YOU HAVE THE CONTROL

ESTACIÓN DE CARGA MOTORIZADA DE MANDOS DE JUEGO

DISEÑO DE SISTEMAS EMPOTRADOS

GRUPO 3

Proyecto por: Sahar Yousefi

Índice

Proyecto **3**

Materiales **4**

Diseño de hardware **5**

Código **7**

- **Inicialización** **7**
- **Ejecución** **9**

Diseño de piezas **14**

Problemas encontrados **16**

- **Servos sin rotación continua** **16**
- **Sistema de parada de servos** **16**
- **Resistencia de materiales** **17**
- **Errores de detección de objetos** **17**

Mejoras futuras **17**

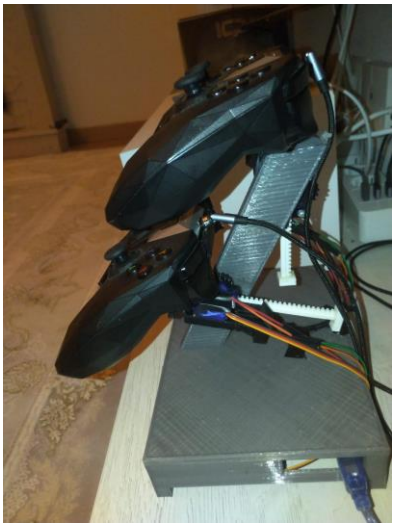
Anexo **18**

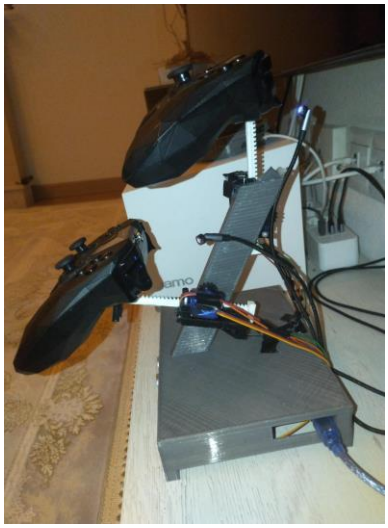
- **Código completo** **18**

Proyecto

You Have the Control es una estación de carga motorizada para gamepads.

Al acercarse el usuario, los mandos se mueven para poder ser recogidos con mayor facilidad. Al colocarlos de vuelta, las plataformas que sujetan los mandos se retraen y los cables de carga magnéticos se conectan automáticamente a los mandos.





