UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño e implementación de un prototipo para resolución automática del cubo de Rubik

Trabajo Fin de Máster

Máster Universitario en Ingeniería Mecatrónica

AUTOR/A: Monzón Vivas, Agustín

Tutor/a: Casanova Calvo, Vicente Fermín

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Final de Máster

DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO PARA RESOLUCIÓN AUTOMÁTICA DEL CUBO DE RUBIK

Autor:

Agustín Monzón Vivas

Tutor:

Vicente Fermín Casanova Calvo

Resumen

La fabricación de un mecanismo capaz de resolver el cubo de Rubik es un problema atractivo a la par que complejo, el cual se basa en el uso de algoritmos con los cuales se obtiene la secuencia óptima para la resolución del mismo. En este trabajo final de máster se propone una variación de este problema, diseñando e implementando un mecanismo capaz de seguir la secuencia dada por el algoritmo, centrándose así solo en la parte de diseño y fabricación del mecanismo.

El sistema se compondrá de cuatro brazos acabados en una pinza los cuales harán rotar las caras laterales del cubo mediante servomotores. De la misma manera, para la rotación de las caras superior e inferior, dos de las pinzas se soltarán, momento en el cual las otras dos rotarán al unísono dejando las caras mentadas al alcance de las pinzas anteriormente sueltas.

La realización de este proyecto implica la aplicación de conocimientos adquiridos durante el máster, tales como técnicas avanzadas de control, modelado de sistemas dinámicos, diseño CAD, simulación de sistemas dinámicos e implementación de control discreto en sistemas embebidos.

El proyecto empezará por el diseño en CAD del sistema empleando la librería Simscape Multibody de Matlab/Simulink para simular su comportamiento y comprobar si es fabricable. Seguidamente se realizará el código capaz de seguir la secuencia dada por el algoritmo y, finalmente, se pondrá en práctica el sistema haciendo uso de un Arduino que comunique el códgo de MATLAB con el sistema.

Índice

Page

Índice de Figuras

Índice de Tablas

1 Introducción

El presente trabajo, "Diseño e implementación de un prototipo para resolución automática del cubo Rubik", del cual vamos a describir las características principales y funcionamiento del prototipo, consistirá en conseguir un sistema capaz de resolver el puzle tridimensional conocido como Cubo de Rubik moviendo sus caras.

1.1 Objetivos

El objetivo de este trabajo es diseñar un sistema capaz de resolver el puzle tridimensional conocido como Cubo de Rubik o Cubo Mágico.

Por otro lado, el sistema está diseñado para resolver únicamente la versión de $3 \times 3 \times 3$ dado que la complejidad de la resolución de esta versión ha sido calificada como alta.

Por lo tanto, la motivación detrás de la elección de este sistema para la realización del Trabajo Final de Máster es la siguiente:

- La complejidad del sistema para la realización de movimientos rotativos en seis ejes distintos hace de este problema un reto a nivel mecánico.

- El uso de herramientas no empleadas durante el curso, pero que están dentro de las extensiones del entrono de programación mas usado en el máster, plantea todo un reto.

- La programación y control del microcontrolador Arduino UNO mediante el lenguaje de programación *Matlab*.

- Y, por último, el especial interés y pasión del autor por los puzles en tridimensionales y todas sus variantes.

1.2 Estructura de la memoria

El documento se estructura en siete apartados, cada uno de los cuales consta de una breve introducción.

- El primer punto, es una breve introducción que sirve de prólogo para ayudar a comprender mejor las motivaciones, los objetivos y la estructura del trabajo en cuestión.

- El segundo punto, hacemos un pequeño resumen de la historia del Cubo de Rubik, para poder analizar los algoritmos posibles.

- El tercer punto consistirá en explicar el diseño del CAD del sistema.

- El cuarto punto, vamos a desarrollar la simulación y comportamiento del movimiento del sistema.

- El quinto punto documenta la implementación real, incluyendo la fabricación y el ensamblaje de las piezas y las conexiones del sistema y para terminar la programación del microcontrolador.

- El sexto punto, o pliego de condiciones, recoge las delimitaciones generales, técnicas, económicas y legales que determinan la viabilidad, seguridad y correctitud del proyecto.

- El séptimo punto desarrolla el presupuesto del proyecto desglosado para su mejor comprensión.

- El octavo punto, o conclusiones y propuestas de mejora, refleja la experiencia vivida, el aprendizaje adquirido y las lecciones aprendidas por el autor durante el desarrollo del proyecto. Así como unas indicaciones y propuestas para mejorar el proyecto en un futuro.

- Y, por último, los anexos y referencias.

2 Estudio Teórico

2.1 Cubo de Rubik

El Cubo de Rubik, inventado a finales la década de 1970' por Ernő Rubik, es uno de los puzles de combinatoria espacial más famosos de la historia. La versión estándar fue creada por el arquitecto húngaro a principios de la década mientras buscaba la manera de mostrar movimiento en tres dimensiones a sus estudiantes. Esta versión primitiva estaba hecha de madera y papel unido mediante gomas elásticas, pegamentos y clips. Lo llamó 'Bűvös kocka' o Cubo Mágico. [1]

Siendo el puzle más popular hasta la fecha, habiendo vendido más de 350 millones antes de 2018, el Cubo de Rubik ha inspirado películas, obras de arte e incluso concursos tales como la competición de velocidad 'speedcubing'. Sin embargo, el más sorprendido de la trayectoria tomada por el puzle fue su propio creador. [1]

La versión estándar del Cubo de Rubik consiste en pequeños cubos formando una única pieza de 3x3x3 pintados de diferentes colores pero manteniendo cada cara del mismo color. Cada plano formado por 3x3x1 cubos es nombrado como 'corona' y estas pueden rotar 90°, 180° o 270°. Siendo el principal objetivo de dicho puzle el de desordenarlo mediante movimientos aleatorios y buscar la manera de volver a ordenarlo a su estado original. [2]

La dificultad del Cubo reside en la ingente cantidad de posibles combinaciones a las que se pueden llegar partiendo desde cualquier punto. En concreto, hay un total de 4.3252×10^19. Esto hace que la dificultad de la resolución de este puzle sea muy elevada y que se requiera de una estrategia para poder resolverlo la cual consistirá en una serie de movimientos también llamados macro-operaciones. Normalmente, dichas estrategias de resolución suelen requerir de más de 50 movimientos. Sin embargo, se ha demostrado que es posible resolverlo en menos de 20 movimientos independientemente de la configuración inicial. Esta cifra de movimientos se conoce como Número Dios y ha mantenido en vilo a muchos matemáticos. [3]

2.2 Algoritmo de resolución del Cubo de Rubik

A finales de la década de 2010' un estuido realizado en la Universidad de California logró desarrollar un algoritmo capaz de resolverlo en el meno número posible de movimientos. [4]

Este algoritmo fue desarrollado gracias a una teoría matemática llamada teoría de grupos. Gracias a ello, hoy disponemos de un algoritmo al cual le introducimos la configuración inicial y este nos devuelve la secuencia a seguir para su resolución en el mínimo numero de pasos.

Dado que desarrollar una aplicación mediante la cual introduzcamos la configuración inicial y nos devuelva la secuencia empleando el algoritmo ya creado sería muy costoso, se ha optado por emplear una aplicación ya creada en la cual simplemente haya que introducir nuestra configuración y obtengamos como respuesta la secuencia a seguir.

3 Diseño

En el siguiente apartado se procederá a la explicación del diseño *CAD* de las diferentes piezas que conformarán nuestro robot y el cual será empleado en la posterior simulación del mismo.

Todos los componentes han sido diseñados por el autor mediante el software *Solid Works 2019* a través del cual se obtendrán los archivos con extensión *STEP*. Esto es debido a que, posteriormente, el software de simulación requerirá de dicha extensión de archivos.

Por último, cabe destacar que, a pesar de los componentes haber sido diseñados por el autor, los servo motores son una simplificación en *CAD* para poder visualizar su posición en la simulación y comprobar que ningún componente pueda colapsar con ellos.

3.1 Base del robot

En primer lugar y a modo de base sobre la cual el robot estará ubicado, se ha optado por una plancha de madera cuadrada (figura 3.1) de $350mmx350mm$ a la que se le han realizado una serie de orificios donde irán colocados los elementos explicados en los apartados posteriores. Pensado inicialmente como estructura lista para cortar por láser y facilitar la fabricación

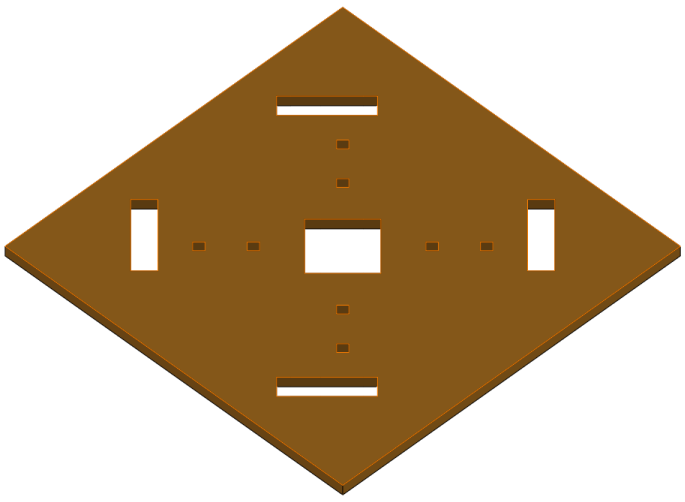


Fig. 3.1: Modelado de la base del robot sobre la cual irá ensamblado todo el robot.

