BOLA EN SUPERFICIE CON CONTROLADOR PID

(**Balancing ball with PID Controller**)

Grupo Bi&Pi. Arquitecturas Específicas y Empotradas.
Universidad de León. Curso 2019-2020

Adrián Andrés Gómez
Andrés Tascón González

Índice

Anexos

Índice de las imágenes

Índice de los planos

Definiciones

- DIY: Do it Yourself, o en español HTM Hazlo tú mismo, es un término referido normalmente a proyectos que se pueden desarrollar entre pocas personas y no requieren un coste demasiado elevado.
- PID: Controlador Proporcional Integral Derivativo, explicado con detalle en el apartado Explicación del proyecto.
- Setpoint: Es el valor normalmente al que se quiere llegar a través del PID y con el que se calcula el error producido, en nuestro caso, el punto medio de la pantalla resistiva a no ser que se cambie.
- Output: Es el valor producido por el controlador PID después de calcular el error y computarlo en un resultado, en nuestro caso, es el ángulo (en microsegundos) que deberán tomar los servos para poner la pantalla en la posición deseada.
- Write: Utilizaremos este término para referirnos a escribir un ángulo en los motores servo, por así decirlo, es el punto medio del servo, el que forma 90 grados sumado o restado al *output*.

Resumen de la idea del proyecto

La idea del proyecto surge tras estar investigando al comienzo del curso proyectos DIY. Viendo un vídeo de recopilación de proyectos con Arduino de este calibre, surge un prototipo de proyecto que nos da la idea para elaborar nuestro propio proyecto.
Fue una elección sencilla puesto que llamaba mucho la atención a simple vista y parecía un proyecto divertido y gustoso de ver una vez finalizado.

La idea del proyecto es construir una estructura que, en la parte inferior albergue una base para sujetar 3 motores servo y que estos muevan con un cabezal una especie de rótula. Todo ello con el fin de poder mover una base en la parte superior en el eje X e Y a la que esté pegada una pantalla resistiva sobre la que ruede una bola de metal.

Gracias a la pantalla resistiva se puede conocer la posición de la bola y mediante la implementación de un código en Arduino con un controlador PID (controlador proporcional, integral y derivativo) se puede calcular el error y por tanto, desplazar la base para mover la bola al punto deseado. De esta manera se puede mover la bola en cuadrados, círculos, se puede mantener en un punto fijo… etc.

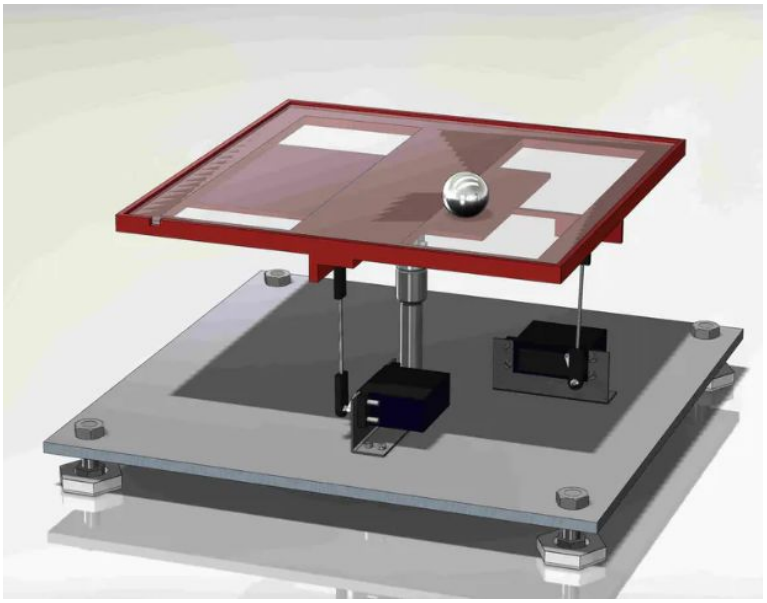


Figura 3.0.

Objetivos iniciales del proyecto

Los objetivos iniciales del proyecto son:

- Construcción de la estructura.
- Desarrollo del código.
- Juntar todo para que funcione correctamente.
- Creación de página web/app móvil para controlar la bola.

Organización del grupo y las tareas

Al ser este proyecto algo relativamente nuevo para nosotros, puesto que ninguno habíamos trabajado previamente con Arduino ni teníamos realmente mucho conocimiento técnico en electrónica, decidimos desarrollar enteramente el proyecto los dos juntos ayudándonos el uno al otro.

El horario de trabajo del grupo ha sido en gran medida el trabajo realizado en clase los lunes de 9:00 a 13:00h, con un descanso de media hora donde se hablaba entre los dos los siguientes pasos a seguir en las siguientes horas de trabajo de mismo día. Además, a partir de diciembre se ha estado quedando casi todos los viernes por la mañana para anticipar el trabajo que se iba a llevar el lunes en el seguimiento con el profesor y preguntar aquellas dudas que nos surgían, con el fin de conseguir un proyecto notorio.

Hemos creado un proyecto en Trello, ya que estamos familiarizados con las metodologías ágiles del desarrollo de software por haberlas utilizado en otros proyectos anteriores. En este tablero se pueden diferenciar el estado de las tareas en las 4 columnas que se pueden observar:

- HERRAMIENTAS: Aquí se ponían los diferentes materiales que todavía no teníamos para poder acabar el proyecto. Exclusivamente es de hardware.
- SIN HACER: Tareas que todavía no se han comenzado.
- EN PROCESO: Tareas que se están realizando en este momento.
- HECHO: Tareas ya finalizadas dentro del proyecto y que su uso correcto está validado.

Figura 5.0.

Al principio del trabajo no nos repartimos ninguna tarea, sino que como mucho, uno de los dos se encargaba de algo en concreto y actuaba como responsable de la tarea, pero el trabajo era mutuo y en conjunto, ya que los primeros pasos los queríamos dar pequeños para que la base del proyecto fuera consistente por dos motivos: no teníamos muy claro el funcionamiento de los diferentes componentes y no queríamos tener que volver empezar desde el principio de nuevo.

Es en las dos últimas semanas cuando cada uno ha trabajado más individualmente en diferentes partes del proyecto para llegar a la fecha de entrega correctamente. En concreto, Adrián se ha encargado más de la parte de hardware y Andrés se ha encargado de la parte de software.

Cronograma del proyecto

En la siguiente imagen se ven detalladas las tareas por las que nos hemos basado para tener una buena organización durante el proyecto, además del tablero de Trello.

Modo de	Nombre de tarea	Duración	Comienzo	Fin
	Aprender el manejo básico de los componentes de Arduino	11 días	lun 16/09/19	lun 30/09/19
	Elección del proyecto	8 días	mar 01/10/19	jue 10/10/19
	Análisis de requisitos	12 días	vie 11/10/19	lun 28/10/19
	Obtención de los servos	3 días	mar 29/10/19	jue 31/10/19
	Esperar a que llegue la pantalla	14 días	mar 29/10/19	vie 15/11/19
	Control de los servos	9 días	vie 01/11/19	mié 13/11/19
	Impresión del triángulo	4 días	lun 18/11/19	jue 21/11/19
	Esperar a que lleguen las rótulas	8 días	vie 22/11/19	mar 03/12/19
	Impresión de los casquillos	1 día	mié 04/12/19	mié 04/12/19
	Preparación de las piernas	5 días	jue 05/12/19	mié 11/12/19
	Unir la estructura	6 días	jue 12/12/19	jue 19/12/19
	Pegar la estructura	1 día	vie 20/12/19	vie 20/12/19
	Entender el PID	3 días	lun 23/12/19	mié 25/12/19
	Sintonizar el PID	2 días	jue 26/12/19	vie 27/12/19
	Ajustar el código al proyecto	4 días	lun 30/12/19	jue 02/01/20
	Implementar las ecuaciones de error	5 días	vie 03/01/20	jue 09/01/20
	Mantenimiento del software	20 días	lun 23/12/19	vie 17/01/20
	Mantenimiento del hardware	56 días	vie 01/11/19	vie 17/01/20

Figura 7.0.

