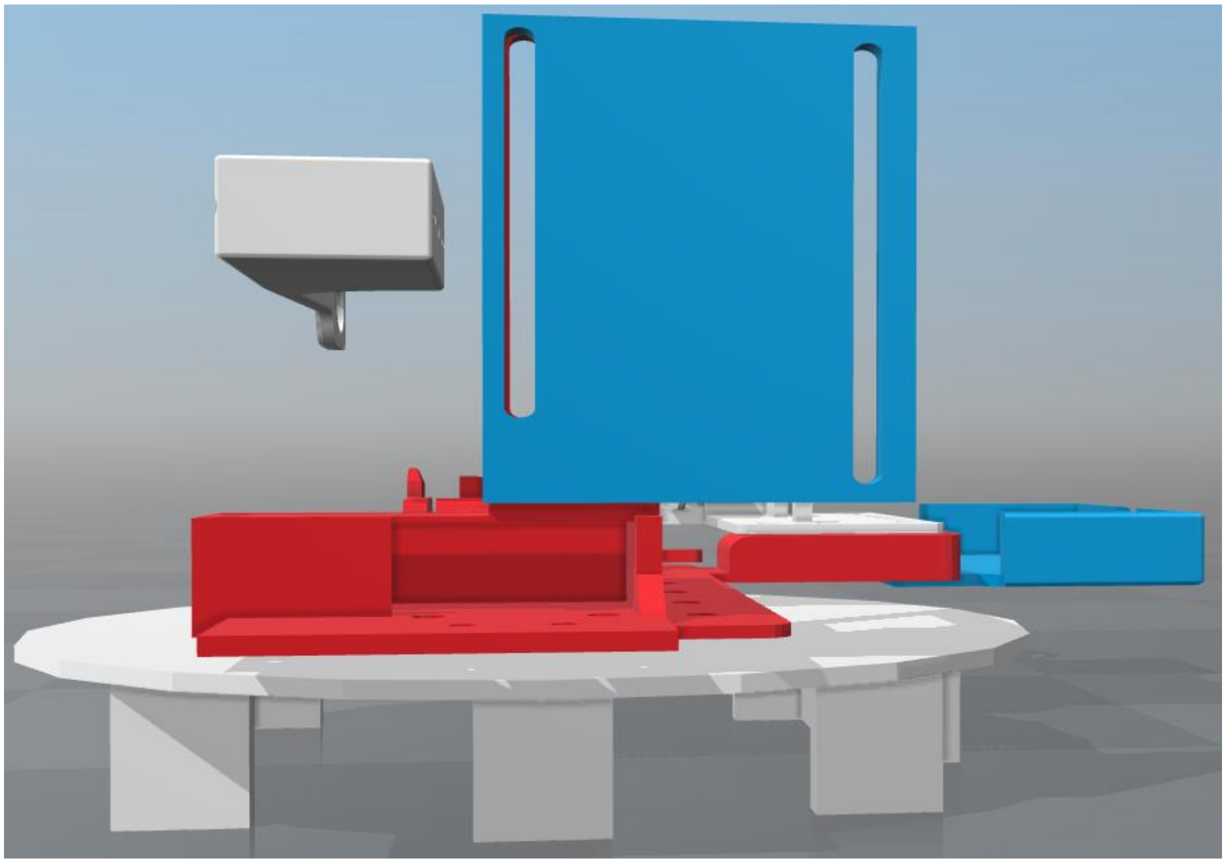
- 16 channel PWM driver – servomotors pinout. The first servomotor is how they all should be properly wired into the pins. The rest are not fully wired to reduce visual clutter.