Empezando de fuera a dentro, los orificios rectangulares que observamos en la figura 3.1 serán los que alojarán las bases sobre las cuales los servo motores irán sujetos. De esta manera, al ser un ensamblaje por presión, será fácil y cómodo remplazar cualquier servo motor que pueda fallar.

Seguidamente, los orificios cuadrados pequeños alojarán los raíles sobre los cuales se deslizarán los servo motores dedicados a la manipulación de cubo de Rubik. De la misma manera que para la base de los servo motores exteriores, el ensamblaje será por presión para facilitar el remplazamiento o modificación de cualquiera de los brazos de nuestro robot.

3.2 Base de los servo motores exteriores

Siguiendo desde la zona más exterior de la base, se procede a detallar el diseño de la pequeña estructura donde los servo motores irán alojados. Dicha estructura es una pieza maciza en forma de U (figura 3.2) sobre la cual se atornillarán los servo motores creando así una unión mecánica. Al igual que la base del robot, el diseño de esta pieza ha sido pensado en primera instancia para ser fabricado por corte láser sobre un conjunto de planchas de material sólido pegadas entre si.

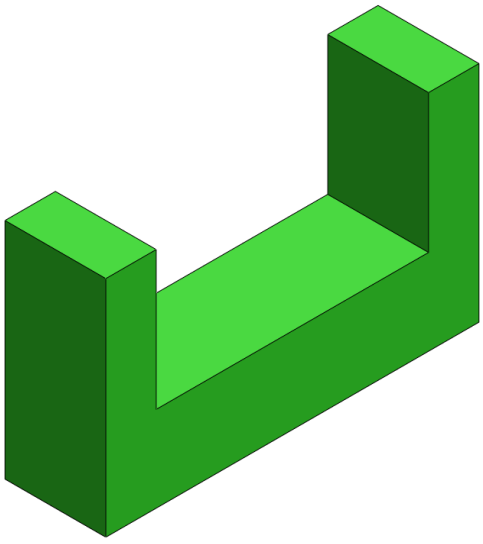


Fig. 3.2: Modelado de las sujeciones de los servo motores con la base del robot.

3.3 Raíl del carro

Seguidamente, sobre los pequeños orificios en forma de cuadrados de la figura 3.1 se colocarán 4 raíles sobre los cuales los servo motores que actúan las pinzas tendrán un grado de libertad de movimiento. Esto permitirá a las pinzas alejarse o acercarse al cubo de Rubik, optimizando así los movimientos de las pinzas. Dichos raíles (figura 3.3) se componen de dos pequeños pilares alojados en las cavidades cuadradas de la base y una tabla sobre la cual un carro, portador de los servo motores, podrá deslizarse.

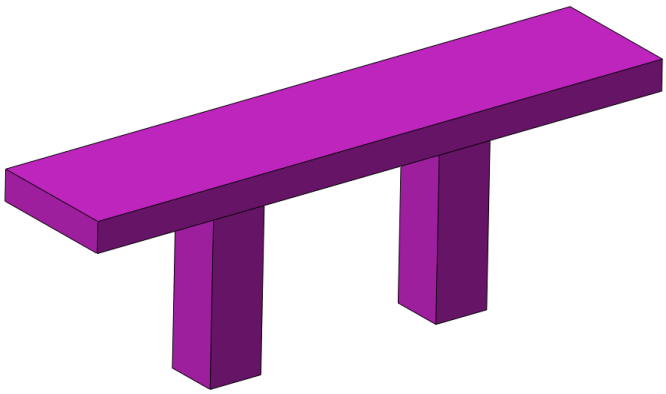


Fig. 3.3: Modelado de los raíles sobre los que se moverán los carros.

3.4 Carro de los servo motores

Sobre los raíles explicados anteriormente se deslizará un carro (figura 3.4) el cual portará el servo motor que accionará de la pinza. Dicho carro será diseñado con una cavidad inferior en forma de T por donde la tabla del raíl pasará evitando los pilares del mismo. Además, esta forma de T restringirá el movimiento en los ejes transversales del raíl, permitiendo un único movimiento en el eje longitudinal.

Por otra parte, la cavidad superior será la que aloje el servo motor en posición vertical, dejando el eje de este lo más alto posible. El servo motor será atornillado mediante tornillos tirafondos para madera sobre la cara frontal, en la superficie libre entre la cavidad en forma de T y la cavidad que alojará el servo motor.

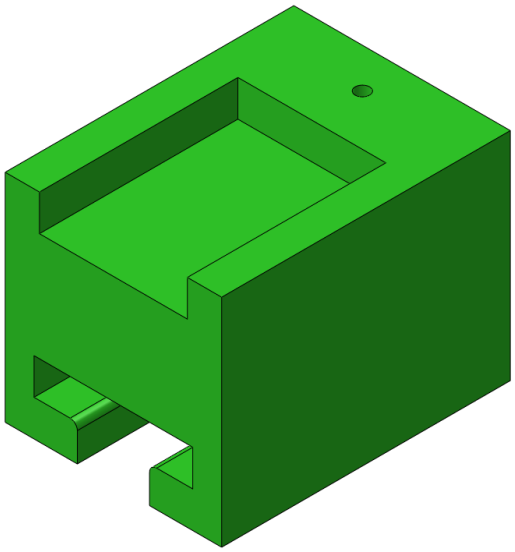


Fig. 3.4: Modelado de los carros que portarán los servo motores.

Por último, el orificio trasero alojará un tornillo con pasante que servirá como eje de rotación al brazo que transmitirá el movimiento del servo motor externo al carro.

3.5 Brazos

Con el objeto de transmitir el movimiento de los servo motores exteriores al carro, se empleará un brazo encargado de transformar el movimiento de rotación de los mismos en un movimiento lineal. Tal y como se observa en la figura 4.8, se trata de una placa plana de 47 x 6 mm con dos agujeros. Uno de los cuales será concéntrico al agujero situado en el carro e irán unidos mediante un tornillo con pasante dado que buscamos que el brazo rote libremente, transmitiendo únicamente el movimiento de translación y no de rotación.

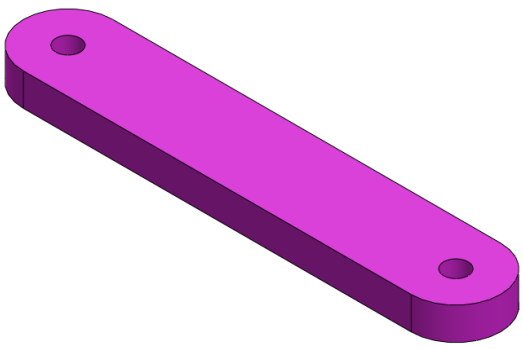


Fig. 3.5: Modelado de los brazos que imparten el movimiento a los carros.

Por otro lado, el agujero contrario se unirá al servomotor mediante una extensión del mismo. En este caso, la unión se realizará con un tornillo M2.

3.6 Pinzas

Para acabar, el elemento encargado de sujetar el cubo de Rubik y hacer rotar sus caras serán cuatro pinzas como la que tenemos en la figura 3.6. Dicha pinza se ha diseñado lo más sencilla posible y sin partes móviles dado que no es necesario que esta, por separado, realice ningún tipo de fuerza, pues al estar las 4 en contacto con el cubo de Rubik, este no tendrá ningún grado de libertad en cuanto a movimientos lineales y solo podrán rotar las caras sobres sus respectivos ejes.

Las pinzas disponen de tres orificios pensados para la unión de la misma al servomotor. El orificio central es donde el eje del servo motor irá situado y unido mediante tornillo. Mientras que los otros dos orificios más pequeños cumplen la función de atornillar la extensión del servo motor a emplear. De esta manera nos aseguramos que la pinza esté siempre centrada con respecto al servo motor.