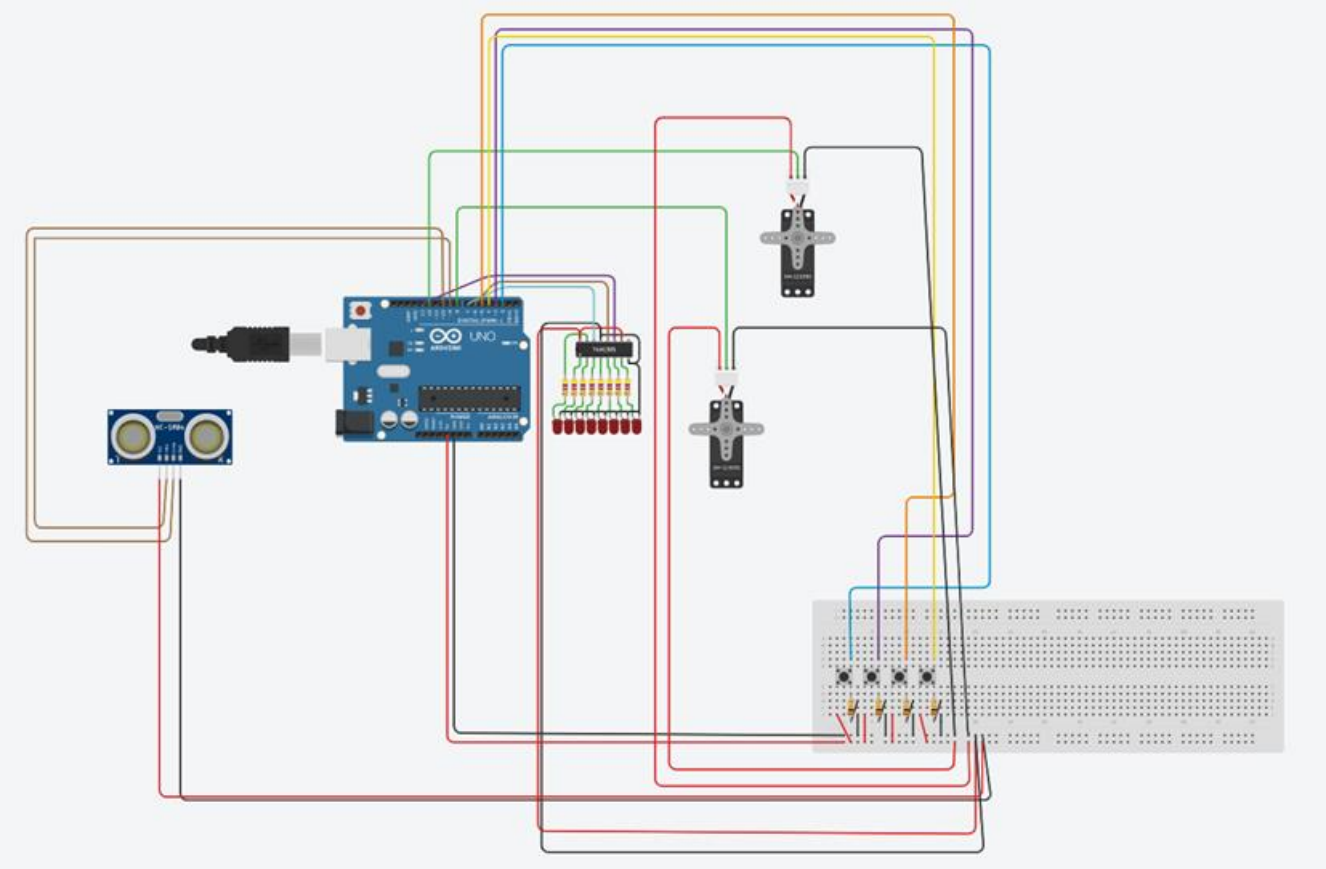
Materiales

TOTAL	**35,67€**
Arduino Uno R3	11,50€
2 servos SG90	5€
4 pulsadores switch	2€
Registro de desplazamiento 74HC595	0,32€
Mini placa de prototipo	1,30€
8 LEDs	0,40€
Sensor ultrasónico SR04	2,70€
Filamento PETG y PLA	10€
12 resistencias	0,45€
Cables	2€

Diseño de hardware

Todo el diseño es original, diseñado desde cero para el proyecto.

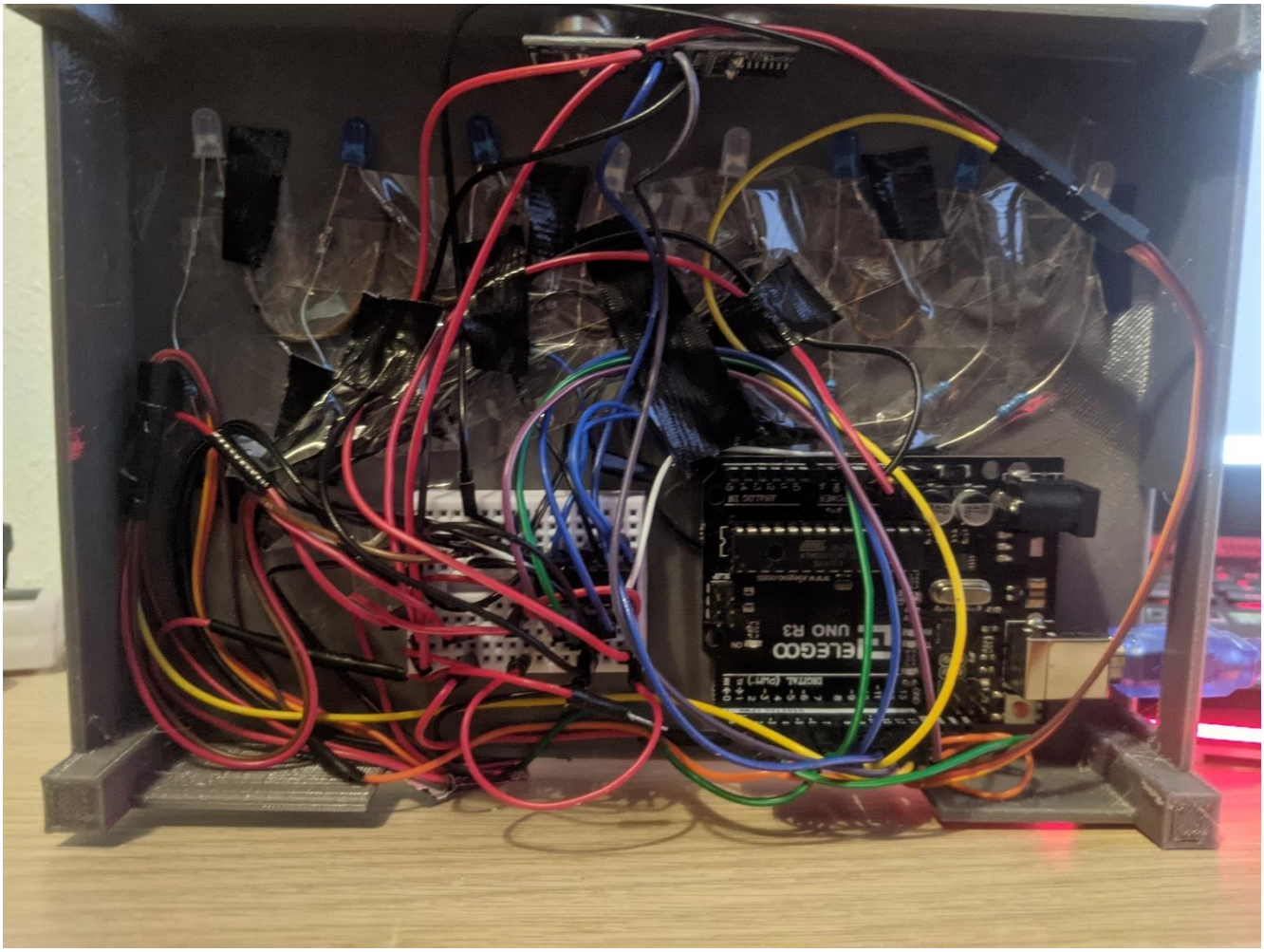
El prototipo inicial y simulación del circuito se realizó en TinkerCad antes de ser implementado con componentes reales.