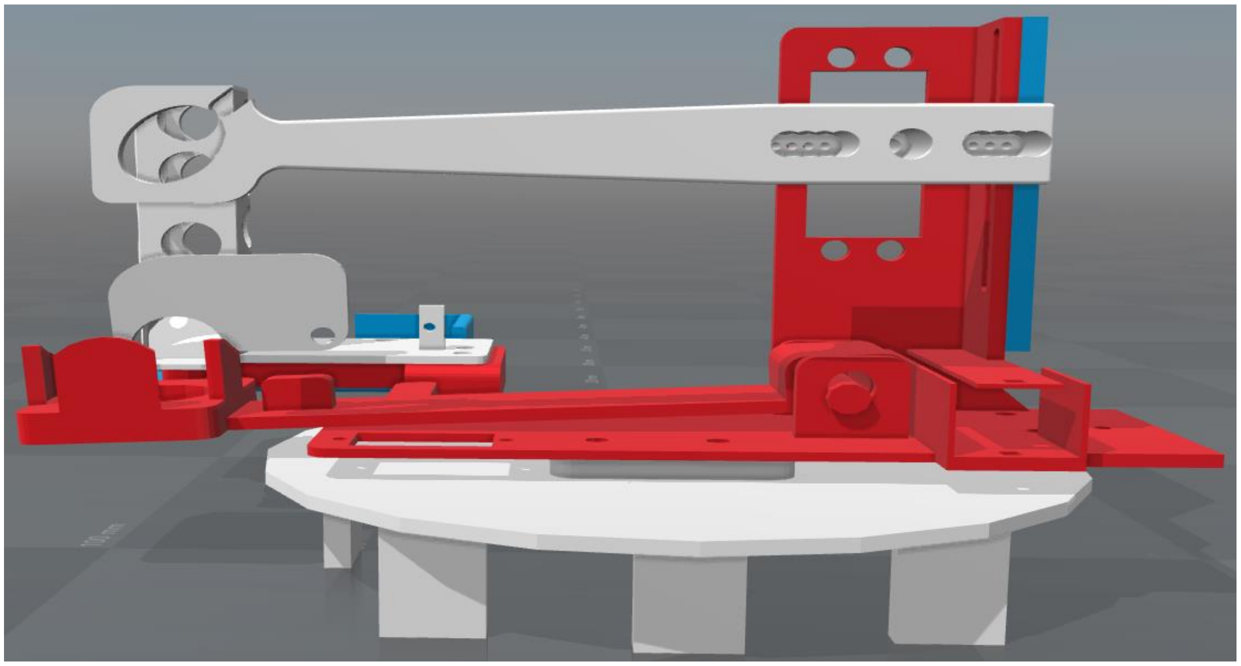
IV. 3D pieces

Overview

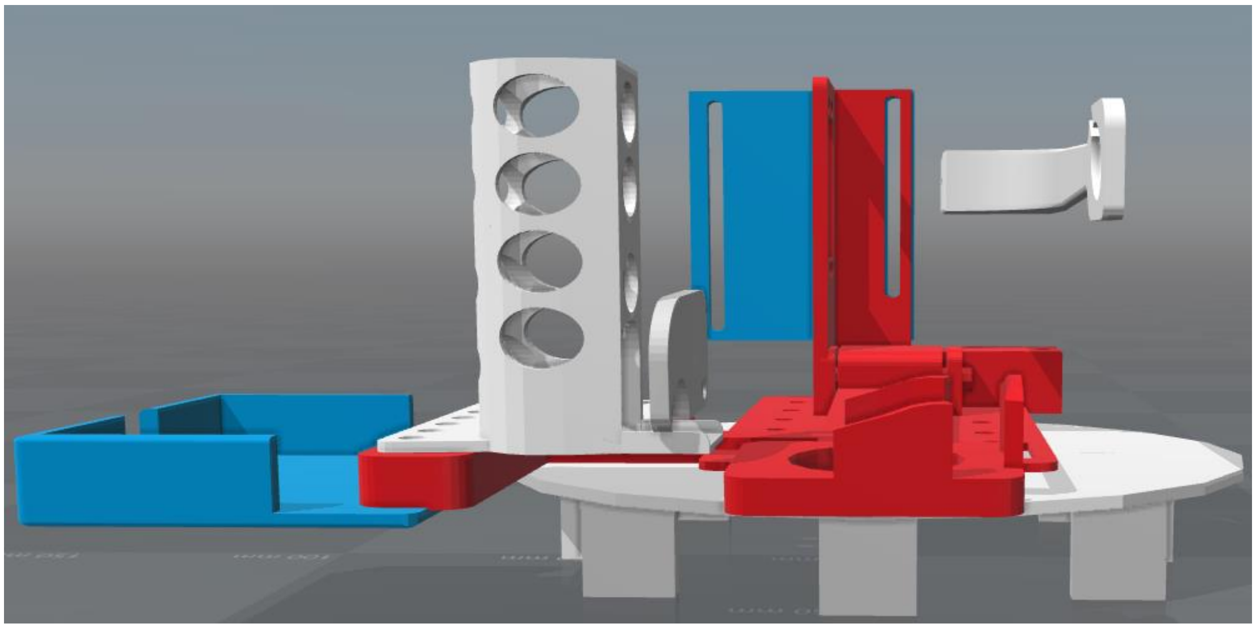
- Front



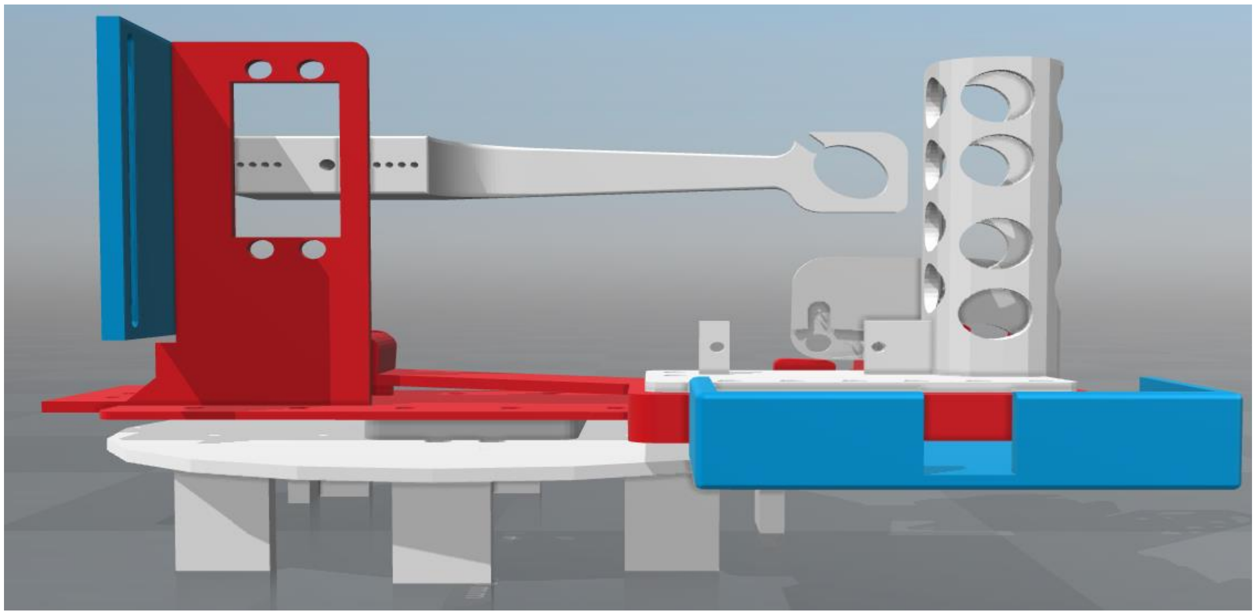