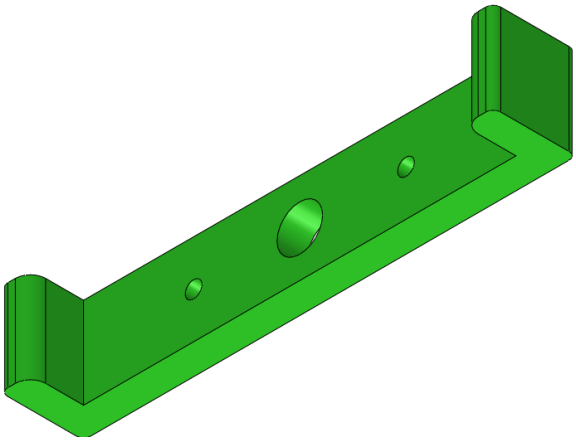
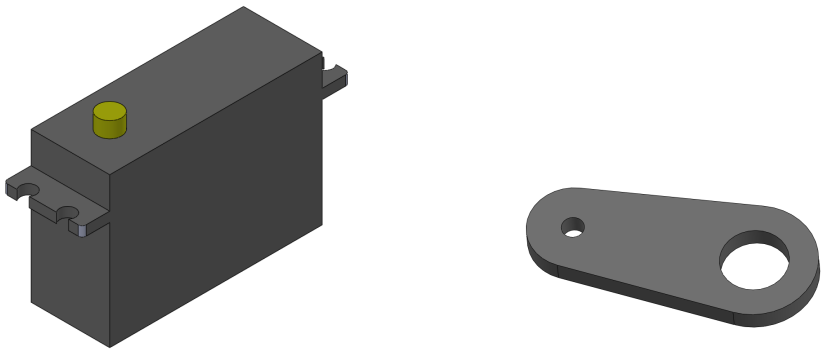


Fig. 3.6: Modelado de las pinzas que amarran el cubo de Rubik.

3.7 Servo motores

Finalmente, con el objeto de comprobar que todo cuadra en el ensamblaje, se ha realizado un diseño muy simple de los servomotores y sus extensiones como podemos observar en las figuras 3.7a y 3.7b respectivamente. Esto nos será útil en un futuro para realizar la simulación ya que será conveniente dotar de un movimiento de rotación al eje del servo motor.



(a) Servo motor (b) Extensión del servo motor

Fig. 3.7: Modelado simplificado del servo motor y sus componentes esenciales para la simulación.

3.8 Ensamblaje del robot

Una vez ya tenemos todos los elementos diseñados, en la figura 3.8 se muestra el ensamblaje completo de como será el sistema diseñado en su conjunto. Dicho ensamblaje será el empleado posteriormente para la simulación del mismo y comprobación de su viabilidad. Cabe destacar que para la simulación se realizarán pequeñas modificaciones para simplificar el modelo y así poder realizar una simulación mas rápida y que consuma menos recursos.

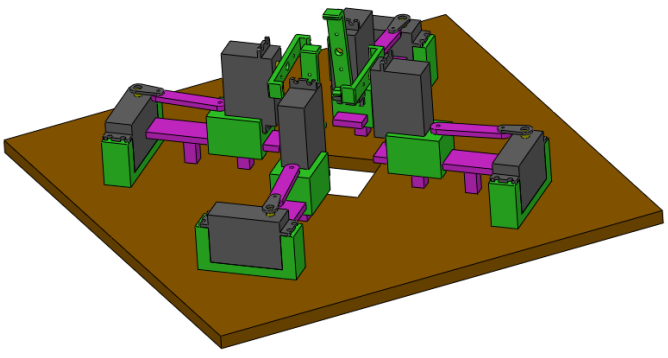


Fig. 3.8: Ensamblado completo del robot.

4 Simulación

Una vez realizado el diseño del sistema, se procede a la simulación del movimiento y comportamiento del mismo. Para ello se hará uso de la herramienta *Simscape Multibody* la cual esta comprendida dentro del entorno de programación *Matlab* y que nos permitirá realizar el modelado y simulación de sistemas dinámicos.

Simscape Multibody proporciona un entorno de simulación para sistemas mecánicos multicuerpo junto con un entorno de diseño CAD limitado. Sin embargo, esta herramienta proporciona unos bloques tipo 'sólido' capaces de importar archivos CAD tales como *STEP* o *IGS*.

4.1 Generación del modelo

Haciendo uso de los bloques tipo 'sólido', se han introducido los modelos generados por CAD en el apartado anterior en el entorno *Simscape Multibody*.

Sin embargo, con el objeto de simplificar el sistema ahorrando así coste computacional, se ha rediseñado la base. En concreto, se ha creado un nuevo modelo de esta en la que todas las partes que poseen una unión fija ahora son un único sólido. Así pues, en la figura 4.1 observamos como los servo motores exteriores y los raíles han sido fijados a la base. Esto nos permitirá emplear un solo bloque para la base en vez de emplear nueve bloques con uniones mecánicas no móviles.

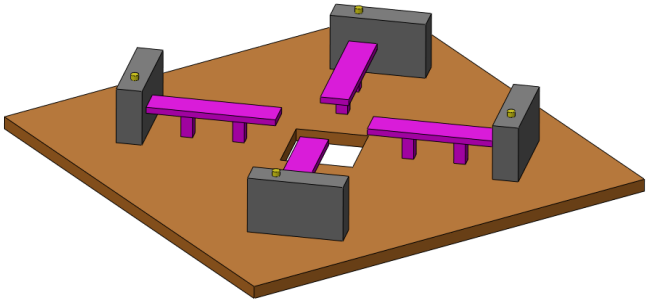


Fig. 4.1: Modelado de la base del robot para la simulación.

Una vez creado el modelo simplificado de la base, se procederá a importar todos los modelos CAD al espacio de trabajo de *Simscape Multibody* y a unirlos todos creando el sistema multicuerpo para su posterior simulación.

Para ello, se hará uso de los siguientes bloques: El bloque 'sólido' (figura 4.6a) el cual nos permitirá importar archivos CAD, modificar el aspecto y generar puntos de referencia sobre los que unir otros bloques del mismo tipo; el bloque 'junta de revolución' (figura 4.6b) mediante el cual se unirán todas aquellas piezas que requieran de un movimiento rotatorio y el bloque 'junta prismática' (figura 4.2c) que tendrá la misma función que el bloque anterior, pero esta vez con piezas que requieran de un movimiento de deslizamiento.

(a) Bloque para la importación de archivos CAD

(b) Bloque de junta de revolución

(c) Bloque de junta prismática

Fig. 4.2: Bloques empleados para la creación del sistema mecánico multicuerpo.

Empezando por la importación de la base rediseñada en la figura 4.1, se empleará el bloque 'sólido' a través del cual se importará el archivo en extensión *STEP* como podemos ver en la figura 4.3. Así mismo, se implementarán los puntos de referencia (*frames*) que serán los que nos permitirá unir las diferentes piezas a la base del robot. Habiéndose creado cuatro *frames* en los ejes de rotación de cada uno de los servo motores y otros cuatro en el centro de cada una de las guías, disponemos de los puntos necesarios para unir los brazos del robot a los ejes de los servo motores, así como los puntos necesarios para la unión de los carros a su guía correspondiente.

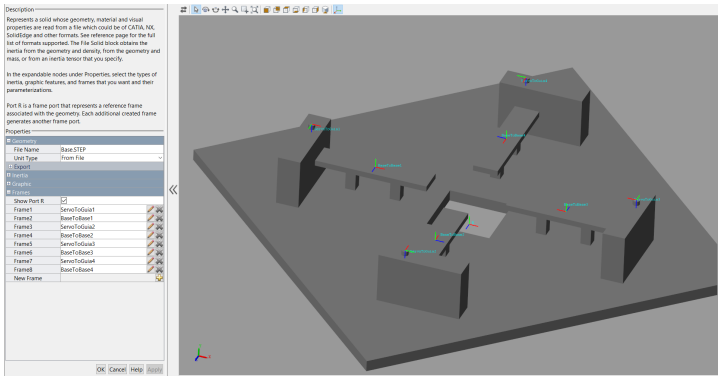


Fig. 4.3: Importación de la base del robot para la simulación.

Seguidamente, se implementarán el resto de piezas. Pero, con el objeto de simplificar nuestro modelo, separaremos el sistema en cuatro 'brazos' iguales los cuales tendrán como entrada el eje de giro del servo motor de la base y se deslizará sobre la guía correspondiente a dicho servo motor. Para ello, se creará un subsistema como el mostrado en la figura 4.4 que albergará todos los componentes del brazo y sus respectivas juntas de revolución y/o prismáticas.

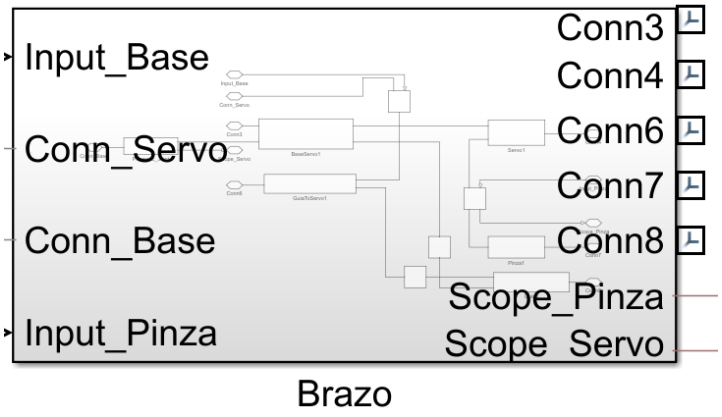


Fig. 4.4: Subsistema de uno de los cuatro brazos del robot.