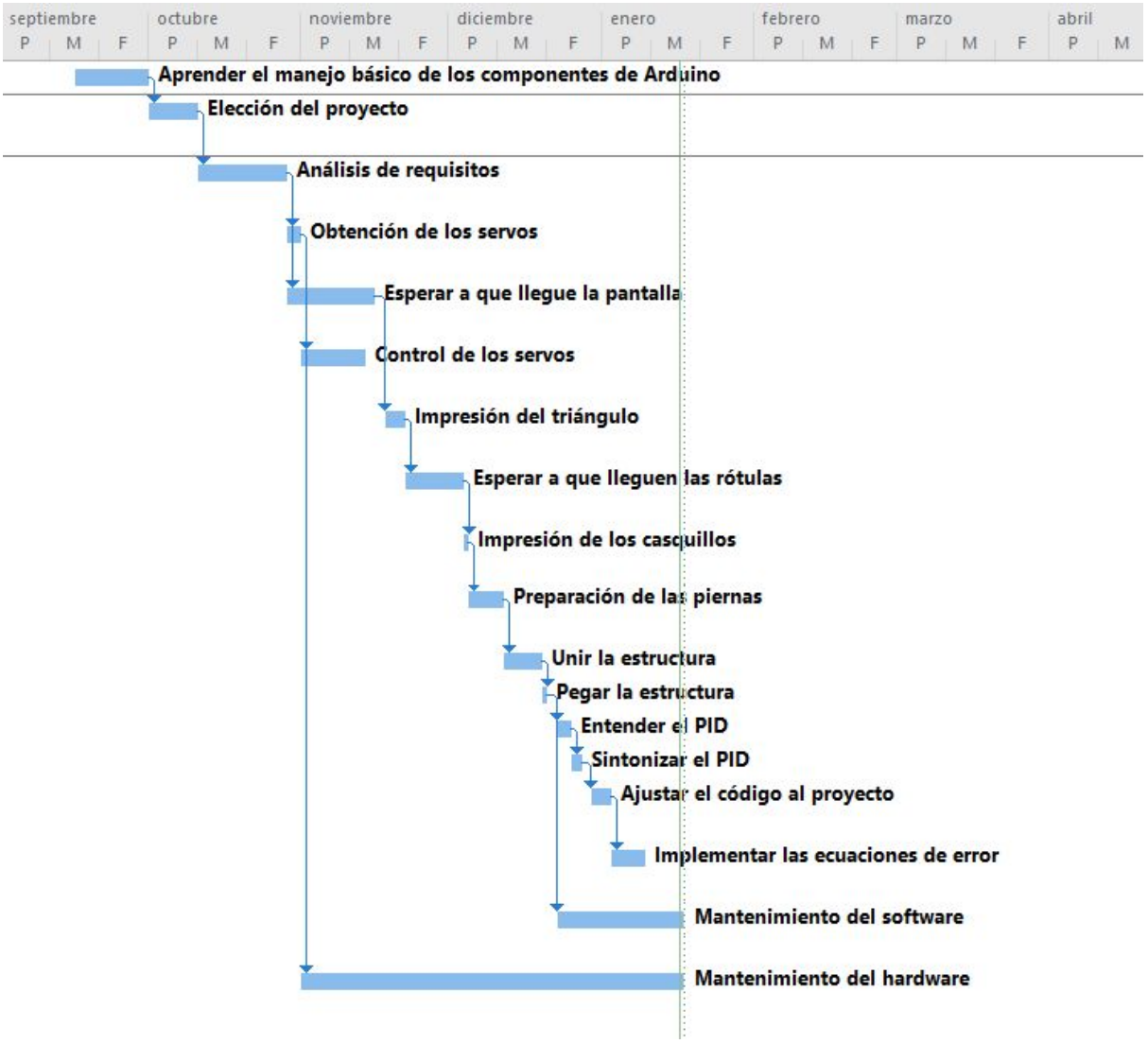

septiembre
octubre
noviembre
diciembre
enero
febrero
marzo
abril
P M F P M F P M F P M F P M F P M F P M F P M
Aprender el manejo básico de los componentes de Arduino
Elección del proyecto
Análisis de requisitos
Obtención de los servos
Esperar a que llegue la pantalla
Control de los servos
Impresión del triángulo
Esperar a que lleguen las rótulas
Impresión de los casquillos
Preparación de las piernas
Unir la estructura
Pegar la estructura
Entender el PID
Sintonizar el PID
Ajustar el código al proyecto
Implementar las ecuaciones de error
Mantenimiento del software
Mantenimiento del hardware

Figura 7.1.

Descripción de los materiales utilizados

El proyecto tiene los siguientes componentes:

- 1.x Placa de Arduino UNO.
- 3.x Motores servo DS3218MG.
- 2.x Resistencias de 10 MΩ.
- 1.x Pantalla resistiva 22.5x17.3 cm.
- 6.x Rótulas hembra de M4.
- 3.x Varillas de metal de 2mm. Ø de 15 cm.
- 1.x Bola de metal de 2mm. Ø.
- 1.x Base de conglomerado de 1.5mm de ancho y 40x30cm.
- 12.x Cables conectores.
- 1.x Fuente de alimentación de 5V.
- 12.x Tornillos M4.
- 12.x Tuercas M4.

Para realizar este proyecto hemos requerido de lo siguiente:

- Herramientas básicas de carpintería (destornilladores, sierras, alicates, etc.).
- Pistola de pegamento termofusible.
- Impresora 3D.
- Soldador.
- Pegamento instantáneo.

Descripción de las herramientas de software utilizadas

Para realizar este proyecto hemos utilizado las siguientes herramientas de software:

- Arduino IDE: De cara al desarrollo del software.
- Tinkercad: Software online utilizado para el manejo y edición de piezas para su posterior impresión 3D.
- Repetier Host: Programa utilizado para la impresión de piezas 3D.
- Fritzing: Software para la creación de planos electrónicos.

Explicación del proyecto

Nuestro proyecto se basa en sostener y controlar de la manera que queramos nosotros tanto la velocidad como la posición de una bola de metal sobre una pantalla resistiva, para ello, utilizaremos un controlador PID. Con una pantalla resistiva podemos saber en qué momento la bola está encima de la pantalla [2], pues hace cuando la bola está sobre la superficie de la pantalla, entran en contacto dos capas, así es posible obtener en el programa metido en el Arduino las dos coordenadas: X e Y.

Un controlador PID está compuesto de tres parámetros que proporcionan una acción proporcional, integral y derivativa (PID) [2].

Utiliza varias constantes:

- **K_p**: Constante de proporcionalidad. Multiplica la señal de error. Si es aumentada:
  - Aumenta la velocidad de respuesta.
  - Disminuye el error del sistema.
  - Aumenta la inestabilidad.

- **K_d**: Constante de derivación. Velocidad a la que el sistema se acerca a la referencia. Si es aumentada:
  - Aumenta la estabilidad.
  - Disminuye la velocidad.

- **K_i**: Constante de integración. Velocidad con la que se repite la acción proporcional. Si es aumentada:
  - Disminuye el error.
  - Aumenta la velocidad.
  - Aumenta la inestabilidad.

Como hemos podido observar, vamos a tener en cuenta tres constantes. Para que el controlador funcione lo mejor posible, hemos de encontrar una combinación de valores de las tres óptima. En nuestro caso, tras probar varias, utilizamos un mecanismo de sintonización basado en aumentar poco a poco la constante de proporcionalidad para disminuir el error. Si el sistema se vuelve inestable antes de alcanzar la respuesta deseada, se aumenta la constante de derivación. Si nuevamente sigue siendo inestable, se aumenta la constante de

integración y si además, después de aumentar la constante de integración sigue siendo inestable, se volverá a aumentar la constante de derivación.
Por consiguiente, nosotros hemos utilizado la siguiente combinación de valores en los parámetros antes descritos:

- **K_p:** 0.3
- **K_d:** 0.14
- **K_i:** 0.25

La ecuación que utiliza el controlador es la siguiente:

$$c(t) = Kp \cdot e(t) + Ki \cdot \int e(t)dt + Kd \cdot \frac{\partial e(t)}{\partial t}$$

donde:

$c(t)$: Valor del controlador en el tiempo t.

Kp : Constante de proporcionalidad.

$e(t)$: Error en el tiempo de t.

Ki : Constante de integración.

Kd : Constante de derivación.

$\partial e(t)$: Diferencial del error de t.

∂t : Diferencial de t.

Desarrollo y montaje

El desarrollo del proyecto ha sido largo y hemos tenido numerosos problemas. Hemos encontrado especialmente una gran cantidad de problemas en el montaje de la estructura, puesto que la documentación que seguimos en su gran mayoría estaba poco detallada.

Desarrollo y montaje de la estructura

Lo primero que hicimos fue pedir la pantalla resistiva para tenerla lo antes posible. En una búsqueda exhaustiva en todos los portales de venta de electrónica y grandes almacenes, decidimos comprar una pantalla de 10,4 pulgadas en Aliexpress que llegaba a las dos semanas con buena relación dimensión-tiempo de llegada-precio. Es una parte fundamental para el proyecto (primeramente entender cómo funciona para después poder montarla en la estructura).

Entre tanto, nos dedicamos a imprimir algunas partes de la estructura ya que conseguimos varios archivos .STL buscando documentación online [1] para realizar nuestro propio montaje. Se imprimieron la pieza base donde se iba a sostener todo el proyecto y la plataforma donde se pega la pantalla. Además, comenzamos a tantear qué motores servos del laboratorio nos servirían, ya que teníamos que tener en cuenta que tuvieran la suficiente fuerza como para aguantar el peso de parte de la pantalla y de la bola, y una rápida respuesta de giro de las órdenes del Arduino.

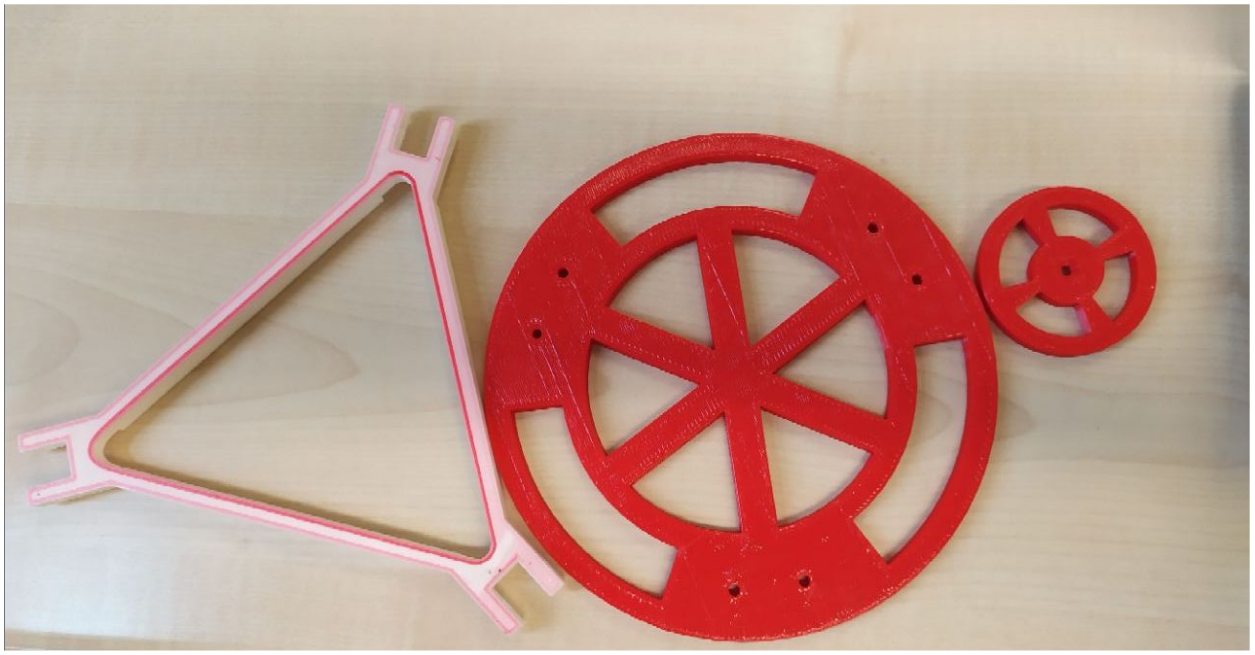


Figura 10.a.0.