- Arm side



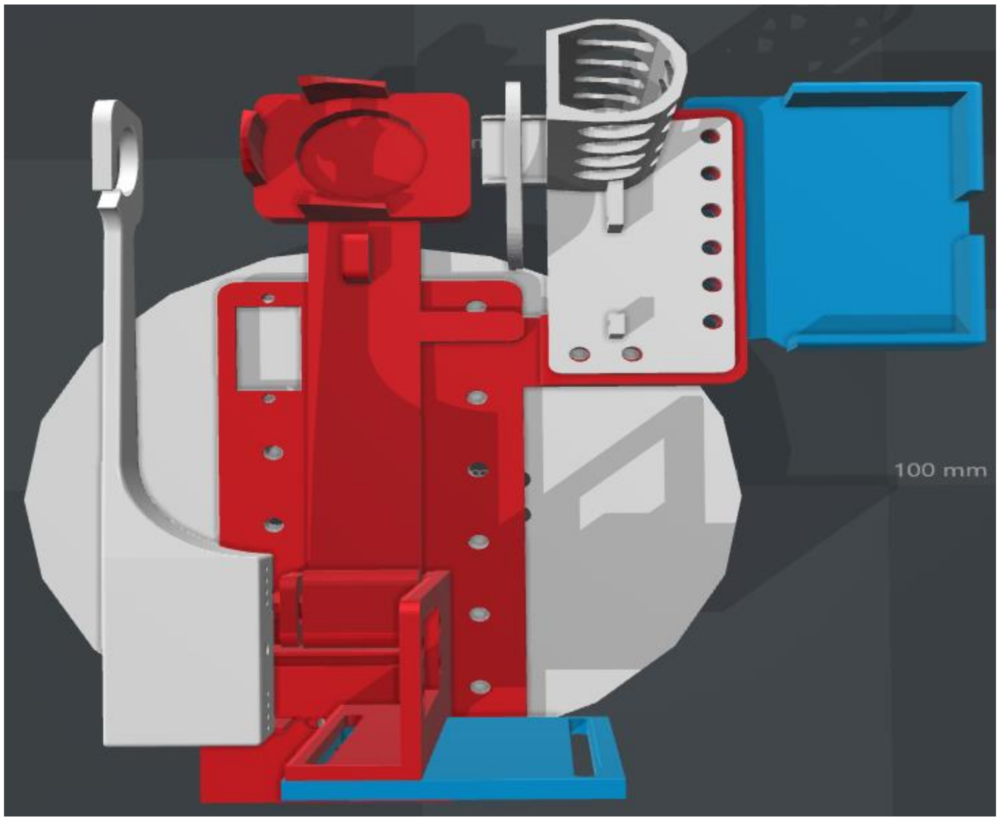
- Back



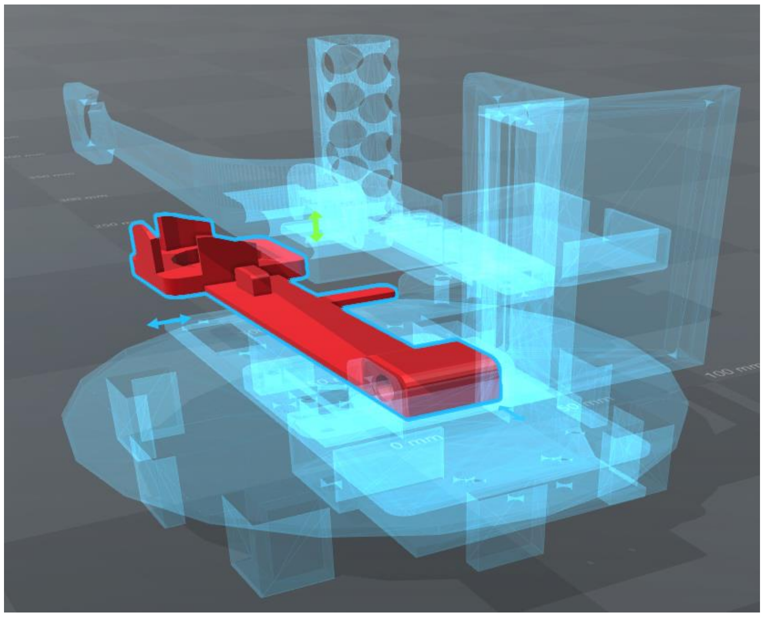
- PWM Driver side