Una vez creado el subsistema, empezaremos importando el modelo del carro y creando los *frames* correspondientes (figura 4.5). En este caso, disponemos de tres *frames* que nos servirán para unir el carro a la guía ('*BaseToBase1*'), al brazo del servo motor ('*BaseToGuia*') y al segundo servo motor que acciona la pinza ('*Base*'). Cada uno de los *frames* mentados ha sido situado en el centro de la cara que va en contacto con la superficie correspondiente.

Cabe destacar que es de vital importancia situar cada *frame* con la misma orientación que el *frame* al que va unido en la pieza correspon-

diente. De no ser así, ambas piezas no se alinearán correctamente y producirá un error en la simulación.

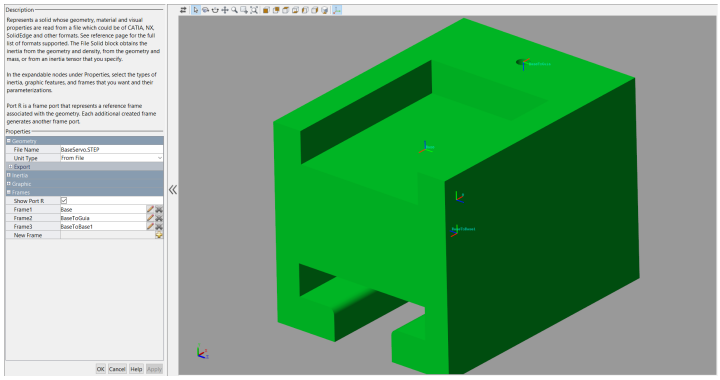
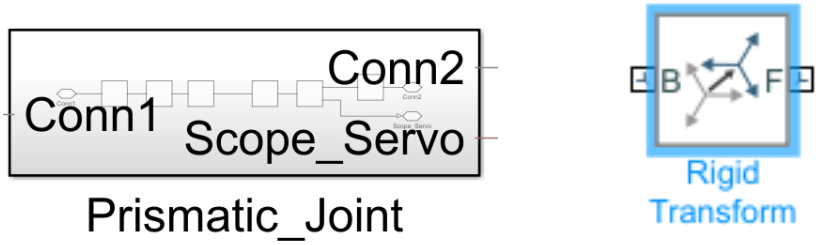


Fig. 4.5: Importación del modelo del carro para la simulación.

Seguidamente, se unirá el carro a la guía mediante un par prismático. Sin embargo, dicho par solo actúa sobre el eje Z por restricciones de la herramienta empleada. Esto supone un problema dado que los *frames* que unen dichas piezas están orientados con el eje Y como eje longitudinal. Por ello, será conveniente crear un subsistema empleando transformaciones rígidas para cambiar el eje de movimiento como muestran las figuras 4.6a y 4.6b respectivamente.



(a) Subsistema de la junta prismática

(b) Bloque de transformada rígida

Fig. 4.6: Bloques empleados para la creación del cambio de eje en el par prismático.

Detallando el subsistema de la junta prismática y empezando de izquierda a derecha, el la figura 4.7 vemos como el primer bloque de transformación rígida rotará el *frame* 90° sobre el eje X. Situando así el eje Z como eje de cota. Seguidamente se introduce un bloque de junta prismática para crear un grado de libertad en el eje Z del

sistema, es decir, que el carro tiene libertad de movimiento en la misma dirección que actúa la gravedad. Una segunda transformada rígida volverá a rotar el *frame* sobre el eje X, pero esta vez en sentido contrario ($-90°$) y una tercera transformada rígida rotará el *frame* $90°$ sobre el eje Y posicionando así el eje Z como eje transversal. Sobre dicho eje se situará un par prismático sensorizado el cual nos permitirá observar la posición y velocidad del carro en cada instante de la simulación. Finalmente, se empleará una última transformada rígida que, análogamente a la segunda transformada rígida, volverá a situar el *frame* en su posición inicial.

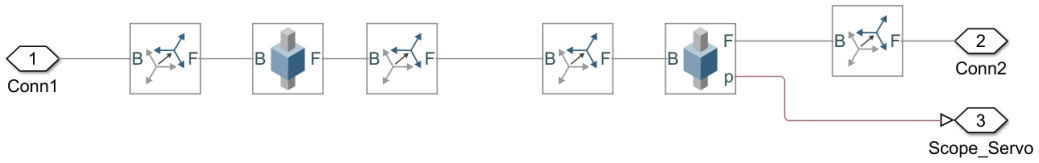


Fig. 4.7: Vista detallada del subsistema empleado como junta prismática.

Una vez unido el carro a la guía, se procederá a unir el brazo que transformará el movimiento circular del servo motor a movimiento de translación. Para ello, análogamente a la importación del modelo del carro, se importará el modelo CAD del brazo como muestra la figura 4.8. En este modelo se emplearán dos *frames* situados cada uno en el centro de cada orificio para, así, poder conectar esta pieza mediante juntas de revolución.

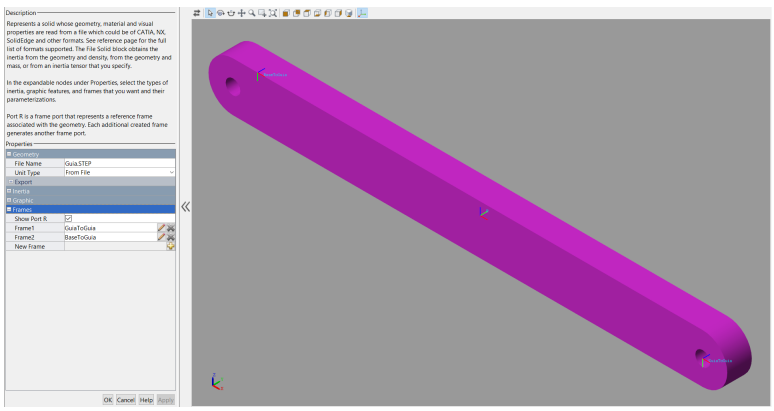


Fig. 4.8: Importación del modelo del brazo para la simulación.

Siendo una de las juntas de revolución la que conectará dicho brazo con el carro, mientras que la otra junta de revolución conectará el brazo con la extensión del servo motor. Por lo que el modelo de la extensión del servo motor será importado creando los *frames* en el centro de cada orificio tal como se ha realizado en la pieza anterior y muestra la figura 4.9. En este caso, el orificio diminuto será el que se unirá con el brazo mostrado en la figura 4.8 y el orificio grande irá fijo con el eje de rotación del servo motor.

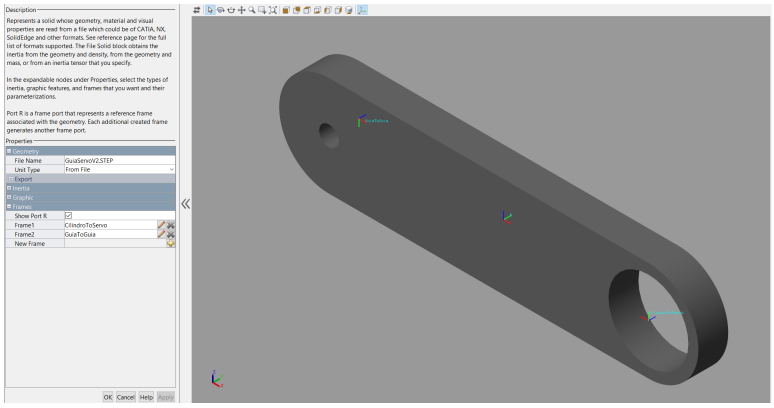


Fig. 4.9: Importación del modelo de la extensión del servo motor para la simulación.

Con tal de simplificar la simulación, será esta última junta de revolución, que une todo lo que es el brazo del robot con la base del mismo, la encargada de recibir la entrada de movimiento giratorio. Cabe destacar que la forma de generar esta entrada será explicada de una manera más extensa en el subapartado posterior.

Volviendo al carro, procedemos a colocar el servo motor sobre este. Para ello importaremos el modelo CAD del servo motor como nos muestra la figura 4.10 creando los *frames* necesarios para la unión del modelo con el resto del brazo del robot. En este caso se emplearán dos *frames*, uno en el centro de la cara circular del eje del servo motor y el otro en el centro de la cara plana más alejada de este eje. Siendo este último *frame* el que irá unido mediante una unión fija a la cara superior del carro.