Una vez recibida la pantalla nos dimos cuenta de que no nos habíamos percatado de que las dimensiones de las piezas impresas y la pantalla recibida no estaban relacionadas, por lo que lo primero que realizamos fue la redimensión de la pieza que sujetaba la pantalla. También pensamos en redimensionar la pieza base pero al ser prescindible, se decidió sustituirla por un tablero de conglomerado de 40x30cm suficiente para el montaje de los tres motores en forma triangular y la placa de Arduino. Vimos un buen sustituto el tablero de conglomerado debido a su bajo coste, y la facilidad que da taladrar los servos, en el caso de que decidiéramos hacerlo así, para que tengan un giro perfecto..

De cara a imprimir la base superior, que es la que sujeta la pantalla resistiva, como se ha mencionado antes, se necesitaba redimensionar la pieza, ya que al tener una pantalla con las dimensiones más grande de lo planteado, el triángulo no ocupaba toda la pantalla y no era posible que quedase encajada de alguna forma. Además, al redimensionarla, no se podía imprimir en el laboratorio porque era demasiado grande para las impresoras que se dispone (la base de impresión de la impresora es de 20cm y la pieza es de 23cm de largo). Esperamos una semana a ver si era posible imprimirla de una pieza en impresoras externas a nosotros (otro laboratorio, grupo de investigación, empresa externa, etc.), pero no obtuvimos soluciones, ya sea porque no se disponía de la tecnología o porque el presupuesto era demasiado elevado.

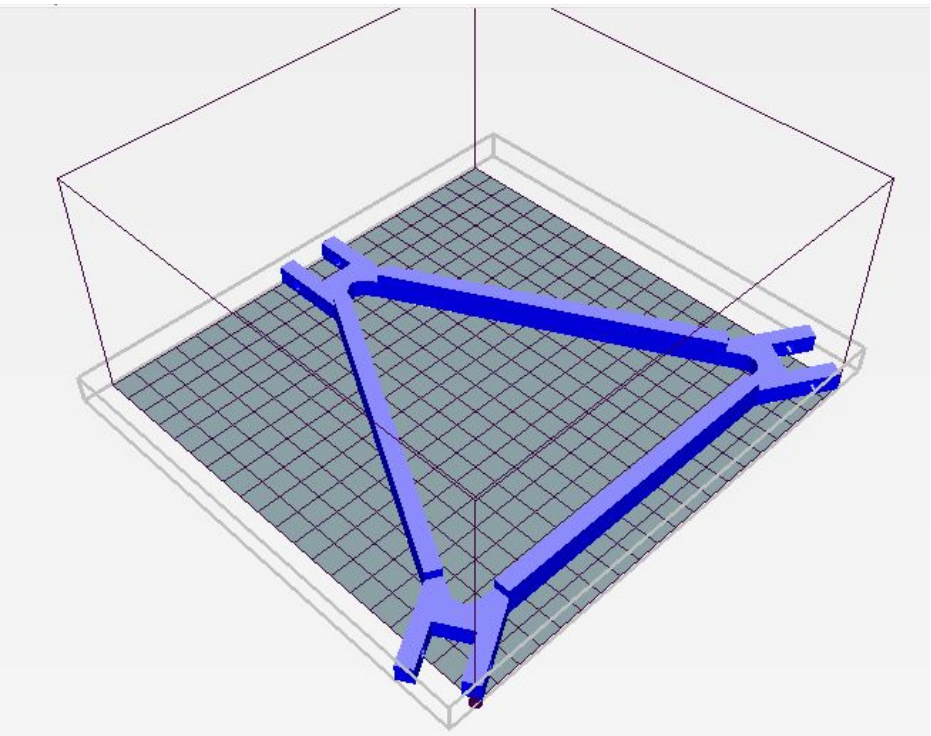


Figura 10.a.1.

Dividimos la figura 10.a.1 en tres partes semejantes en cuanto a dimensión para que al unirse, los puntos débiles no recaigan sobre los vértices, que es donde los servo hacen toda la fuerza, y se unan con pegamento instantáneo por el medio de los lados del triángulo, para que no haya sobrefuerzas en la montura.

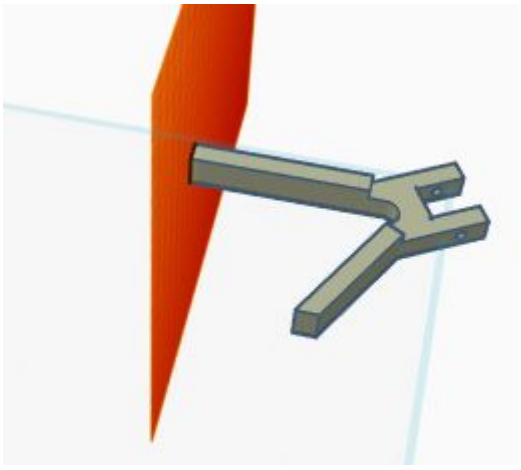


Figura 10.a.2.

Obtuvimos 3 motores servo (DS3218MG) del laboratorio de la universidad y dedicamos un día a entender su funcionamiento, ya que antes teníamos conocimientos de motores de menos potencia y voltaje que los que vamos a usar en esta práctica, que son motores que tienen una fuerza de 20kg.

Para que estos motores estuvieran fijos y un poco elevados de la base del proyecto, se diseñó la montura de la base (figura 10.a.3).

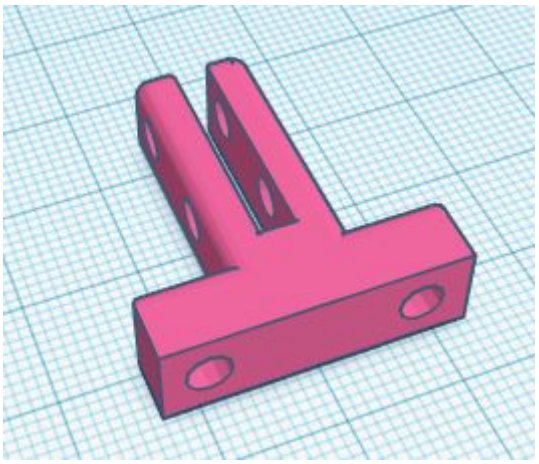


Figura 10.a.3.

Figura 10.a.4.

Se pone una por cada lado del servo para que quede fijado por las dos partes, y se ponen dos tornillos, para que queden unidos. En cuanto a estas monturas tuvimos varios problemas: el primero viene cuando las seis impresiones que hemos realizado no encajan porque se han realizado demasiado estrechas. Esto sucedió porque las dimensiones de las monturas las realizamos muy exactas sin dejar margen de unos milímetros de fallo en la impresión. Con una lija se consiguió solventar el problema, permitiendo que encajaran a la perfección. Además de lijar las impresiones, se tuvo que utilizar una sierra eléctrica para lijar parte del motor, ya que la parte de plástico donde encaja la montura no era recta. Una vez que encajaban, se utilizaron dos tornillos en cada lado de los servos para que se unieran correctamente. Al no tener rosca interior las impresiones, se utilizó una llave

Allen del grosor necesario para que con presión pudieran entrar los tornillos, ya que sin rosca no entran. Llegado este punto los motores están ya completados, y faltaría pegarlos a la base y unirlos con la pierna que transmite el giro a la superficie de la pantalla.

El siguiente problema viene cuando siguiendo el montaje de la referencia [1], no podemos imprimir las piernas del proyecto, las piezas que unen los servos con la pantalla y que permiten que la pantalla se balancee para que la bola consigua equilibrio para no salirse de la pantalla. El archivo para la impresión de esta pieza no estaba disponible para poder imprimirla ya que estaba en una extensión de un programa de licencia de pago, por lo cual decidimos desarrollar nosotros mismos nuestras propias piernas del proyecto. Es por esto que ideamos junto a D. Rubén Ferrero unas varillas de metal para que la pantalla coja la altura pertinente. Con una herramienta Dremel puesta para tallar, cortamos en tres partes iguales una vara metálica de 3 mm de diámetro, una idea sencilla e ideal que nos ahorró un gran comedero de cabezas.
De cara a unir el triángulo de la pantalla con los servos, compramos seis rótulas hembra para encajar el motor con la superficie, así las piernas no se giran y permiten un movimiento recto en el giro del motor. Las rótulas llegaron realmente pronto, pudiendo proseguir con la estructura. Tuvimos que hacer un par de modificaciones para adaptarlas a nuestro proyecto y nos sirvieran:

1. La cabeza donde se inserta el tornillo que se une con el motor y con la pantalla era giratorio, por lo tanto tuvimos que pegar con pegamento instantáneo para que el movimiento fuera recto.
2. Al ser rótulas hembra, no tenían la rosca hecha por dentro, entonces tuvimos que recurrir de nuevo, como con los tornillos en los motores, a utilizar una llave Allen junto a un tornillo del mismo diámetro que la varilla, para poder hacer la rosca, y luego introducir la varilla. Esto ha sido una labor costosa ya que la varilla no tiene mango y hay que meter en las seis rótulas la misma longitud de las varillas, para que queden iguales y la pantalla esté en plano.

Figura 10.a.5.

Una vez tenemos las varillas, pensando que habíamos solucionado el problema de no tener las piernas de la estructura, nos damos cuenta de que al redimensionar el triángulo de la pantalla, también crece de tamaño el vértice, donde el espacio para meter la rótula y que queden encajados el triángulo y la pierna ahora es excesivo, entonces no queda unida. Decidimos junto a D. Rubén Ferrero crear seis casquillos para imprimir enseguida en la impresora y tenerlos al momento, ya que son rápidos de conseguir.