Los servos están conectados a la placa Arduino Uno directamente, al usar sólo dos servos no hace falta una placa driver dedicada. Se utilizan dos pulsadores como detectores de fin de carrera para cada servo, con resistencias para obtener señales HIGH y LOW que pudiera leer el Arduino y poder ser utilizadas en el código de control.

El medidor de ultrasonidos se encarga de detectar objetos a menos de 25cm delante de la plataforma para activar los servos.

Para la iluminación, hizo falta utilizar un registro de desplazamiento conectado a través de una mini placa de prototipo. Al tener que manejar 8 LEDs, los pines disponibles en el Arduino no eran suficientes. Los LEDs están cableados en paralelo para poder activarlos de forma individual a través del registro de desplazamiento. El registro de desplazamiento y todo el cableado de los leds se externalizó en una mini placa de prototipo conectada al Arduino.



Código

Inicialización

Se definen las constantes y variables para los servos, sus pines y sus ángulos de trabajo:

```
#include <Servo.h>
#include <NewPing.h>

// servos
#define BOTTOM_SERVO_PIN 8;
#define BOTTOM_SERVO_STOP_ANGLE 87;
#define BOTTOM_SERVO_FORWARD_ANGLE 92;
#define BOTTOM_SERVO_BACKWARD_ANGLE 84;
Servo bottomServo;

#define TOP_SERVO_PIN 12;
#define TOP_SERVO_STOP_ANGLE 89;
#define TOP_SERVO_FORWARD_ANGLE 85;
#define TOP_SERVO_BACKWARD_ANGLE 93;
Servo topServo;
```

Se definen las conexiones del sensor de ultrasonidos, y se inicializa. También se definen las variables para las distancias medidas y si se ha detectado un objeto o no:

```
// ultrasonic
#define ULTRASONIC_TRIG_PING = 9;
#define ULTRASONIC_ECHO_PING = 10;
unsigned int distance = 0;
boolean objectDetected = false;
NewPing sonar(ULTRASONIC_TRIG_PING, ULTRASONIC_ECHO_PING, 150);
```

Se definen los pines de los pulsadores de fin de carrera, y las variables para guardar su estado:

```
// buttons
#define BOTTOM_SERVO_RETRACTED_BUTTON_PIN 2;
#define BOTTOM_SERVO_EXTENDED_BUTTON_PIN 3;
boolean bottomServoRetracted = false;
boolean bottomServoExtended = false;
#define TOP_SERVO_RETRACTED_BUTTON_PIN 4;
#define TOP_SERVO_EXTENDED_BUTTON_PIN 5;
boolean topServoRetracted = false;
boolean topServoExtended = false;
```