Con respecto al primer *frame* creado en el servo motor, este servirá para unir la pinza encargada de la rotación de las caras del cubo de

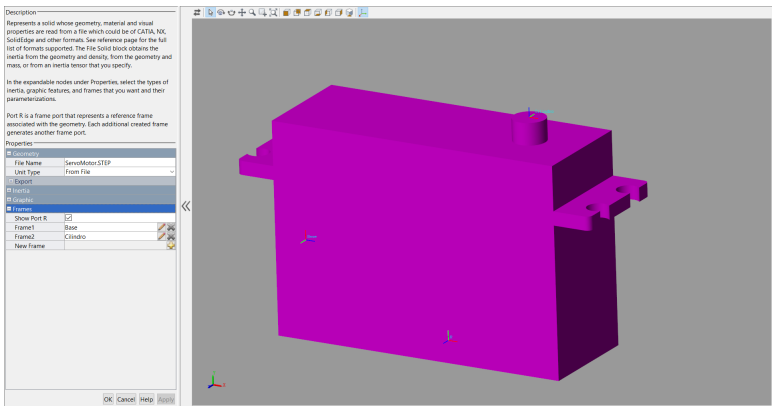


Fig. 4.10: Importación del modelo del servo motor para la simulación.

Rubik mediante una junta de revolución. Para ello, empezaremos importando el modelo 3D de la pinza y creando el *frame* correspondiente (figura 4.11). En este caso, el *frame* irá colocado en el centro de la cara del orificio ya que es este punto el que se unirá con el servo motor.

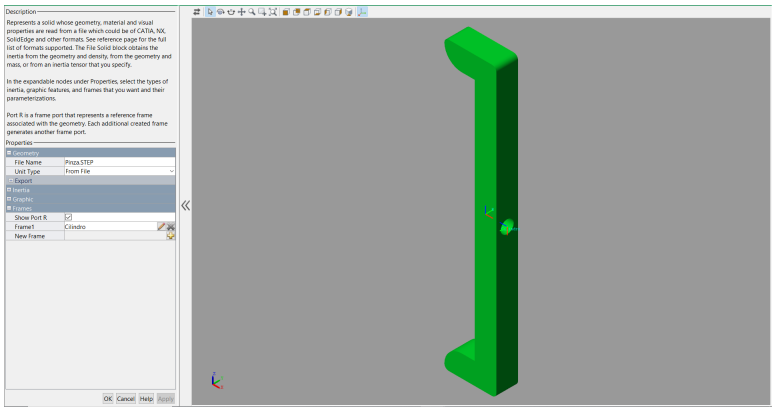


Fig. 4.11: Importación del modelo de la pinza para la simulación.

Dado que el eje del servo motor es una entrada de movimiento, la junta de revolución será la encargada de recibir dicha entrada a través de un *input*. Por otro lado, esta junta de revolución también irá sensorizada con el objeto de observar su posición y velocidad a lo largo de la simulación.

Con todos los modelos CAD importados y unidos correctamente, se creará un subsistema que albergue todo el sistema que es el brazo robot. Esto nos ayudará a crear los otros tres brazos dado que son todos iguales y así no será necesario repetir el proceso de importación y creación de *frames* de cada modelo CAD. la vista detallada de dicho subsistema se muestra en la figura 4.12 donde observamos

las juntas prismáticas y de revolución que unen cada componente. Por otro lado, es importante remarcar que los puntos *1* y *2* son las entradas de movimiento del sistema, mientras que los puntos *10* y *11* son las salidas de los sensores de las juntas que nos permitirán observar el comportamiento de las mismas en un bloque *scope*.

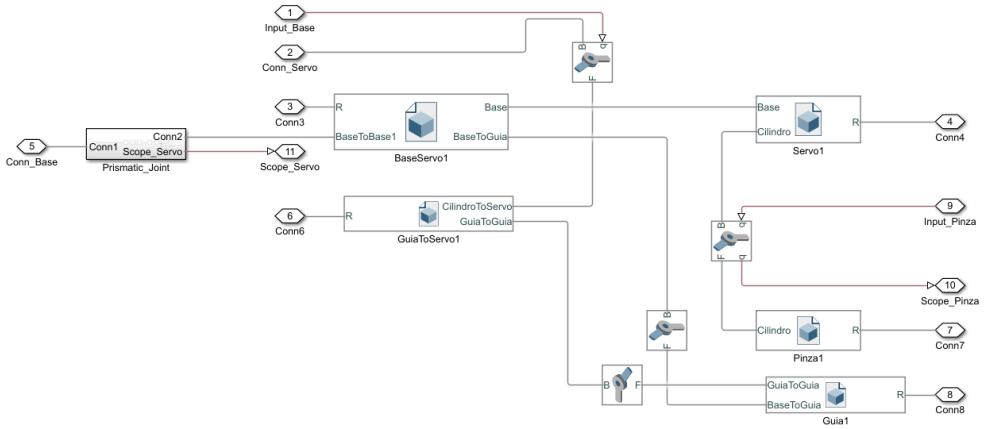
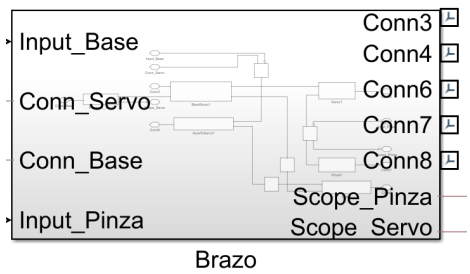
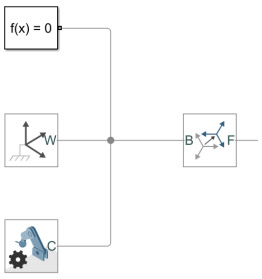


Fig. 4.12: Vista detallada del subsistema empleado como brazo del robot.

Con el subsistema del brazo del robot creado (figura 4.13a), se procede a la creación del sistema completo uniendo los cuatro brazos con la base previamente importada. Por otro lado, la base se unirá a un conjunto de bloques al que llamaremos 'mundo' ya que son estos bloques los encargados de situar las condiciones iniciales tales como la gravedad (dirección, sentido y módulo). Este conjunto de bloques viene mostrado en la figura 4.13b.



(a) Subsistema del brazo del robot.



(b) Conjunto de bloques que forman el mundo.

Fig. 4.13: Bloques empleados para la creación del sistema completo.

Finalmente, la figura 4.14 muestra el sistema ya ensamblado. Teniendo la base unida por un extremo al mundo y por el otro a cada uno de los brazos. Cabe destacar como cada brazo posee sus dos entradas simulando el movimiento de cada servo motor y otras dos salidas con el fin de observar el comportamiento de los mismos. Las salidas han sido separadas en dos *scopes* diferentes con el objeto de separar los movimientos de giro de las pinzas y los desplazamientos de los carros.

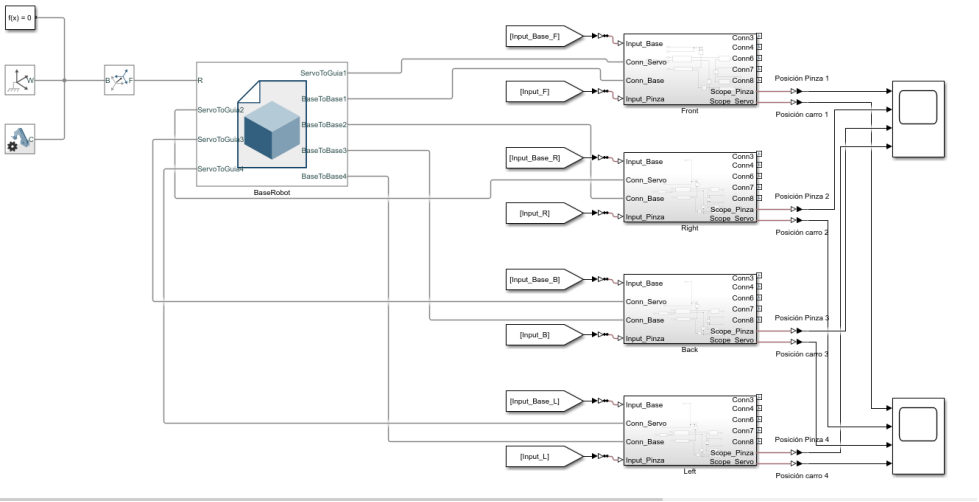


Fig. 4.14: Vista detallada del sistema que conforma el robot que resuelve el cubo de Rubik.

4.2 Generación de las referencias

El siguiente paso consistirá en la generación de una secuencia definida por nosotros emulando la secuencia proporcionada por el algoritmo que resuelve el cubo de Rubik.

Para ello, se empleará un generador de señales o *signal builder* como el mostrado en la figura 4.15. Este bloque nos permitirá generar una secuencia de señales con tal de emular todos los pasos que el robot debería seguir para la resolución de un caso concreto del cubo de Rubik. Concretamente, se generarán ocho señales, una por cada servo motor del que disponemos, de las cuales cuatro dotarán de movimiento a las pinzas y las otras cuatro acercarán o alejarán las pinzas según sea necesario.