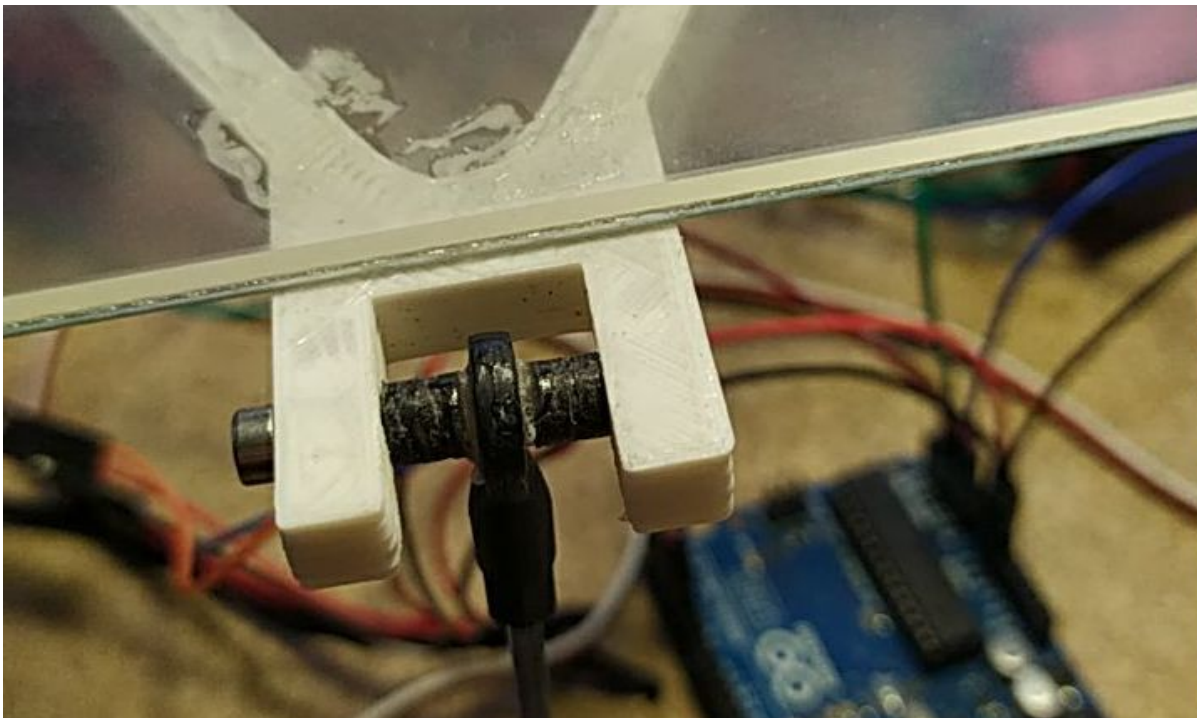


Figura 10.a.6.

De este modo la estructura está completa, a falta de unir los servos con la base inferior y la pantalla con la base superior.

Los siguientes días estuvimos solucionando problemas relacionados con la alimentación de los servos, al tener tres motores no podíamos alimentarlos únicamente con los 5V de la placa Arduino (ya que no proporciona 5V reales) conectada a un ordenador mediante puerto USB.
Primeramente decidimos conectar todo con una fuente de alimentación conectada a la luz de 5V. El proceso es sencillo: al salir dos cables de la fuente de alimentación, uno es el que transmite los 5V, y otro es la toma de tierra, conectarlos a la protoboard, y unir en serie cables para la alimentación de los tres motores, e igual para la toma de tierra.
Esto nos causó un problema, ya que la toma de tierra de todo el proyecto no tenía una única salida y debido a las características de la protoboard estaríamos perdiendo voltaje. Es por esto que para asegurar un circuito más directo donde evitaríamos posibles futuros fallos, decidimos hacer un empalme en la fuente de alimentación: en cada salida de la fuente soldamos cuatro cables: tres para los motores y uno para alimentar la placa de Arduino por la entrada *VIn*, en vez de alimentarlo por USB con el ordenador.

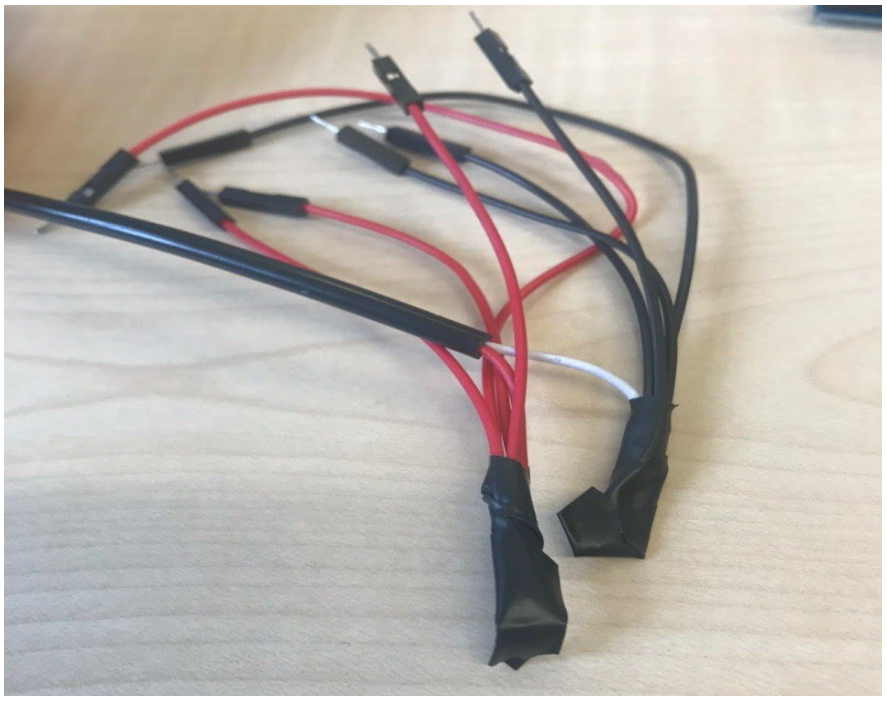


Figura 10.a.7.

El siguiente paso ha sido configurar las posiciones de los servos, para ello, tuvimos que desmontar el cabezal y mediante un simple código de Arduino, ejecutando el método *attach()* a los servos y unas pruebas con el método write [3] para dejar la referencia de la posición inicial y poner el cabezal adecuadamente.

Una vez configurados todos los servos, volvimos a juntarlos con sus monturas y procedimos a pegarlos con la base inferior. En la unión de los motores con la base teníamos bastantes dudas sobre qué sería mejor, si taladrar con tornillos o utilizar la pistola termodinámica.

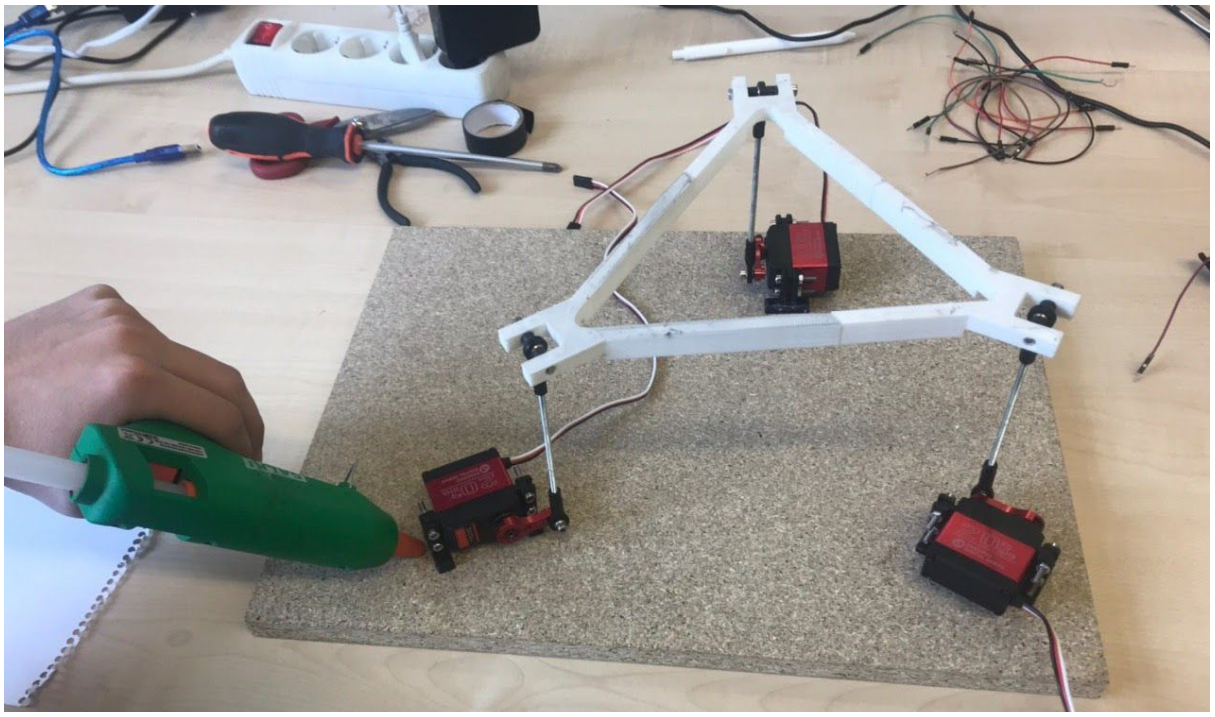


Figura 10.a.8.

Después de pegar la base inferior con los servos con la pistola termofusible y que reposara el pegamento, procedimos a pegar la pantalla con la base superior con pegamento instantáneo por la parte inferior de la pantalla, y aprovechamos a fijar un poco más las rótulas nuevamente con la pistola termofusible.

Una vez quedó todo fijo, la estructura estaba montada finalmente. Con un aspecto como el de la siguiente figura.