Se definen los pines del registro de desplazamiento, y las variables de control de luces:

```
// lights
#define SHIFT_REGISTER_DATA_PIN 7;
#define SHIFT_REGISTER_LATCH_PIN 6;
#define SHIFT_REGISTER_CLOCK_PIN 13;
#define LIGHT_DELAY 100
unsigned int currentLight = 0;
boolean enableLights = false;
boolean reverseLightDirection = false;
```

Se conectan todos los componentes:

```
void setup()
{
  Serial.begin(115200);
  bottomServo.attach(BOTTOM_SERVO_PIN);
  topServo.attach(TOP_SERVO_PIN);
  bottomServo.write(BOTTOM_SERVO_STOP_ANGLE);
  topServo.write(TOP_SERVO_STOP_ANGLE);
```

```
  pinMode(BOTTOM_SERVO_RETRACTED_BUTTON_PIN, INPUT);
  pinMode(BOTTOM_SERVO_EXTENDED_BUTTON_PIN, INPUT);
  pinMode(TOP_SERVO_RETRACTED_BUTTON_PIN, INPUT);
  pinMode(TOP_SERVO_EXTENDED_BUTTON_PIN, INPUT);

  pinMode(ULTRASONIC_TRIG_PING, OUTPUT);
  pinMode(ULTRASONIC_ECHO_PING, INPUT);

  pinMode(SHIFT_REGISTER_DATA_PIN, OUTPUT);
  pinMode(SHIFT_REGISTER_LATCH_PIN, OUTPUT);
  pinMode(SHIFT_REGISTER_CLOCK_PIN, OUTPUT);
}
```

Ejecución

En el bucle de ejecución, primeramente se leen los estados de los botones:

```
void loop()
{
  delay(50);
  distance = sonar.ping_cm();
  bottomServoRetracted =
digitalRead(BOTTOM_SERVO_RETRACTED_BUTTON_PIN) == HIGH;
  bottomServoExtended =
digitalRead(BOTTOM_SERVO_EXTENDED_BUTTON_PIN) == HIGH;
  topServoRetracted = digitalRead(TOP_SERVO_RETRACTED_BUTTON_PIN)
== HIGH;
  topServoExtended = digitalRead(TOP_SERVO_EXTENDED_BUTTON_PIN) ==
HIGH;
```

A continuación, se comprueba si hay algún objeto delante de la plataforma. No queremos que los servos cambien de dirección antes de extenderse o retraerse por completo, por eso sólo medimos si no se mueven.

```
// only check for objects when servo is not moving
if (bottomServoRetracted || bottomServoExtended)
{
  if (25 > distance && distance != 0)
  {
    objectDetected = true;
  }
  else if (distance != 0)
  {
    objectDetected = false;
  }
}
```

El código de control del servo inferior es el más complicado, ya que en él se basa el comportamiento de las luces y la espera para dejar que el usuario recoja o deje los mandos.

Si la plataforma no está extendida, y hay un objeto detectado, se envía la señal al servo de que gire hacia delante para extender la plataforma. Además, las luces se iluminan en secuencia de izquierda a derecha.

```
// bottom servo control
if (!bottomServoExtended && objectDetected == true)
{
  bottomServo.write(BOTTOM_SERVO_FORWARD_ANGLE);
  reverseLightDirection = false;
  enableLights = true;
}
```

Si la plataforma está extendida, y hay un objeto detectado, se espera 4 segundos para que el usuario pueda recoger o dejar el mando con comodidad. Además, se encienden todas las luces.

```
else if (bottomServoExtended && objectDetected == true)
{
  // wait 4 seconds when extended to let user pick up gamepads
  // also turn on all the lights
  ledWrite(B11111111);
  currentLight = 0;
  delay(4000);
}
```

Si la plataforma no está retraída, y no hay ningún objeto, se envía la orden al servo de girar hacia atrás. Además se cambia la dirección de la secuencia de luces, para que se enciendan de derecha a izquierda.

```
else if (!bottomServoRetracted && objectDetected == false)
{
  bottomServo.write(BOTTOM_SERVO_FORWARD_ANGLE);
  // lights go in reverse direction when retracting
  reverseLightDirection = true;
  enableLights = true;
}
```

Si la Plataforma está completamente extendida o retraída, se manda el ángulo neutro al servo para detener el movimiento y que no se queme. Además, se apagan las luces.

```
else
{
  bottomServo.write(BOTTOM_SERVO_STOP_ANGLE);
  // restart the lights and turn off
  currentLight = 0;
  enableLights = false;
}
```