Cabe destacar que el *signal builder* genera señales temporales con

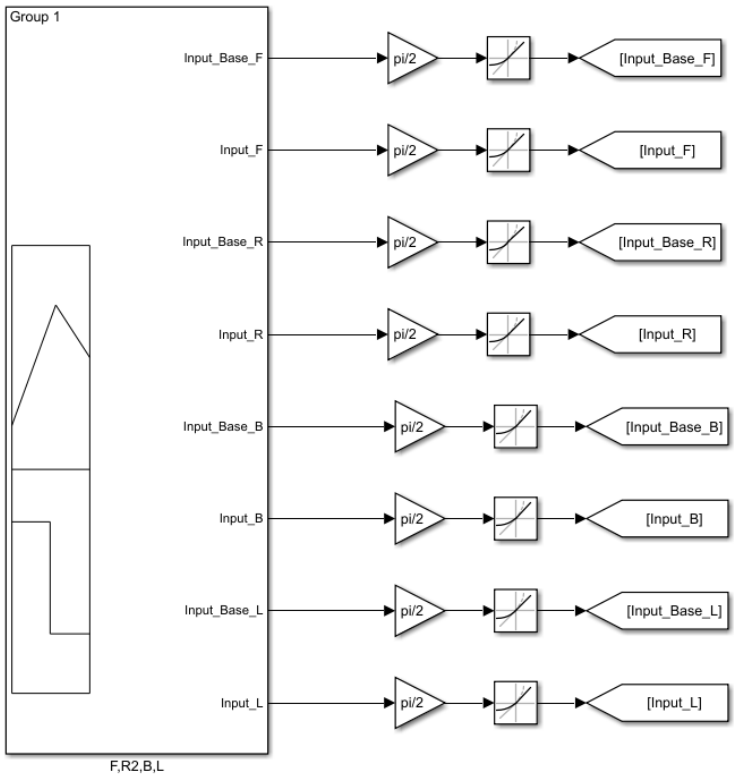


Fig. 4.15: Generador de señales para la generación de referencia.

una ganancia predefinida que nosotros hemos optado como '1' ya que nos facilitará a la hora de trabajar con ellas. Sin embargo, la señales generadas son señales lineales y los servo motores generan señales radiales. Por ello, será conveniente emplear el bloque de ganancia (figura 4.16) para transformar la señal en radianes multiplicándola por $\frac{\pi}{2}$. Seguidamente, dado que el comportamiento del servo motor no es instantáneo mientras que la señal generada si que realiza escalones instantáneos, emplearemos un bloque *rate limiter* (figura 4.16) con el objeto de transformar dicho escalón en una rampa de subida lo más parecida al comportamiento de nuestro servo motor. En nuestro caso, según las especificaciones técnicas de nuestro servo motor, tarda 0,16 segundos en llegar a la posición final. Por lo que se buscará el comportamiento más similar posible mediante este últimoi bloque.

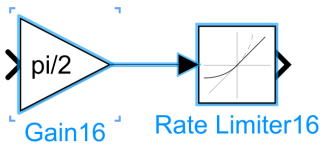


Fig. 4.16: Bloques de ganancia para adaptar la referencia a los servo motores.

Una vez creados los bloques pertinentes para la generación de la

referencia, se creará un conjunto de señales emulando dos casos concretos para la resolución del cubo de Rubik.

Empezando por un caso sencillo, se emulará la secuencia generada por el algoritmo '$F, R2, B, L$' lo cual quiere decir que la pinza de la cara frontal girará 90°, la de la cara derecha girará 180°, la de la cara trasera girará 90° y la de la cara izquierda girará 90°. Para ello, es conveniente que entre cada giro de la pinza el carro se desplace hacia atrás y la pinza vuelva a su posición inicial para que las pinzas no colapsen entre si. Para ello, la figura 4.17 muestra todos los pasos a seguir, siendo las líneas moradas las referencias de las pinzas mientras que las rojas son las referencias de los carros.

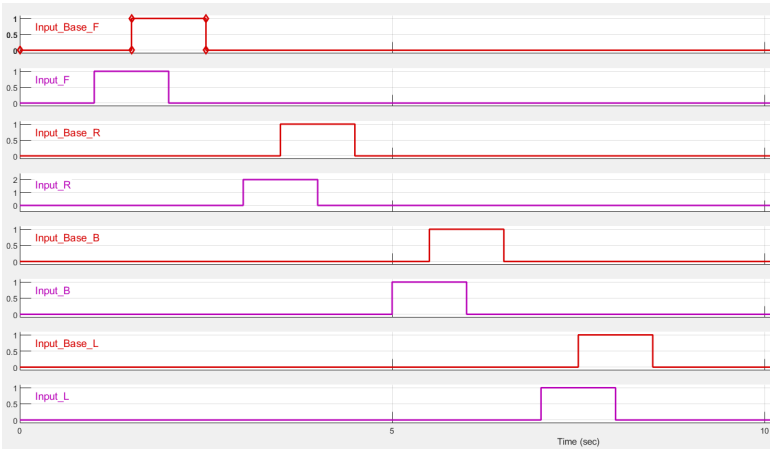


Fig. 4.17: Generación de referencia primer caso.

Seguidamente, dado que solo disponemos de cuatro pinzas y el cubo tiene seis caras, será necesario girar el cubo con el objeto de dejar las caras superior e inferior accesibles. Por ello, dos de las pinzas, en este caso la derecha y la izquierda, rotarán al unísono mientras que las otras dos pinzas estarán desacopladas. Permitiendo así la libertad de movimiento del cubo de Rubik.

Teniendo esto en cuenta, se creará una segunda referencia en la que se realizará el giro de la cara inferior, como muestra la figura 4.18, con el fin de comprobar el correcto funcionamiento del sistema. Para lograr el giro de la cara inferior se realizará una pequeña secuencia en la que las pinzas frontal y trasera se desacoplarán mediante la translación de sus respectivos carros alejándose del cubo de Rubik. Seguidamente, las pinzas laterales rotarán al unísono, pero teniendo

en cuenta que la pinza derecha (R) rotará 90° mientras que la pinza izquierda (L) rotará −90° para así lograr un giro del cubo.

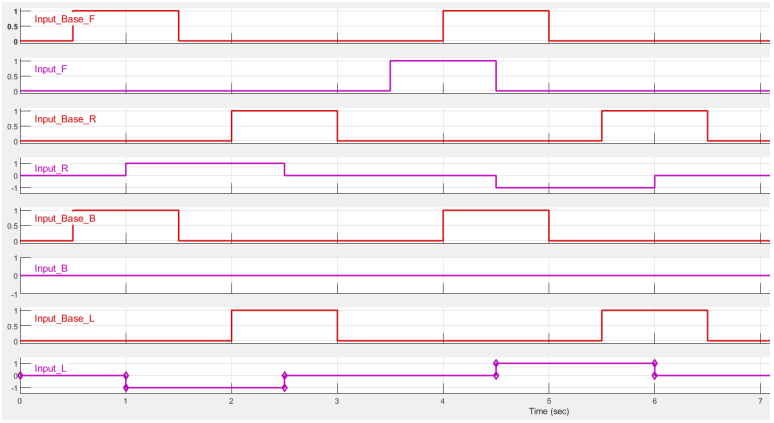


Fig. 4.18: Generación de referencia segundo caso.

Una vez el cubo girado, se acoplarán las pinzas frontal y trasera. Acopladas todas las pinzas, se procede a desacoplar las dos pinzas laterales para volverlas a situar en su posición inicial. Seguidamente se acoplarán estas dos pinzas y la pinza trasera, que ahora accede a la cara inferior, rotará los grados pertinentes. Tras completar la rotación, se desacoplarán las pinzas frontal y trasera con el fin de volver a situar el cubo en su posición inicial mientras que la pinza trasera, ya desacoplada, también es situada en su posición inicial.

Con esto, ya dispondríamos de dos buenos casos como referencia para la simulación del sistema y la comprobación de su correcto funcionamiento.

4.3 Resultados de la simulación

Para la correcta interpretación de los resultados se dividirá en dos casos. Siendo el primero la emulación de la secuencia generada por el algoritmo y el segundo, el giro de la cara inferior.

Empezando por el primer caso, las gráficas 4.19 y 4.20 muestran el seguimiento de la referencia en posición de los cuatro carros, ordenados de forma que el primer carro será el situado en la cara frontal, el segundo el situado el la cara derecha, el tercero el de la cara trasera

y el cuarto el de la cara izquierda.

Como podemos observar en ambas gráficas, el seguimiento de la referencia es más que bueno. Sin embargo, hay un error permanente en los dos últimos carros. Este error permanente es despreciable pues no es ni siquiera de un milímetro y físicamente no afectará al comportamiento real del sistema. Cabe destacar que este error se puede dar debido a los diferentes elementos situados entre el eje del servo motor y el carro que, según la precisión de como se hayan generado los *frames*, puede estar afectando a los ejes de rotación creando este error permanente casi despreciable.