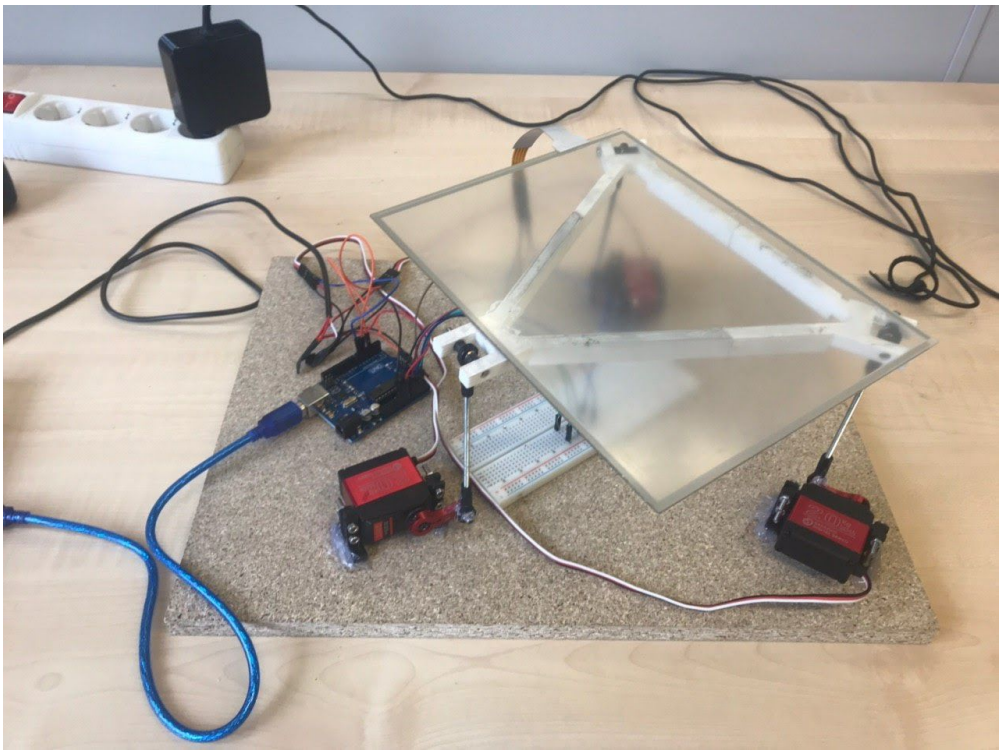


Figura 10.a.9.

A la hora de conectar los cables también tuvimos numerosos fallos: el primero fue controlar que la pantalla resistiva de 4 cables nos devolviera las dos coordenadas, para más tarde cuando se tuviera la estructura montada, poder trabajar con el software para controlar la bola. Se estuvo dos semanas para poder controlar el uso de la pantalla, ya que no venía con instrucciones y no nos habíamos encontrado anteriormente ante este tipo de pantallas, por lo tanto tuvimos que buscar numerosa información, donde la mayoría no nos servía, pero encontramos un vídeo donde lo explican bien y nos ha funcionado [9]. En el Anexo 2 podemos ver que las 4 salidas tienen que ir cada una a una entrada analógica del Arduino. Al tener 4 entradas, dos son para cada coordenada. Es por esto que se debe introducir una resistencia grande de por medio para que la corriente tenga oposición y recibir un valor 0 si no hay ninguna presión sobre la pantalla.
Clarificar que la pantalla, al ser un rectángulo, obtenemos dos coordenadas X e Y, por las que obtenemos la posición exacta de la bola. Las esquinas serían las coordenadas (0,0), (X máximo, 0), (0, Y máximo) y (X máximo, Y máximo). Los valores máximos de las coordenadas rondan 600-700 respectivamente.

En cuanto a la conectividad de los motores, tienen tres entradas: la de voltaje, la toma de tierra, y el valor que recibe para que realice el giro.

En el Anexo 3 vemos el plano electrónico de todo el proyecto, donde destacamos que la entrada de los 5 voltios sale de la fuente de alimentación para

los tres motores y la placa de Arduino, y la toma de tierra de todo el proyecto es un circuito cerrado.

Destacar que durante la última semana las salidas de valores de la pantalla no han sido correctas. Anteriormente cuando la bola no estaba sobre la pantalla, las salidas de la pantalla eran X=0 e Y=0. Lo que es la salida normal. De un día para otro, vemos que la salidas de la pantalla sin tener presión en la pantalla, donde debería dar valores neutros, obtenemos valores pequeños, de entre 30 y 70, es decir, como si hubiera una presión en la parte inferior izquierda de la pantalla. Esto hace que tengamos que modificar el código para obviar este fallo de la pantalla, habiendo intentado solucionarlo pensando que sería error de los conectores, una sobrecarga de voltaje, resistencias con pocos Ohmios… Al haber reemplazado todos los demás componentes y revisado el circuito innumerables veces, concluimos que el fallo es interno de la pantalla por causas ajenas a nosotros, ya que se le ha dado un buen trato. Es por esto que no podemos estabilizar la bola sobre la pantalla entera, pues inmovilizamos las acciones que tienen que ver con valores bajos de la pantalla.

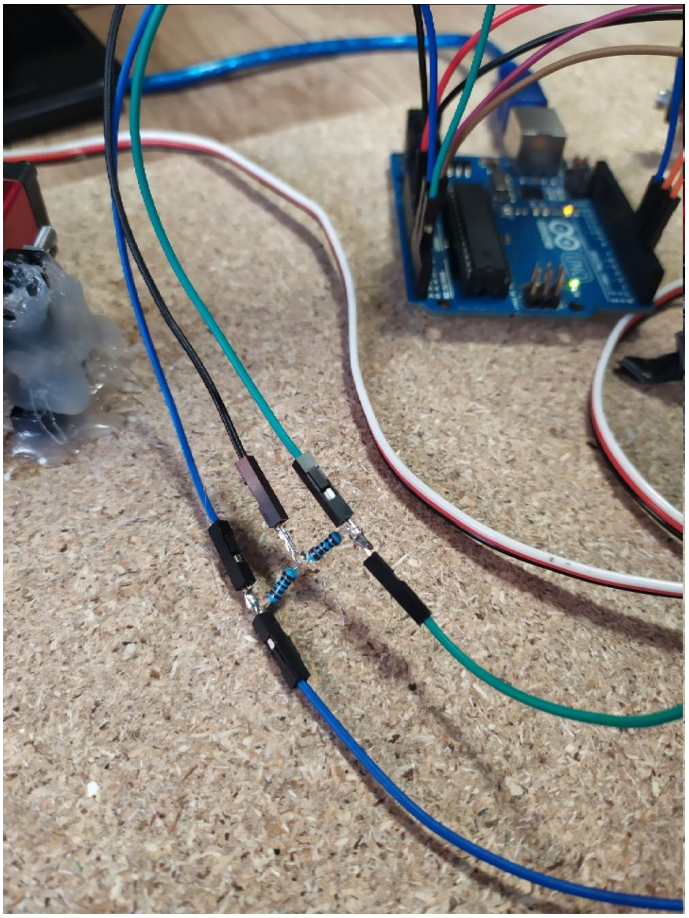


Figura 10.a.10.

Se ha llegado a obviar una protoboard que teníamos en el montaje por intentar encontrar el error, llegando a soldar las resistencias para evitar así el posible fallo que dan al no conectar correctamente los componentes en la protoboard.

Durante las dos últimas semanas del proyecto, mientras hacíamos las pruebas con el proyecto desarrollando el código, nos hemos encontrado con diferentes problemas que hemos solucionado mientras nos encontrábamos las trabas por el camino. Cuando subimos de nuevo el código mientras estamos realizando las pruebas, había algunos momentos donde los servos hacían un giro inesperado y el cabezal chocaba contra la madera, levantando el servo, y así despegando el pegamento, y luego ya volvía a la posición de inicialización. Esto nos llevó un quebradero de cabeza ya que despegaba los motores de la base, haciendo a veces inestable la estructura. Es por esto que se decidió poner más pegamento de la pistola sobre las monturas de los servos.

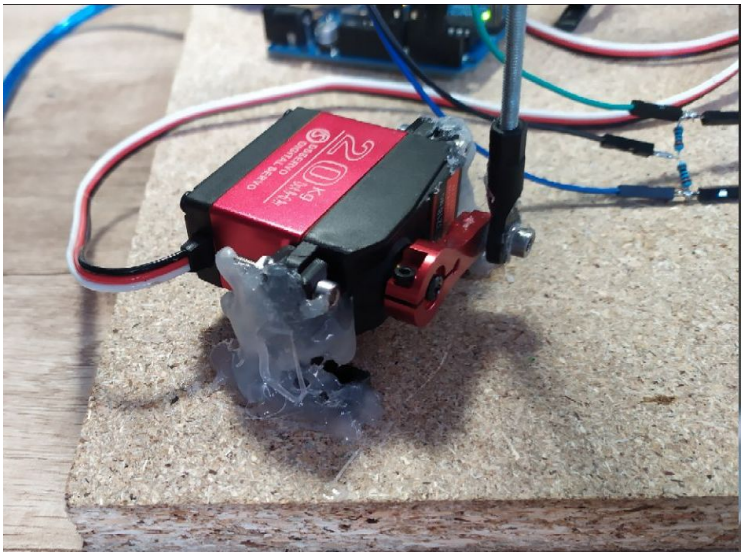


Figura 10.a.11.

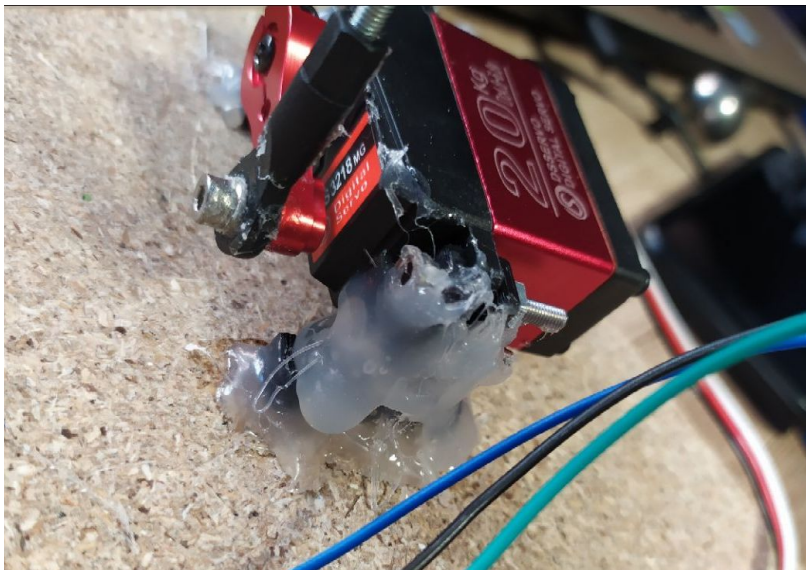


Figura 10.a.12.