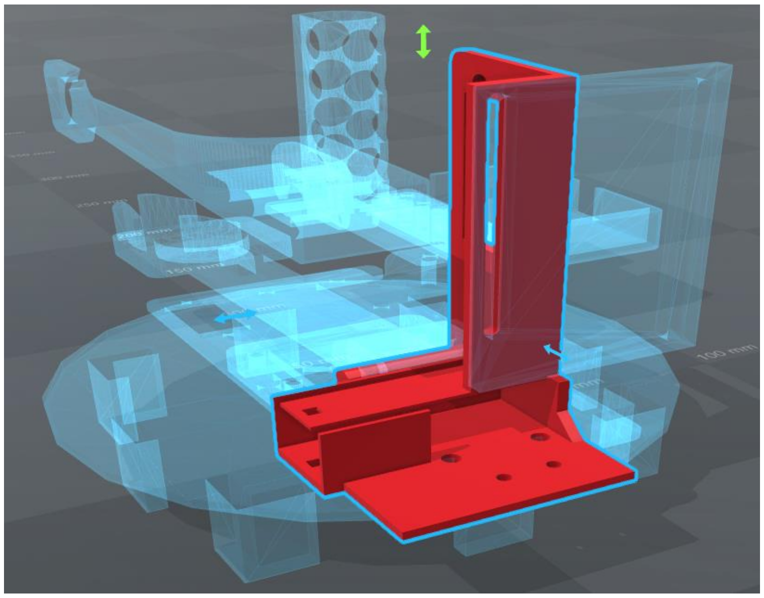
- Top



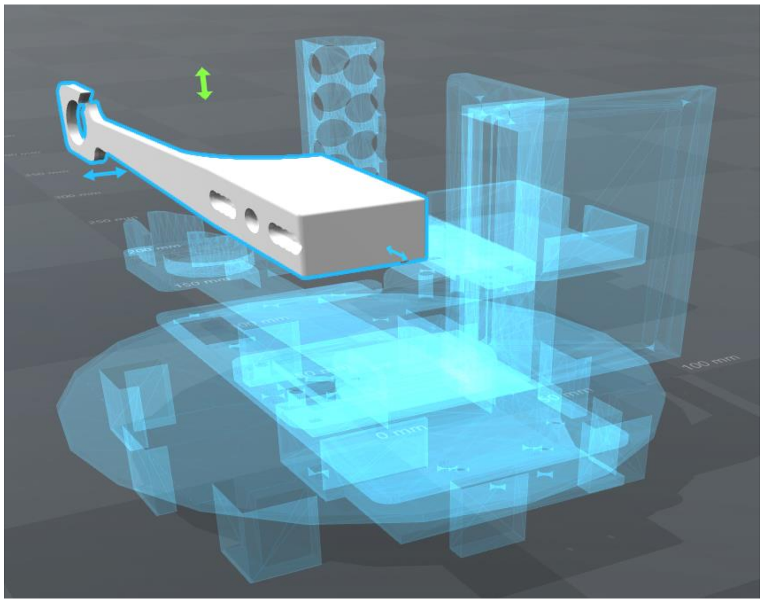
- **Arm:** Holds the ball and fires it.