El control del servo superior es más simple. Es igual que el del servo inferior pero sin el resto de instrucciones relativas a las luces.

```
// top servo control
if (!topServoExtended && objectDetected == true)
{
  topServo.write(TOP_SERVO_FORWARD_ANGLE);
}
else if (!topServoRetracted && objectDetected == false)
{
  topServo.write(TOP_SERVO_BACKWARD_ANGLE);
}
else
{
  topServo.write(TOP_SERVO_STOP_ANGLE);
}
```

Para el control de las luces, se envía el código binario al registro de desplazamiento para encender el primer o último led, y se va moviendo mediante el operador *right shift* o *left shift*, dependiendo de la dirección en la que queremos que se enciendan las luces en secuencia.

```
// light control
if (enableLights == true)
{
  if (reverseLightDirection)
  {
    ledWrite(B11000000 >> currentLight);
  }
  else
  {
    ledWrite(B00000011 << currentLight);
  }

  delay(LIGHT_DELAY);
```

```
    currentLight++;

    if (currentLight > 8)
      currentLight = 0;
  }
  else
  {
    ledWrite(B00000000);
  }
}
```

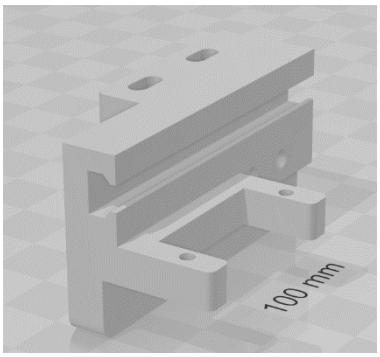
Se utiliza esta función de utilidad para encender LEDs a través del registro de desplazamiento (fuera de la función *loop()*)

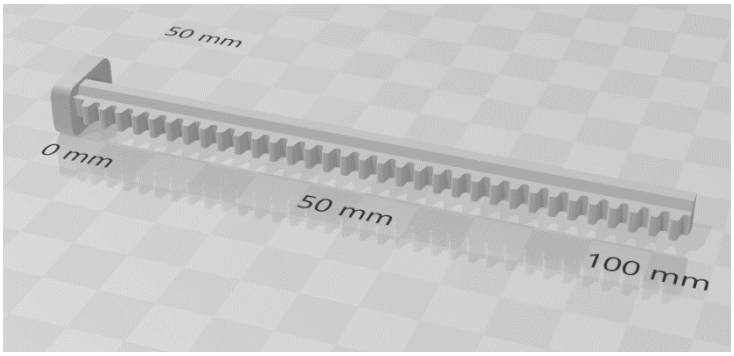
```
void ledWrite(int Led)
{
  shiftOut(SHIFT_REGISTER_DATA_PIN, SHIFT_REGISTER_CLOCK_PIN,
LSBFIRST, Led);
  digitalWrite(SHIFT_REGISTER_LATCH_PIN, HIGH);
  digitalWrite(SHIFT_REGISTER_LATCH_PIN, LOW);
}
```

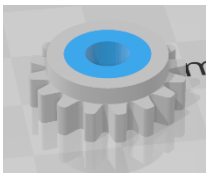
Diseño de piezas

Todas las piezas han sido impresas con impresora 3D.

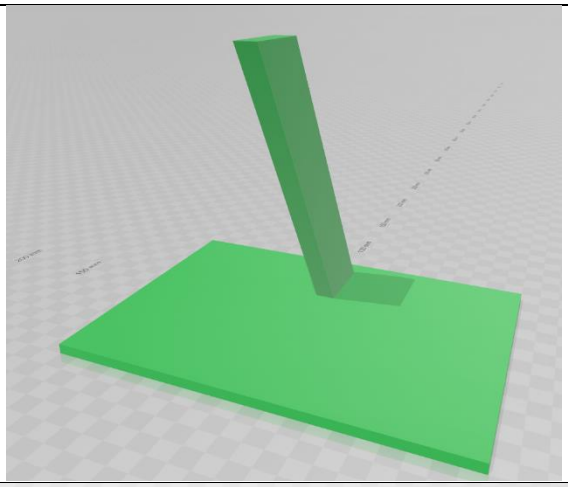
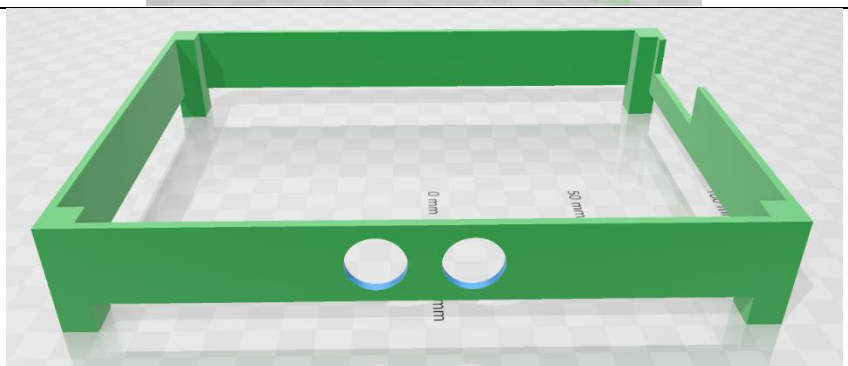
Se ha utilizado el siguiente modelo de actuador lineal basado en servo de Thingiverse: https://www.thingiverse.com/thing:3170748