En un movimiento de los comentados, las dos monturas de los servos parten a la mitad, viéndonos un lunes por la tarde sin la posibilidad de conseguir otras dos monturas. es por esto que se decide hacer una base de pegamento sobre las monturas, quedando como en las figuras de arriba. En ese momento quedan estables y podemos continuar con el proyecto.

Desarrollo del código

Para el desarrollo del código, nos hemos basado en el primer proyecto que encontramos en la plataforma Github [1] y en otro proyecto similar que encontramos buscando documentación, que utilizaba seis servos, dos por cada eje (Eje Y, Eje X izquierdo y Eje X derecho) [5].

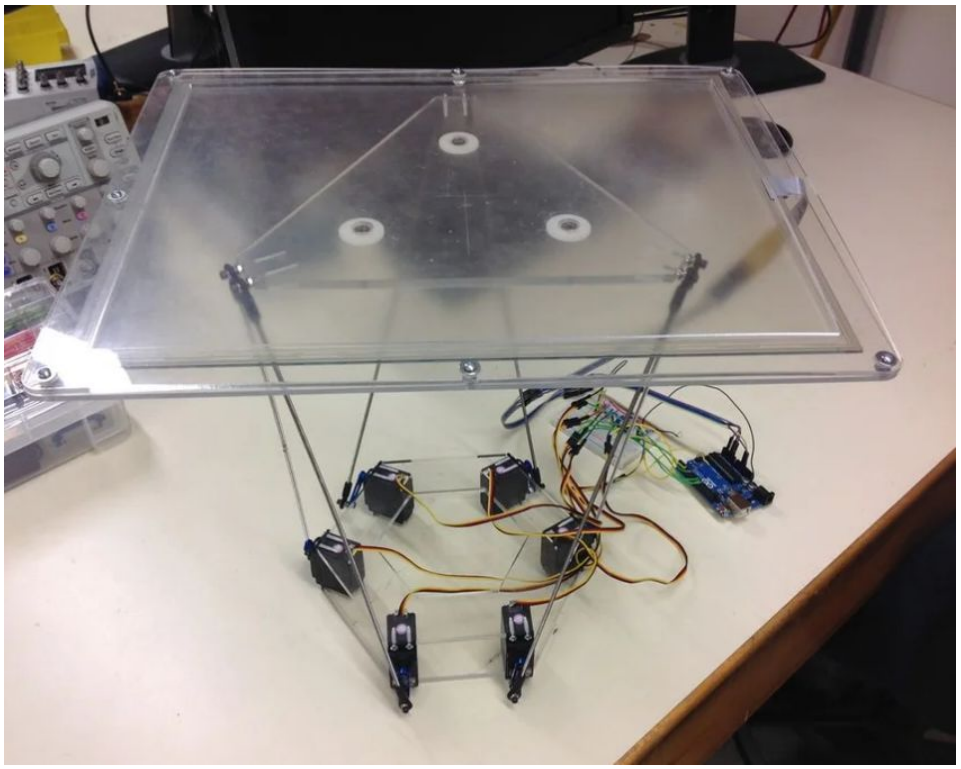


Figura 10.b.0.

Sin embargo, al ser este proyecto muy dependiente del tamaño de la pantalla, el código de ambos proyectos lo utilizamos como referencia. Es más, arreglamos ciertos errores del código a nuestro parecer para hacer funcionar nuestro proyecto. El código íntegro se puede ver en Anexo 1.

A continuación vamos a explicar los principales métodos/variables del código de forma que quede claro lo que hacen y su propósito.

Al principio del todo se especifica que la velocidad de datos en bits sea de 115.2000 para que la pantalla reaccione a tiempo a los movimientos de la pantalla.

Las variables globales principales son las siguientes:

- setkp, setki y setkd: Variables que almacenan los parametros proporcionales, integrados y derivados de cara a calcular la siguiente posición de giro de los servos.
- compT: Tiempo de computación de los servos en ms.
- isBall: Parámetro que almacena si la bola está sobre la pantalla o no.

La función **readTouchData()** se encarga de hacer *attach* a los pins de la pantalla resistiva y de pasar 5V para poder leer desde una entrada analógica y sacar así el valor xCor (coordenada X) e yCor (coordenada Y). Además tenemos una variable Booleana *isBall* que será *True* o Verdadero si está haciendo contacto la bola con la pantalla y será *False* o Falso si no hay bola. Esto nos sirve para realizar los cálculos de una manera más óptima.

La función **computeError(int i)** toma un número i para calcular el error 3 veces. En esta función nos tuvimos que detener bastante tiempo y debuggear debido a la escasa documentación que encontramos.
Encontramos la solución gracias a depurar el código e ir calculando el error poco a poco con ejemplos sencillos.

Simplemente para el primer servo, el del eje Y, calcula su error restando yCor que es el valor donde se encuentra la bola e yO, que es el *setpoint* especificado en el programa.
Para el segundo servo, el del eje X derecho, calcula su error restando xCor - xO puesto que como la X crece hacia su lado, a mayor X implica que la bola está hacia la derecha y por tanto el servo tiene que escribir un ángulo grande y levantar la plataforma, poniendo un ejemplo sencillo, si la xCor de la bola es 450, un valor próximo al máximo, siendo el *setpoint* de 250, la resta dará un valor de 200, al cual se le multiplica el valor de la raíz cuadrada de 3, por razones trigonométricas y para incrementar el valor producido y conseguir así un ángulo mayor.
A este valor se le resta yO - yCor ya que si la Y es mayor, el primer servo subirá y los servos de la X deben de bajar para inclinar así la plataforma. Con yO - yCor el servo baja cuanto mayor sea la Y. El valor obtenido es dividido entre 2 por los 2 servos del eje X.
Para el tercer servo, el del eje X izquierdo, el proceso es muy similar al anterior.
Se cambian de lado algunos valores como xO - xCor ya que como la X crece hacia la derecha, cuanto mayor sea la X significa que el tercer servo debe de bajar la plataforma e inclinarla. Con la coordenada del eje Y se demuestra lo mismo que con el anterior motor.

La función **computePID(int i)** como las anteriores, toma un número entero i para cada iteración del bucle, en nuestro caso 3, 1 por cada servo. Genera el *output*

para los servos gracias a los valores obtenidos de error en la función computeError(int i) y al algoritmo desarrollado para nuestro controlador PID.
Dicho algoritmo se basa en utilizar el error para calcular 4 términos: *PTerm, DTerm, ITerm* y *ATerm* y una vez calculados dichos términos se suman y dan como resultado el output.
Después de calcular el *output*, se almacenan los valores para calcular el *output* de la próxima iteración.

La función **limitOutput(int i)** toma como valor un número entero i porque va a ser llamada dentro de un bucle 3 veces, para limitar el *output* o el valor de salida que van a tener los servos. Hacemos esto porque en nuestro caso, no queremos nunca que los servos hagan un ángulo de menos de 70 grados, esto es porque pegaría con la base inferior y podría producir daños a la estructura.
La función simplemente compara el valor producido en el array *output* (una vez calculado ya) y si es mayor que la constante *outMax* que está definida en el principio del programa y para nuestro proyecto, después de varias pruebas la hemos dejado en 600.
Este número es debido a que estamos utilizando el método writeMicroseconds() [6] de la librería Servo.h, en vez del método write() puesto que nos proporciona más precisión a la hora de llegar a un ángulo en concreto. Como explica la función, el valor del parámetro de la función writeMicroseconds() puede oscilar desde 700 hasta 2300.
Como nosotros hacemos un writeMicroseconds(1600 + output[i]), el *output* como máximo podrá ser de 2200 microsegundos y mínimo 1500 debido a la constante *outMin* declarada al principio también que tendrá un valor de -100.

La función **writeServos()** ordena a los servos que escriban el ángulo introducido como parámetro (en microsegundos). Utilizamos como parámetro el array *output* que tiene almacenado el valor deseado para los servos.

La función **controlPID()** es la función que une al resto, si el booleano *isBall* es True se ejecuta un bucle *for* de 3 iteraciones donde se llama a la funcion readTouchData() para leer los datos, después se calcula el error producido con computeError(), a continuación se calcula el *output* con computePID() y por último se limita el *output* con limiteOutput(). Además si se ejecuta todo sin ningún problema, la función devuelve un True para gestionarlo desde el bucle del main, si no, devuelve un False.

El programa se ejecuta en un bucle en el que primero se leen las coordenadas con readTouchData() y después se llama a la función controlPID(), cuando esta acaba, con los arrays de *output* actualizados, se llama a la función writeServos().

Esto se ejecuta en bucle y hace que la bola se intente estabilizar siempre en el medio de la pantalla, a no ser que se introduzca por consola algún caracter. Si se envía por consola un ‘1’, se activa la función square(); si se envía un ‘2’, se activa la función line(); y si se envía un ‘3’, se activa la función triangle().
La función **square()** simplemente cambia el setpoint a unos valores predeterminados para formar un cuadrado.
La función **line()** hace que la bola se mueva en una línea recta en sobre la pantalla.
La función **triangle()** permite que la bola se mueva en triángulo.

Resultados obtenidos

Como se ha ido documentando durante toda la memoria, el resultado que hemos obtenido tras estas semanas es el controlador PID que sostiene a la bola sobre la pantalla resistiva. El proyecto ha sido un éxito rotundo donde hemos conseguido junto a la estructura bien montada y el código perfectamente hecho, que la bola pueda sostenerse sobre la pantalla sin darle ninguna orden.

Si queremos solo conectarlo a la corriente, sin que esté conectado al ordenador, solo tendrá la función en la que la bola siempre quiere ir al punto medio de la pantalla. Por otro lado, si se alimenta con la fuente y además se conecta mediante USB al ordenador donde está el código, se podrán enviar impulsos al Arduino para que cambie el modo de tratar a la bola: que se mueva en línea recta, triángulo y cuadrado.

Durante la última semana del proyecto donde hemos avanzado en gran medida a la mejora del código software, hemos tenido la desventaja de no poder trabajar con el 10% de la parte izquierda de la pantalla, por lo que ha sido una gran frustración el no poder enseñar el proyecto en su fase perfecta.