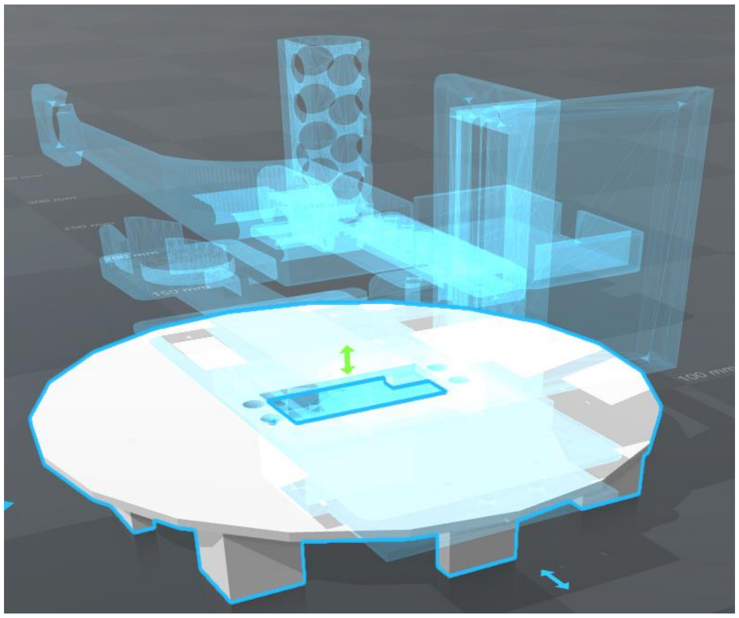
- **Arm motor base:** Holds one of the MG99X motors, the Arduino Nano and the HC-SR04 sensor.



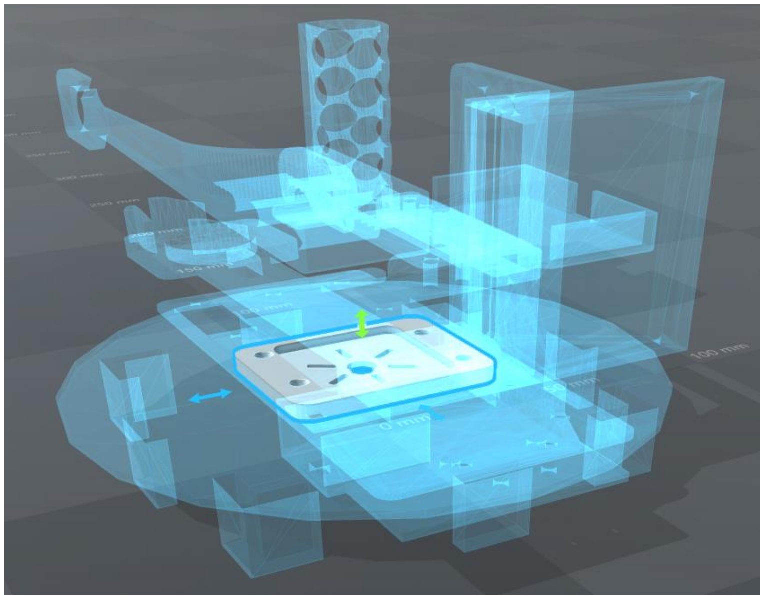
- **Arming arm:** Used to arm the catapult. The hook holds the rubber band that tenses the arm.



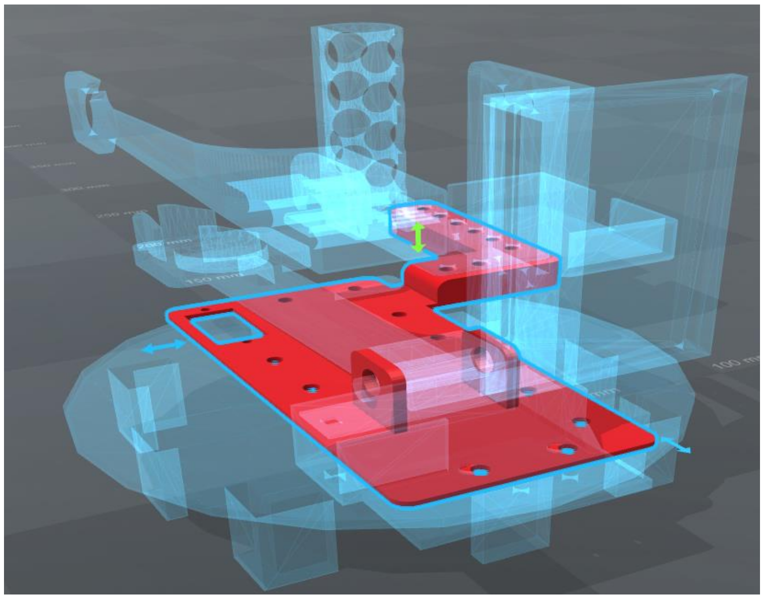
- **Base:** Holds the other MG99X servomotor to spin the platform.