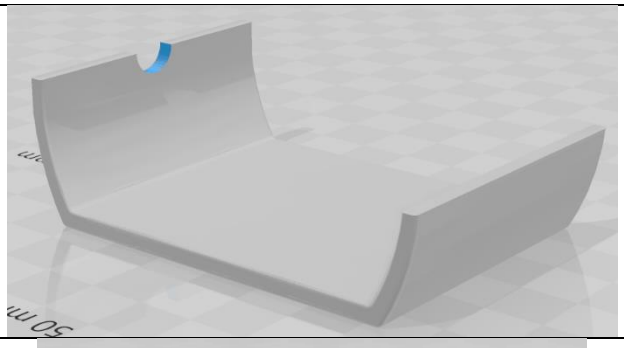
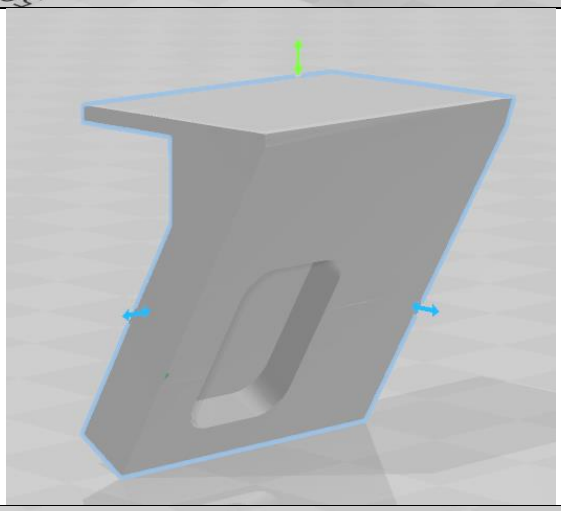
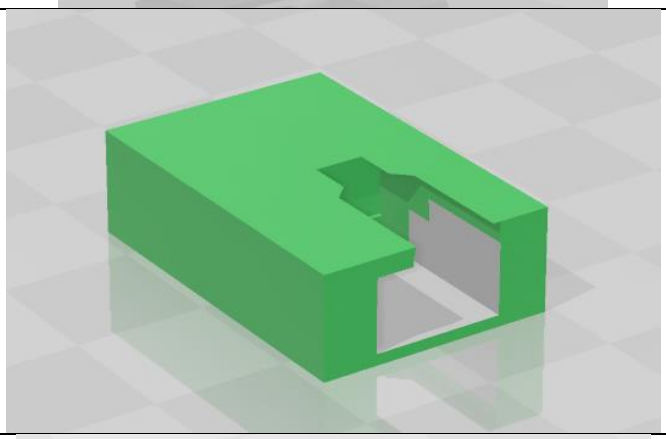
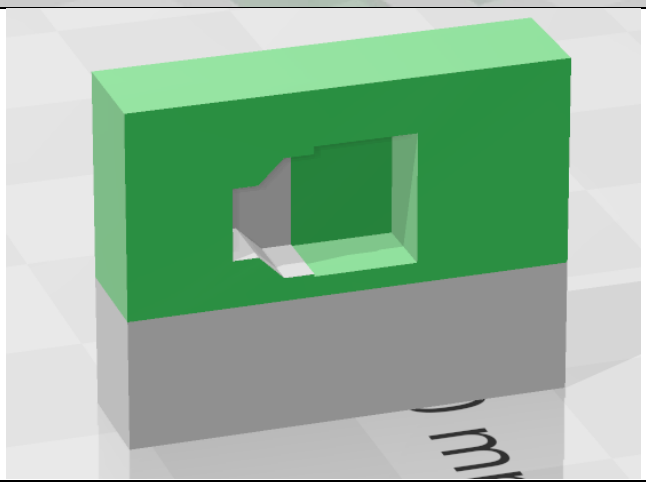






El resto de piezas han sido diseñadas desde cero en Microsoft 3D Builder.