Además, hemos conseguido sintonizar las constantes del controlador PID de modo que el *output* que produzca sea más “agresivo” cuanto más error arroje la posición de la bola con respecto al *setpoint* establecido. Es decir, cuando el *setpoint* esté en el punto medio de la pantalla, si la bola se encuentra en las esquinas, los servos emitirán un *output* mucho mayor para evitar así que la bola se caiga.

Esto último ha resultado complicado con la complejidad añadida de que la pantalla dejó de funcionar al 100% en los últimos momentos del desarrollo del proyecto, sin tener tiempo para poder cambiar la pantalla por otra o encontrar una solución acorde.

Presupuesto del proyecto

En la siguiente tabla se describe el coste de los materiales del proyecto:

Herramienta	Unidades	Precio por unidad	Total
Placa de Arduino UNO	1	2,30€	2,30€
Motor servo DS3218MG	3	17,99€	53,97€
Resistencias de 10 MΩ	2	0,74€	1,48€
Pantalla resistiva 22.5x17.3 cm [8]	1	15,67€	15,67€
Rótulas hembra de M4	6	1,13€	6,80€
Varillas de metal de 2mm. Ø de 15 cm	2	2,24€	4,48€
Bola de metal de 2mm. Ø	1	3,40€	3,40€
Base de conglomerado de 1.5mm de ancho y 40x30cm	1	2,30€	2,30€
Cables conectores	12	0,21€	2,60€
Fuente de alimentación de 5V	1	9.99€	9,99€
Tornillos M4	12	0,06€	0,73€
Tuercas M4	12	0,07€	0,93€
Pistola de pegamento termofusible	1	5,80€	5,80€
Pegamento termofusible	5 barras	0,48€	2,40€

Impresora 3D	1	95€	95€
Filamento 3D	300g	7,89€	7,89€
Soldador	1	22€	22€
Pegamento instantáneo	3	1,80€	5,4€
Máquina Dremel multiherramienta	1	159€	159€

El coste total de los materiales del proyecto es de 402,14 euros. A esto hay que sumarle el coste de las personas que están trabajando en el proyecto. Teniendo en cuenta que somos dos personas en el proyecto, y hemos trabajado 140 horas cada uno en el proyecto, donde cobramos 15€ la hora, el coste de los trabajadores es de 4200€. Siendo así, el presupuesto final del proyecto es de **4602,14€**.

Posibles mejoras o evoluciones

Hemos dejado en el proyecto una buena base donde se puede innovar en muchas partes. El propósito inicial del proyecto era la creación de una aplicación móvil donde las pulsaciones del dedo sobre la pantalla indicarían la posición donde la bola se estabilizaría en la pantalla. Esto supondría un mejor manejo de la bola, pudiendo hacer cualquier movimiento o secuencia sin tener que preprogramarlo en el código del programa. Además, se podría también programar en una parte de esa aplicación en qué coordenadas en concreto puede la bola quedarse quieta. A raíz de eso, también se puede seleccionar si se quiere mover la bola de alguna forma predeterminada: triángulo equilátero, cuadrado, pentágono, etc.
Otra alternativa puede ser la creación de una página web junto a un servidor web (sencillamente con el Arduino, o si se quiere liberar de memoria la placa de Arduino, juntarlo con una RaspberryPi o similar).

Aumentar el diámetro de la bola también puede ser buena idea si se quiere que el proyecto sea más vistoso y se quiera exponer en algún concurso.

En cuanto a evoluciones dentro del programa, además de las mejoras, proponemos la integración de una pantalla LCD debajo de la pantalla resistiva. Con esto conseguimos que en la posición donde está la bola se pueda visualizar un color sobre la pantalla. En el caso, por ejemplo, de que la bola se quiera mover en cuadrado, se pueden encender cuatro focos en la pantalla para que se vea qué recorrido va a llevar.

Conclusiones

Es un proyecto que los dos veíamos al principio muy ambicioso y visual, pero por otro lado sabíamos que nos iba a suponer un gran esfuerzo en cuanto a tiempo y trabajo investigando por nuestra cuenta cómo funcionaba cada parte por separado para después juntarlo.

Además, hemos sabido llevar el solucionar muchos problemas que no nos dejaban proseguir el proyecto, ya sea por fallos o desconocimiento. Se ha aprendido de ellos para no cometerlos de nuevo. Estas últimas semanas se ha trabajado más de lo habitual para poder completarlo y dejarlo presentable.

Ha sido un proyecto no trivial de completar, cada vez que se intentaba arreglar algo aparecía otro error, sobre todo en la parte del montaje de la estructura, en la que carecíamos de experiencia.

A pesar de esto, lo consideramos un proyecto exitoso debido a que hemos cumplido la gran parte de los objetivos iniciales, los cuales se decidieron en poco tiempo y sin tener realmente conocimiento sobre la dificultad real del proyecto.

Hemos aprendido mucho al montar toda la estructura y al dedicar tantas horas al código del proyecto y aunque cada uno nos hayamos especializado más en una parte que en otra, siempre nos hemos ayudado con los errores y demás, es por ello que ahora tenemos bastante conocimiento tanto de electrónica como de manejar servos y cómo funcionan los controladores PID, no solo en nuestro caso si no que en general, además de saber sintonizarlos correctamente acorde al resultado necesario.

Ambos afrontamos el proyecto con mucha ilusión al ser una materia desconocida y no nos ha defraudado. De hecho hemos pensado en hacer más proyectos DIY porque es una buena experiencia y una excusa tanto para trabajar en equipo como para aprender más y reflejarlo en proyectos útiles en la vida real.

Bibliografía

[1] : J. Schwert, "schwertJake/Ball-On-Plate-Machine", GitHub, 2020. [Online]. Available: https://github.com/schwertJake/Ball-On-Plate-Machine.
[2] : "Controlador PID - Control Automático - Picuino", Picuino.com, 2020. [Online]. Available: https://www.picuino.com/es/arduprog/control-pid.html. [
[3] : "Arduino - Write", Arduino.cc, 2020. [Online]. Available: https://www.arduino.cc/en/Serial.Write.
[4] : "Ball and Plate", Arduino Project Hub, 2020. [Online]. Available: https://create.arduino.cc/projecthub/davidhamor/ball-and-plate-c48027.
[5] : "PID Controlled Ball Balancing Stewart Platform", Instructables, 2020. [Online]. Available: https://www.instructables.com/id/PID-Controlled-Ball-Balancing-Stewart-Platform/.
[6] : "Arduino - ServoWriteMicroseconds", Arduino.cc, 2020. [Online]. Available: https://www.arduino.cc/en/Reference/ServoWriteMicroseconds.
[7]O. B, "Pantallas táctiles: capacitivas vs resistivas", *Xatakamovil.com*, 2020. [Online]. Available: https://www.xatakamovil.com/desarrollo/pantallas-tactiles-capacitivas-vs-resistivas.
[8]"10,4 pulgadas 4 cables para LQ104V1DG52/51 G104SN03 V.1 AMT 9509 225*173mm pantalla táctil resistiva digitalizador-in Paneles y LCD de tableta from Ordenadores y oficina on AliExpress", *aliexpress.com*, 2020. [Online]. Available: https://es.aliexpress.com/item/32815591993.html.
[9]"How to use a 4 wire resistive touchscreen with an Arduino", *Youtube.com*, 2020. [Online]. Available: https://www.youtube.com/watch?v=XlkIjnTbxH0.

Anexo 1.

```
#include <Servo.h>
#include <inttypes.h>
#include <stdlib.h>


#define r3 1.73205 // square root of 3

#define servopin1 6
#define servopin2 5
#define servopin3 3

#define outMax 2100-1500
#define outMin -1500+1250

char controlByte = '0';

Servo servo1; //servo on y axis
Servo servo2; // x right servo
Servo servo3; // x left servo

double xCor, yCor; //Coordinates from touchscreen
double xO, yO;

double error[3];//holds current errror
double lastError[3] = {0,0,0};
double lastLastError[3] = {0,0,0};

double output[3] = {0,0,0}; //Hold actual output

double ITerm[3] = {0,0,0};
double lastDVal[3] = {0,0,0};
double lastLastDVal[3] = {0,0,0};
unsigned long now, deltaT, lastT;

double setkp[3] = {0.3,0.5,0.5};
double setki[3] = {0.15,0.2,0.2};
double setkd[3] = {0.1,0.14,0.14};
double setka[3] = {1,1,1};
```

```
double kp[3], ki[3], kd[3], ka[3]; // working tunings
int compT; //computation time in ms

bool compute_bool = true, computeDone = true, setPointChange = false, isBall =
false;

int auxTimeCount = 100, methodCount = 0;

int squareCoords[4][2] = {{380,360}, {360,185}, {205, 135}, {180, 310}};
int lineCoords[2][2] = {{259,146}, {285,377}};
int triangleCoords[3][2] = {{272,246},{408-30,328-30},{372-30,165+30}};

void setup() {
  // put middleYur setup code here, to run once:
  Serial.begin(115200);

  servo1.attach(servopin1);
  servo2.attach(servopin2);
  servo3.attach(servopin3);
  servo1.writeMicroseconds(1500);
  servo2.writeMicroseconds(1500);
  servo3.writeMicroseconds(1500);
  compT = 10;
  xO = 250;
  yO = 229;
  setPIDTunings(true);
}

void loop() {
  // put middleYur main code here, to run repeatedly:
  readTouchData();
  if(isBall){
    if(compute_bool){
      if(controlPID()){
        writeServos();
        computeDone = true;
      }
    }
    if(computeDone){
      receiveSerial();
      switch(controlByte){
```

```
        case('1'):
          auxTimeCount++;
          Serial.println(auxTimeCount);
          if(auxTimeCount > 100){
            square();
            auxTimeCount = 0;
            setPointChange = true;
          }else{
            break;
          }
        case('2'):
          auxTimeCount++;
          if(auxTimeCount > 25){
            line();
            auxTimeCount = 0;
            setPointChange = true;
          }else{
            break;
          }
        case('3'):
          auxTimeCount++;
          if(auxTimeCount > 25){
            triangle();
            auxTimeCount = 0;
            setPointChange = true;
          }else{
            break;
          }
      }
    }
  }
}

void readTouchData(){
  pinMode(A2, INPUT);
  pinMode(A3, INPUT);
  digitalWrite(A2, LOW);
  pinMode(A1, OUTPUT);
  digitalWrite(A1, HIGH);
  pinMode(A0, OUTPUT);
  digitalWrite(A0, LOW);
  xCor = analogRead(A3);
```

```
  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  digitalWrite(A0, LOW);
  pinMode(A3, OUTPUT);
  digitalWrite(A3, HIGH);
  pinMode(A2, OUTPUT);
  digitalWrite(A2, LOW);
  yCor = analogRead(A1);
  // printCoords();
  if(xCor > 95 && yCor > 40){
    isBall = true;
  }else{
    isBall = false;
    readTouchData();
  }
}

void limitOutput(int i){

  if(output[i] > outMax){
    output[i] = outMax;
  }
  else if(output[i] < outMin){
    output[i] = outMin;
  }
  if(ITerm[i] > outMax){
    ITerm[i] = outMax;
  }
  else if(ITerm[i] < outMin ){
    ITerm[i] = outMin;
  }
}

bool controlPID(){
  now = millis();
  deltaT = now - lastT;
  if(isBall){
    for(int i = 0; i<3; i++){
      readTouchData();
      computeError(i);
      computePID(i);
```