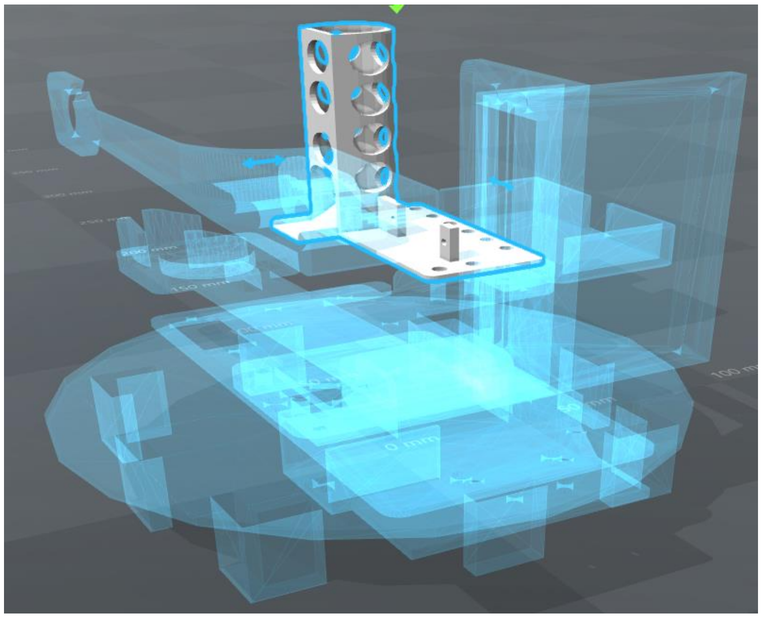
- **Base motor mount:** A small adaptor for the star shaped pieces that usually come with the MG99X series servomotors. Meant to couple the motor shaft with the plate.



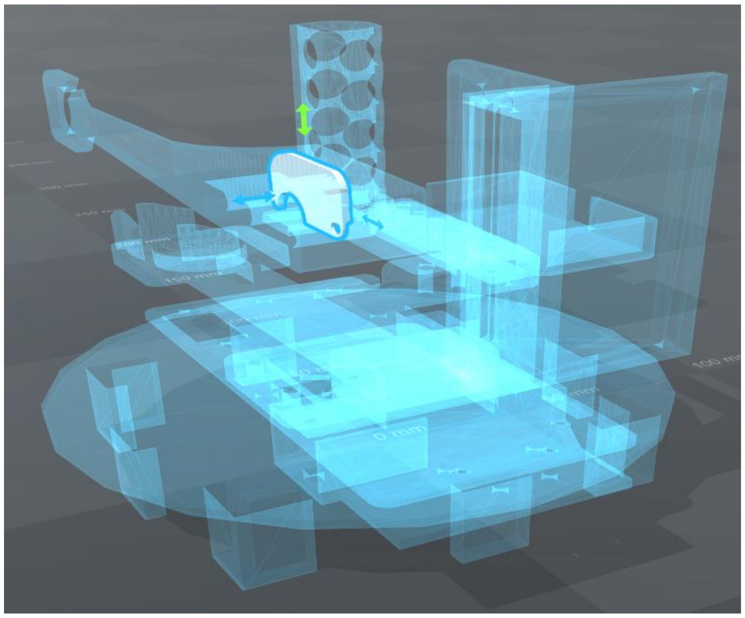
- **Plate:** This piece holds the arm, the arm motor base, and the tray of the catapult.



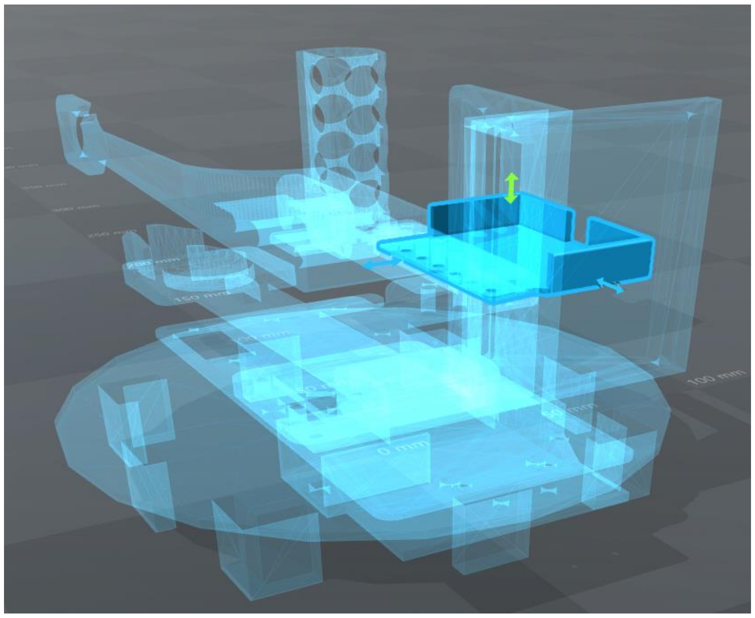
- **Tray:** holds the balls and the feeding mechanism (one SG90 servo).