Base	
Patas	

Plataforma para mando	
Endstop delantero de cremallera diagonal	
Endstop delantero de cremallera vertical	
Endstop trasero	

Problemas encontrados

Servos sin rotación continua

El primer problema fue que los servos sólo giran 180 grados, y necesitamos que giren de forma continua para poder usarlos como actuadores lineales.

Los servos de rotación continua son más caros y difíciles de conseguir, así que se decidió modificar los servos SG90 para convertirlos en rotación continua.

Para ello, hay que abrir el servo y cortar la punta del eje que entra en el potenciómetro. Así el servo no recibe información sobre su posición, y girará continuamente dependiendo del ángulo que se envíe.

Una vez hecha la modificación, hay que encontrar el ángulo neutro del servo. Suele estar en los 90 grados, pero cada servo tiene una calibración un poco diferente. Para ello, se hizo un pequeño programa que fuese mandando ángulos de 0 a 180 y así poder encontrar el ángulo en el que el servo deja de moverse.

Para controlar la velocidad de rotación, cuanto más cercano sea el ángulo al neutro, más despacio se moverá el servo. Por ejemplo, si el ángulo neutro es 90, enviando 91 el servo se moverá muy despacio, 92 más rápido, y normalmente mandando 95 en adelante se mueve a la máxima velocidad.

Para que el servo gire en dirección contraria, se envían ángulos por debajo del ángulo neutro.

Sistema de parada de servos

Al convertir los servos a rotación continua, no tenemos información sobre la posición del servo, así que hizo falta pensar una manera de saber cuándo el servo debía parar al terminar el recorrido de cada plataforma.

Se decidió utilizar pulsadores como detectores de fin de carrera, con resistencias para conseguir señalas altas y bajas como inputs.

Resistencia de materiales

La mayoría de las piezas han sido impresas en plástico PETG, pero una vez montado, se descubrió que las cremalleras eran demasiado flexibles, por lo que se doblaban al estar extendidas con el peso de los mandos. Se decidió imprimir de nuevo las cremalleras en plástico PLA, que es más rígido, solucionando así el problema.

Errores de detección de objetos

Durante las pruebas del medidor de ultrasonidos, se obtenían mediciones muy erráticas de la distancia, que dificultaba poder detectar un objeto frente a la plataforma de forma consistente.

Investigando posibles soluciones, se encontró la librería NewPing https://bitbucket.org/teckel12/arduino-new-ping/wiki/Home que soluciona la mayoría de problemas con los sensores de ultrasonidos baratos.

Mejoras futuras

La idea original era que los mandos estuviesen custodiados por un dragón, que abriese las alas a la vez que los mandos se movieran. Pero no hubo tiempo para diseñar el modelo 3D del dragón, imprimirlo y montarlo.