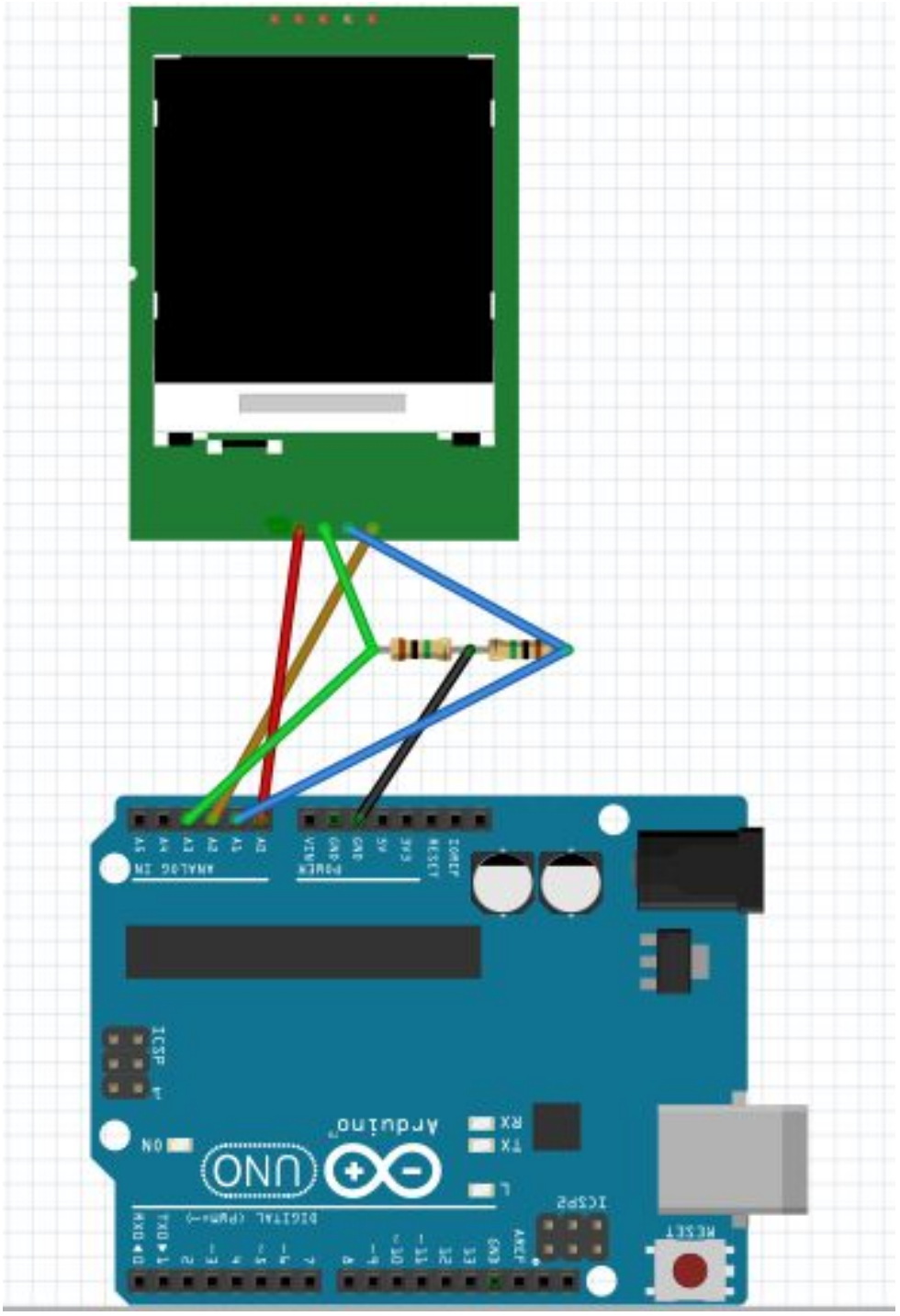
```
      limitOutput(i);
    }
    lastT = now;
    return true;
  }else{
    return false;
  }
}

void computeError(int i){
  if (i == 0){
    error[0] = 1.1*(yCor - yO);
  }
  if (i == 1){
    error[1] = (r3*(xCor - xO) + (yO - yCor))/2.0;
  }
  if (i == 2){
    error[2] = (r3*(xO - xCor) + (yO - yCor))/2.0;
  }
  printError(i);
}

void computePID(int i){
  Serial.println("/////COMPUTE//////");

  // calculate each PID term individually:
  double PTerm = (error[i] * kp[i]);
  ITerm[i] += (ki[i] * error[i]);
  double DVal = (error[i] - lastError[i]);
  double DTerm = kd[i] * (((5*DVal) + (3*lastDVal[i]) + (2*lastLastDVal[i]))/48.0);
  double ATerm = ka[i] * (((DVal - lastDVal[i]) + (lastDVal[i] - lastLastDVal[i]))/2.0);
  Serial.print("ERROR:");
  Serial.println(error[i]);
  Serial.print("LASTERROR:");
  Serial.println(lastError[i]);
  Serial.print("DVAL:");
  Serial.println(DVal);
  Serial.print("LASTDVAL:");
  Serial.println(lastDVal[i]);
  Serial.print("LASTLASTDVAL:");
  Serial.println(lastLastDVal[i]);
  Serial.print("PTERM:");
```

```
  Serial.println(PTerm);
  Serial.print("DTERM:");
  Serial.println(DTerm);
  Serial.print("ATERM:");
  Serial.println(ATerm);
  Serial.print("ITERM:");
  Serial.println(ITerm[i]);

  // Calculate Output:
  output[i] = PTerm + ITerm[i] + DTerm + ATerm;
  Serial.print("OUTPUT:");
  Serial.println(output[i]);
  // save some calculations for later:
  lastError[i] = error[i];
  lastLastDVal[i] = lastDVal[i];
  lastDVal[i] = DVal;
}

void writeServos(){
  Serial.println("WRITE//////////////////////////////");
  Serial.println(output[0]);
  Serial.println(output[1]);
  Serial.println(output[2]);
  Serial.println("END WRITE//////////////////////////");
  servo1.writeMicroseconds(1500 + output[0]); //1500 cambiar outmax
  servo2.writeMicroseconds(1500 + output[1]);
  servo3.writeMicroseconds(1500 + output[2]);
}

void setPIDTunings(bool aggressive){
  if(!aggressive){
    for(int i = 0; i < 3; i++)
    {
      kp[i] = setkp[i];
      kd[i] = setkd[i] / (compT/1000.0);
      ki[i] = setki[i] * (compT/1000.0);
      ka[i] = setka[i] * (compT/1000.0);
    }
  }
  else{
    for(int i = 0; i < 3; i++)
    {
```

```
      kp[i] = setkp[i];
      kd[i] = setkd[i] / (compT/1000.0);
      ki[i] = setki[i] * (compT/1000.0);
      ka[i] = setka[i] * (compT/1000.0);
    }
  }
}
/* Changing setpoint to square */

void square(){
  Serial.println("SQUare has been summoned!");
  xO = squareCoords[methodCount][0];
  yO = squareCoords[methodCount][1];
  Serial.println("XO");
  Serial.println(xO);
  Serial.println("YO");
  Serial.println(yO);
  methodCount++;
  if(methodCount > 3){
    methodCount = 0;
  }
}

void line(){
  Serial.println("Line has been summoned!");
  xO = lineCoords[methodCount][0];
  yO = lineCoords[methodCount][1];
  methodCount++;
  if(methodCount > 1){
    methodCount = 0;
  }
}

/* Changing setpoint to triangle */

void triangle(){
  Serial.println("Triangle has been summoned!");
  xO = triangleCoords[methodCount][0];
  yO = triangleCoords[methodCount][1];
  methodCount++;
  if(methodCount > 2){
    methodCount = 0;
```

```
  }
}

/* Methods for changing state */
bool receiveSerial()
{
  if(Serial.available() > 0)
  {
    Serial.print("Aqui hay algo");
    int incomingByte = Serial.read();
    controlByte = (char)incomingByte;
    Serial.println(controlByte);
    return true;

  }
  else return false;
}

void printCoords(){
  Serial.print("x=");
  Serial.print(xCor);
  Serial.print("       y=");
  Serial.print(yCor);
  Serial.println();
}

void printError(int i){
  Serial.println("ERROR ////////////////////////////");
  Serial.print(i);
  Serial.println(error[i]);
  Serial.println("YCor////////////////////////////");
  Serial.println(yCor);
  Serial.println("YO////////////////////////////");
  Serial.println(yO);
  Serial.println("XCor////////////////////////////");
  Serial.println(xCor);
  Serial.println("xO////////////////////////////");
  Serial.println(xO);
}
```

Anexo 2.



Anexo 3.