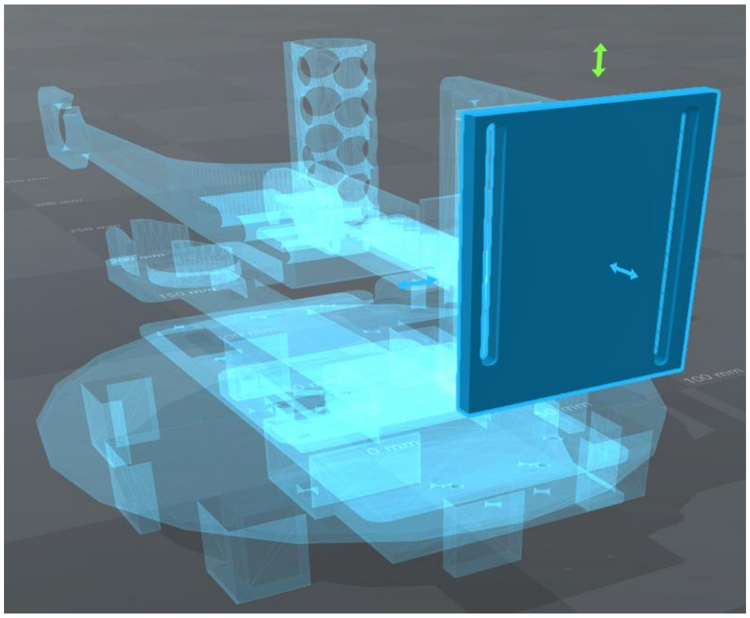
- **Tray gate:** Attaches to a "+" shaft adapter for the SG90 servomotor. They usually come with this part.



- **16 channel I2C PWM driver mount:** There are no holes for the standoff holes of the PWM board, I attached it with a couple of balls of non-conductive sticky putty.



- **LED matrix spacer:** This is where you should mount the LED matrix. This part is separated from the base motor mount because it's easier to print them this way.



V. I2C

I2C (Inter-Integrated Circuit) is a multi-master, multi-slave, packet switched serial bus used to create a means of communication between low speed controllers. It uses just two open-drain wires: a Serial Clock Line (SCL) and a Serial Data Line (SDA). When used between Arduino-based hardware, the voltage of the bus is usually +5V. The Wire.h Arduino library supports up to 112 unique devices connected to the I2C bus, although several workarounds can be done to "skip" this limit, like having a demultiplexer transmit from a single address to several pieces of hardware.

The role of the master(s) in the I2C protocol is to generate the clock signal and initiate the communication with the slaves. The address of the master in the I2C bus will always be 0. In the Arduino IDE, this is done by including the Wire.h library and calling the Wire.begin() method without an argument. Whatever Arduino board that executes this line will become a master of its I2C bus.

The slave(s) in the I2C protocol simply receive the clock signal and responds to whatever the master asks them to do. A Master device can send data to a slave or send a data petition to which a slave must respond by transmitting data to the master. A slave device is not able to create a petition by it's own. Being a packet-switched bus means the data is transmitted in packages.

In this project, the Arduino Uno is the I2C master. The Arduino nano and the PWM driver are both I2C slaves. The master is responsible for the general flow of the program. The PWM driver is limited to replicating whatever PWM signal the master asks it to generate to a specific servomotor. The role of the Arduino nano is to serve as an interface between the LED matrix, the ultrasound sensor and the master. The master can ask the Arduino nano to display pre-defined patterns on the LED matrix depending on the state of the robot, and it can also ask the nano to respond with a single byte to know whether or not the ultrasound sensor has detected an object within a predefined range. Note that communication between I2C devices is not limited to single bytes, the bus can be read several times and be treated like a normal data stream, but for the purposes of this project I chose to keep it simple. In my github repository [link] you can find an example for multi packet data transmission between a master and a slave.

VI. Code

The code is hosted in my github repository in this [link].

The catapult code is separated in two Arduino sketches.