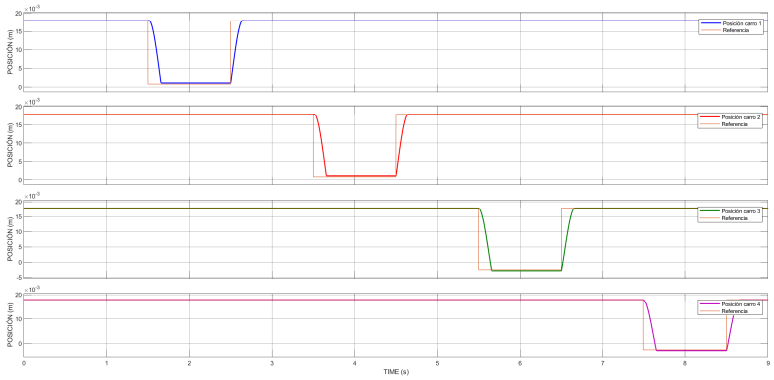


Fig. 4.19: Seguimiento de la referencia de los carros.

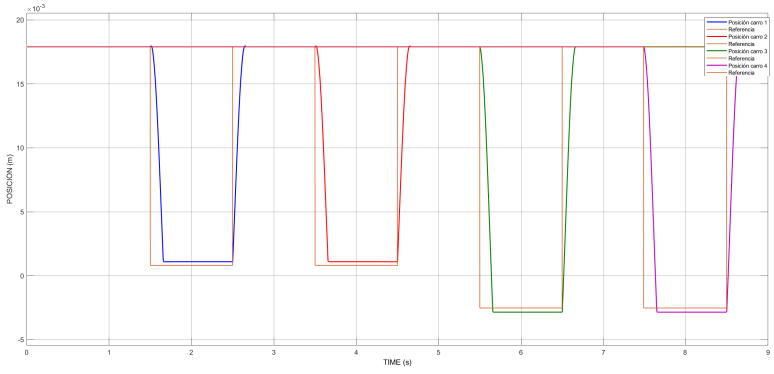


Fig. 4.20: Seguimiento de la referencia de los carros.

Por otro lado, las gráficas 4.21 y 4.22 muestran el seguimiento de la referencia en posición de las pinzas ordenadas de igual manera que los carros.

En este caso, observamos un seguimiento de la referencia casi perfecto. Sin embargo, el movimiento de la pinza a la hora de rotar 180° es un poco lento.

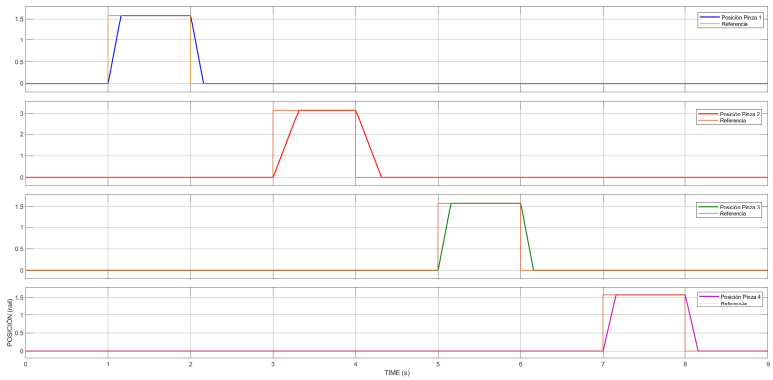


Fig. 4.21: Seguimiento de la referencia de las pinzas.

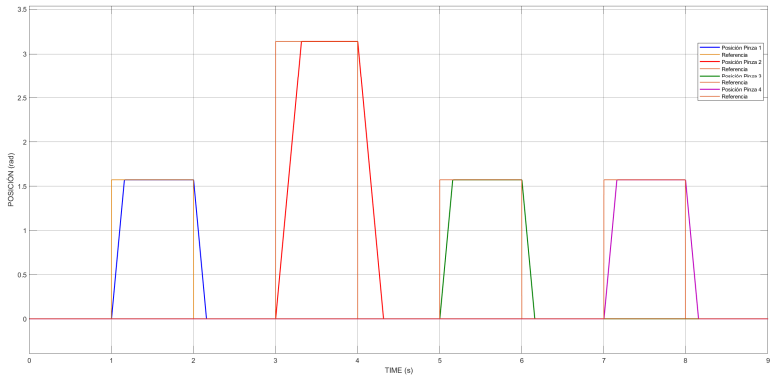


Fig. 4.22: Seguimiento de la referencia de las pinzas.

Finalmente, para el segundo caso obtenemos las gráficas 4.23 y 4.24 las cuales muestran el seguimiento de la referencia en posición de los cuatro carros. En esta ocasión, al igual que en el caso anterior, se observa un error permanente despreciable ya que es inferior a un milímetro.

Por último, las gráficas 4.25 y 4.26 las cuales muestran el seguimiento de la referencia en posición de las cuatro pinzas. Obteniendo un error permanente nulo y un seguimiento de la referencia bastante rápido.

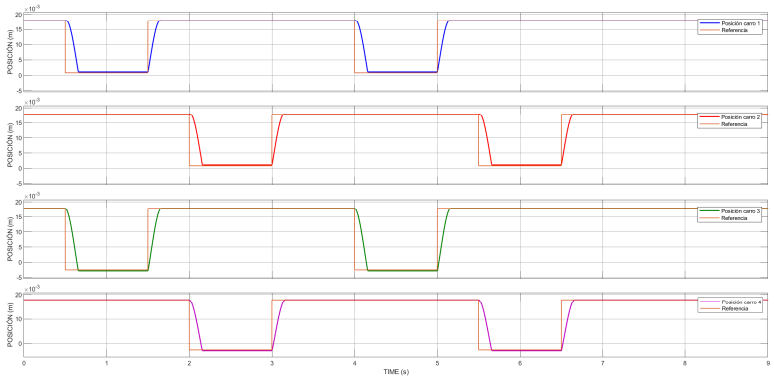


Fig. 4.23: Seguimiento de la referencia de los carros.

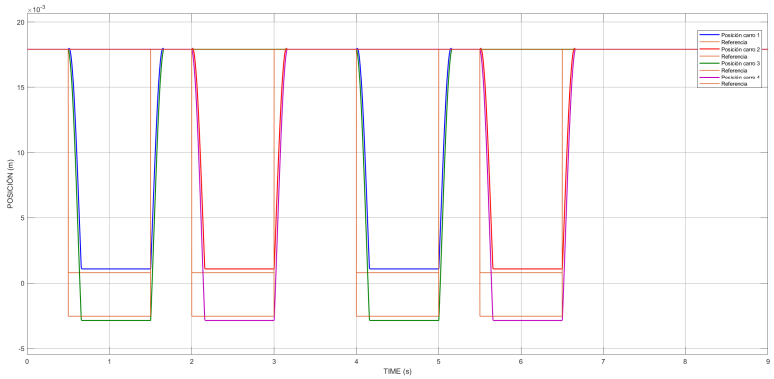


Fig. 4.24: Seguimiento de la referencia de los carros.

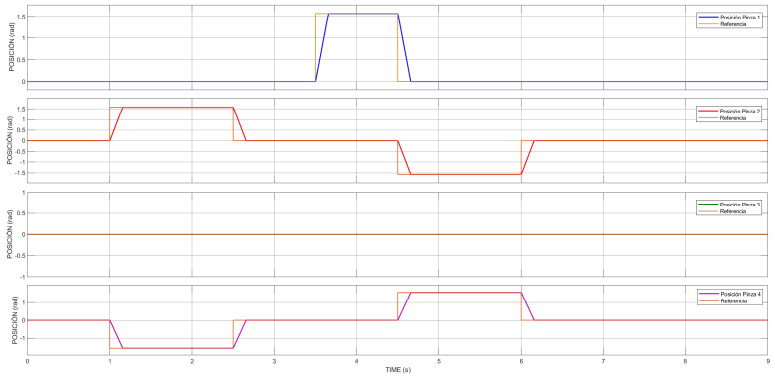


Fig. 4.25: Seguimiento de la referencia de las pinzas.

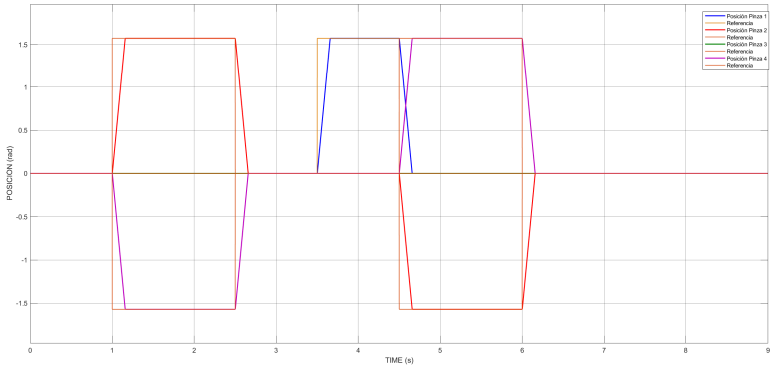


Fig. 4.26: Seguimiento de la referencia de las pinzas.