Anexo

Código completo

```
#include <Servo.h>
#include <NewPing.h>

// servos
#define BOTTOM_SERVO_PIN 8;
#define BOTTOM_SERVO_STOP_ANGLE 87;
#define BOTTOM_SERVO_FORWARD_ANGLE 92;
#define BOTTOM_SERVO_BACKWARD_ANGLE 84;
Servo bottomServo;

#define TOP_SERVO_PIN 12;
#define TOP_SERVO_STOP_ANGLE 89;
#define TOP_SERVO_FORWARD_ANGLE 85;
#define TOP_SERVO_BACKWARD_ANGLE 93;
Servo topServo;

// ultrasonic
#define ULTRASONIC_TRIG_PING = 9;
#define ULTRASONIC_ECHO_PING = 10;
unsigned int distance = 0;
boolean objectDetected = false;
NewPing sonar(ULTRASONIC_TRIG_PING, ULTRASONIC_ECHO_PING, 150);

// buttons
#define BOTTOM_SERVO_RETRACTED_BUTTON_PIN 2;
#define BOTTOM_SERVO_EXTENDED_BUTTON_PIN 3;
boolean bottomServoRetracted = false;
boolean bottomServoExtended = false;
#define TOP_SERVO_RETRACTED_BUTTON_PIN 4;
#define TOP_SERVO_EXTENDED_BUTTON_PIN 5;
boolean topServoRetracted = false;
```

```
boolean topServoExtended = false;

// lights
#define SHIFT_REGISTER_DATA_PIN 7;
#define SHIFT_REGISTER_LATCH_PIN 6;
#define SHIFT_REGISTER_CLOCK_PIN 13;
#define LIGHT_DELAY 100
unsigned int currentLight = 0;
boolean enableLights = false;
boolean reverseLightDirection = false;

void ledWrite(int Led)
{
  shiftOut(SHIFT_REGISTER_DATA_PIN, SHIFT_REGISTER_CLOCK_PIN,
LSBFIRST, Led);
  digitalWrite(SHIFT_REGISTER_LATCH_PIN, HIGH);
  digitalWrite(SHIFT_REGISTER_LATCH_PIN, LOW);
}

void setup()
{
  Serial.begin(115200);
  bottomServo.attach(BOTTOM_SERVO_PIN);
  topServo.attach(TOP_SERVO_PIN);
  bottomServo.write(BOTTOM_SERVO_STOP_ANGLE);
  topServo.write(TOP_SERVO_STOP_ANGLE);

  pinMode(BOTTOM_SERVO_RETRACTED_BUTTON_PIN, INPUT);
  pinMode(BOTTOM_SERVO_EXTENDED_BUTTON_PIN, INPUT);
  pinMode(TOP_SERVO_RETRACTED_BUTTON_PIN, INPUT);
  pinMode(TOP_SERVO_EXTENDED_BUTTON_PIN, INPUT);

  pinMode(ULTRASONIC_TRIG_PING, OUTPUT);
  pinMode(ULTRASONIC_ECHO_PING, INPUT);
```

```
  pinMode(SHIFT_REGISTER_DATA_PIN, OUTPUT);
  pinMode(SHIFT_REGISTER_LATCH_PIN, OUTPUT);
  pinMode(SHIFT_REGISTER_CLOCK_PIN, OUTPUT);
}

void loop()
{
  delay(50);
  distance = sonar.ping_cm();
  bottomServoRetracted =
digitalRead(BOTTOM_SERVO_RETRACTED_BUTTON_PIN) == HIGH;
  bottomServoExtended =
digitalRead(BOTTOM_SERVO_EXTENDED_BUTTON_PIN) == HIGH;
  topServoRetracted = digitalRead(TOP_SERVO_RETRACTED_BUTTON_PIN)
== HIGH;
  topServoExtended = digitalRead(TOP_SERVO_EXTENDED_BUTTON_PIN) ==
HIGH;

  // only check for objects when servo is not moving
  if (bottomServoRetracted || bottomServoExtended)
  {
    if (25 > distance && distance != 0)
    {
      objectDetected = true;
    }
    else if (distance != 0)
    {
      objectDetected = false;
    }
  }

  // bottom servo control
  if (!bottomServoExtended && objectDetected == true)
  {
    bottomServo.write(BOTTOM_SERVO_FORWARD_ANGLE);
```

```
    reverseLightDirection = false;
    enableLights = true;
  }
  else if (bottomServoExtended && objectDetected == true)
  {
    // wait 4 seconds when extended to let user pick up gamepads
    // also turn on all the lights
    ledWrite(B11111111);
    currentLight = 0;
    delay(4000);
  }
  else if (!bottomServoRetracted && objectDetected == false)
  {
    bottomServo.write(BOTTOM_SERVO_FORWARD_ANGLE);
    // lights go in reverse direction when retracting
    reverseLightDirection = true;
    enableLights = true;
  }
  else
  {
    bottomServo.write(BOTTOM_SERVO_STOP_ANGLE);
    // restart the lights and turn off
    currentLight = 0;
    enableLights = false;
  }

  // top servo control
  if (!topServoExtended && objectDetected == true)
  {
    topServo.write(TOP_SERVO_FORWARD_ANGLE);
  }
  else if (!topServoRetracted && objectDetected == false)
  {
    topServo.write(TOP_SERVO_BACKWARD_ANGLE);
  }
```

```
  else
  {
    topServo.write(TOP_SERVO_STOP_ANGLE);
  }

  // light control
  if (enableLights == true)
  {
    if (reverseLightDirection)
    {
      ledWrite(B11000000 >> currentLight);
    }
    else
    {
      ledWrite(B00000011 << currentLight);
    }

    delay(LIGHT_DELAY);

    currentLight++;

    if (currentLight > 8)
      currentLight = 0;
  }
  else
  {
    ledWrite(B00000000);
  }
}
```