The SATMaster sketch should be uploaded to the Arduino uno, and the SATSlave should be uploaded to the Arduino nano. Keep in mind the Arduino Uno can be replaced for a smaller board (like a nano or a pro mini) if you have one at hand, since we only use the 5V, GND, A4 and A5 pins and the sketch size is very small. This should make the catapult more self-contained.

a. SATMaster

i. Catapult.cpp

This file defines a catapult class that is responsible for the movement of the servomotors. The class stores the PWM pulses needed to achieve the right angles for everything to work. The original idea was to have this class store the Adafruit_PWMServoDriver object so we could call parameter-less functions from the SATMaster script, but that didn't play too well with the double instantiation of the Wire object and the fact that the Adafruit_PWMServoDriver object's begin() method has a call to the Wire.begin() method, which did all sorts of crazy stuff on the servomotors for the first few loops of the execution. Methods:

```
Catapult(Adafruit_PWMServoDriver &servoDriver,
            int firePin,
            int armPin,
            int ballsPin,
            int platformPin);

void Catapult::sweep (Adafruit_PWMServoDriver &servoDriver,
                        int servoID,
                        int originAngle,
                        int destinationAngle,
                        int speed);

void Catapult::rest(Adafruit_PWMServoDriver &servoDriver );

void Catapult::prepareToShoot(Adafruit_PWMServoDriver &servoDriver );

void Catapult::shoot(Adafruit_PWMServoDriver &servoDriver );

void Catapult::openGate(Adafruit_PWMServoDriver &servoDriver);

void Catapult::closeGate(Adafruit_PWMServoDriver &servoDriver);

void Catapult::feedBall(Adafruit_PWMServoDriver &servoDriver );

void Catapult::stepScan(Adafruit_PWMServoDriver &servoDriver);
```

ii. SATMaster.ino

This is the main file for the I2C master board.

```
#define firePin 3
#define armPin 5
#define ballsPin 6
#define platformPin 0

Adafruit_PWMServoDriver servos = Adafruit_PWMServoDriver();
Catapult myCatapult(servos, firePin,armPin,ballsPin,platformPin);
```

The `Adafruit_PWMServoDriver` and `Catapult` objects are declared in the global scope so that we can initialize them in the setup() function and use them in the loop() function.

```
void setup() {

  servos.begin();
  servos.setPWMFreq(50);
  myCatapult.rest(servos);
  delay(500);

}
```

By calling the (`Adafruit_PWMServoDriver Object`).begin() function, we are initializing the Wire object, so we do not need to initialize it separately in the setup function.

```
void loop() {

  Wire.beginTransmission(slaveID);
  int available = Wire.requestFrom(slaveID, (uint8_t)1);
  currentCommand = SCAN_COMMAND;
  Wire.write(currentCommand);
```

`Wire.beginTransmission(slaveID)` starts an I2C transmission to the device located at the address `slaveID`. `Wire.requestFrom(slaveID, (uint8_t)1);` sends to the `slaveID` device a petition to run it's onRequest() function and to transmit a single byte.
`Wire.write(currentCommand)` sends a byte to the slave. Since current command is a uint8_t, it just sends the number as a byte. This is basically the flow of this sketch. The following statement is a combination of request and write function calls to get the slave device to either send us whether or not there is an object within range, or to get it to display our current status in the LED matrix.

b. SATSlave

i. Display

This is a class that has the function calls needed to use the LED display matrix.

```
#include "Arduino.h"
#include <Adafruit_GFX.h>
#include <Adafruit_NeoPixel.h>
#include <Adafruit_NeoMatrix.h>
```

The Adafruit libraries are needed to control the LED matrix because we are using **WS2812B** LEDs. There's also another library called FASTLed that is able to control these LEDs, but I found the adafruit libraries much easier to understand. Methods:

```
Display();
void displayRange(uint8_t range);
void Display::preparing();
void Display::shooting();
void Display::scanning();
void Display::detected();
```

ii. Sensor.cpp

The sensor class stores the data for the sensor pins and has a function that reads the sensor's output and returns a boolean depending if there is an object within our defined range of detection.

iii. SATSlave.ino

This sketch should be uploaded to the slave board. The slave board simply joins the I2C bus and has a few function calls to the Sensor and Display classes so it can fulfill whatever request our master might send out.

```
void setup() {
  Wire.begin(slaveID);
  Serial.begin(9600);
  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestSensorData);
}
```

The onReceive() function calls the relevant Display procedure depending on the Master's request, and the onRequest() function relays the value returned by the sensor class' detectTarget() function.

References

This project couldn't have been possible without thingiverse and instructables user avi_o.

He is the original designer for the catapult. I would have not been able to design this beast in the time frame given for this project without having the amazing base he did. You can find his instructables post with the original catapult in this [link]. I did modify several of the parts from the original catapult. I modified: the arming arm, the arm motor base and added the PWM driver mount. I also used the original idea for the code with simplified functions to call the general actions of the catapult.