5 Implementación

Una vez comprobada la efectividad del sistema mediante la simulación, se procederá al montaje del robot. Para ello, se han empleado las piezas diseñadas anteriormente así como elementos de unión normalizados y componentes electrónicos.

Cabe destacar que ciertas piezas han sufrido un rediseño dado que el sistema de fabricación no admitía ciertas características. Por ello, se empezará detallando el rediseño de dichas piezas para luego continuar con una explicación detallada del montaje y su implementación.

5.1 Rediseño de piezas

El rediseño del carro y de la base de los servo motores exteriores ha sido necesario debido al método de fabricación empleado. Dicho método consiste en sinterizado láser en Nylon, el cual no admite espesores de pared demasiado grandes dado que cuanto mayor el espesor de pared, más velocidad de movimiento requiere el láser fundidor de polvo de Nylon. Generando así deformaciones como estrechamiento de la pieza o paredes no rectas.

Debido a este suceso, el carro, como podemos ver en la figura 5.1a y 5.1b, ha sufrido un vaciado dejando paredes de 3 mm de espesor. Debido al vaciado, ha sido necesario generar un pilar alrededor del orificio superior con el objeto de evitar el colapse en caso de emplear demasiada fuerza para atornillar el brazo que se une a ese punto.

(a) Carro (b) Carro 2

Fig. 5.1: Modelado modificado del carro para la adaptación de su fabricación.

El siguiente elemento que ha sido modificado es la base donde se apoyarán los servo motores exteriores. En este caso, como vemos en la figura 5.2, se ha vaciado tanto la base como las paredes laterales. Dejando unas pequeñas columnas alrededor de los orificios con el objeto de emplear tornillos tirafondos para unir los servo motores a dicho elemento.

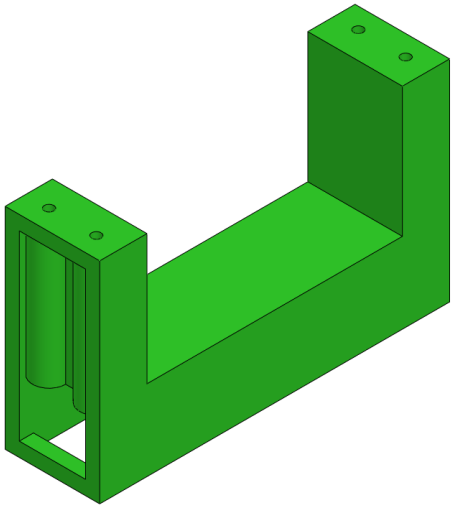


Fig. 5.2: Modelado modificado de la base de los servo motores para la adaptación de su fabricación.

5.2 Instrucciones de montaje

Seguidamente, se mostrará paso por paso el ensamblaje del sistema completo. Dado que el sistema se compone de cuatro brazos idénticos, solo se mostrará el ensamblaje de uno de ellos con el fin de simplificar la explicación teniendo unas imágenes más limpias y con menos elementos.

Empezando desde el exterior, la base del servo motor se unirá a la base del sistema fijándola haciendo uso de cianocrilato como muestra la figura 5.3.

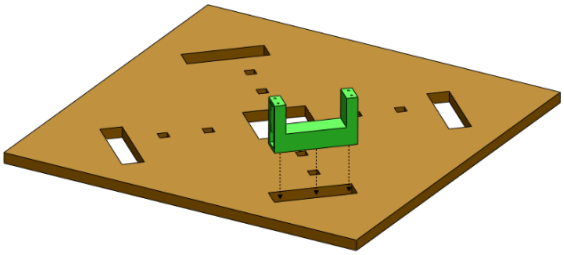


Fig. 5.3: Primer paso del montaje del prototipo.

Una vez unida la base de los servo motores a la base del sistema, se unirá el raíl a la base del sistema tal como muestra la figura 5.4 empleando el mismo método que en el paso anterior.

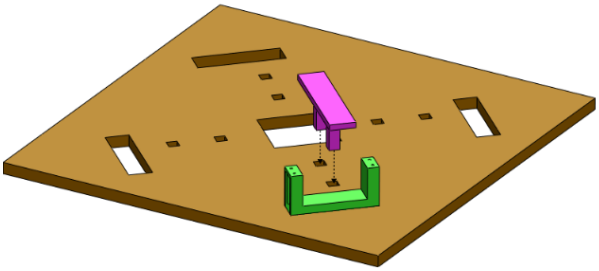


Fig. 5.4: Segundo paso del montaje del prototipo.

Con la base se los servo motores fija, se atornillará el servo motor a su base haciendo uso de cuatro tornillos de métrica 3 con $10mm$ de longitud. Por otro lado, se unirá la extensión del servo motor y se fijará mediante un tornillo de métrica 4 con $5mm$ de longitud tal y como se muestra en la figura 5.5.

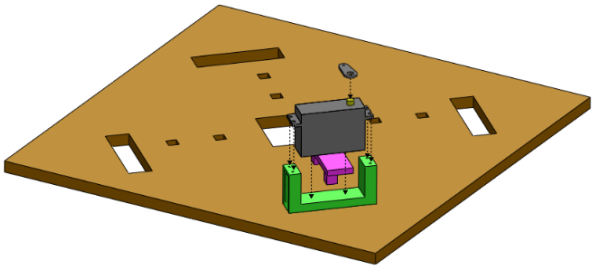


Fig. 5.5: Tercer paso del montaje del prototipo.

Seguidamente se introducirá el carro sobre el raíl deslizándolo desde el extremo interior del mismo como se muestra en la figura 5.6. En este paso, es importante comprobar que el carro desliza sobre el raíl para obtener un buen comportamiento del sistema. En caso contrario, será necesario lijar las caras del raíl pertinentes.

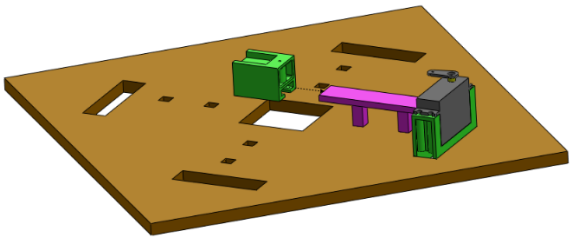


Fig. 5.6: Cuarto paso del montaje del prototipo.

Una vez comprobado que el carro desliza sobre el raíl, se colocará el brazo sobre este uniéndolo, tanto al carro como a la extensión del servo motor, empleando tornillos de métrica 2 con $10mm$ y $5mm$ de longitud respectivamente como muestra la figura 5.7.

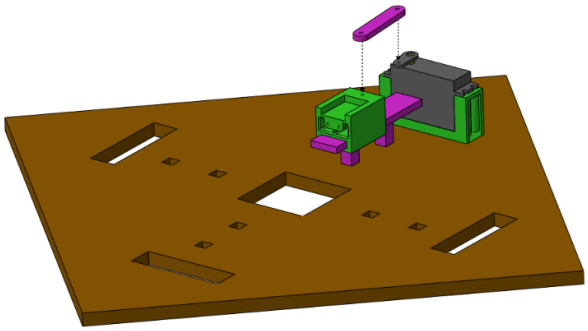


Fig. 5.7: Quinto paso del montaje del prototipo.

Análogamente al tercer paso, colocaremos el servo motor sobre el carro uniéndolo mediante tornillos de métrica 3 con $10mm$ de longitud tal y como vemos en la figura 5.8 situando el eje del mismo en la parte superior. Cabe destacar que el servo motor no debe tener ningún tipo de holgura en esta posición. Por lo que en caso que la tuviese, sería necesario emplear un elemento fijador como puede ser el cianocrilato.

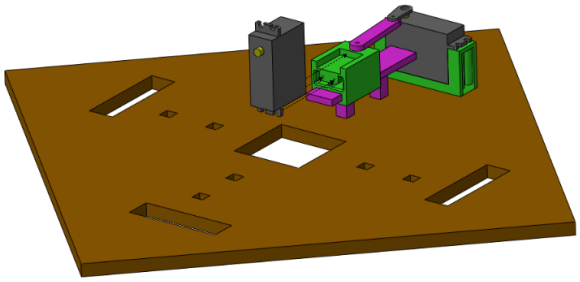


Fig. 5.8: Sexto paso del montaje del prototipo.

Por último, la pinza se unirá al eje ser servo motor como muestra la figura 5.9. Es importante que la pinza este completamente vertical en la posición 0° del servo motor. Por lo que será conveniente enviar el servo motor a dicha posición antes de colocar la pinza.

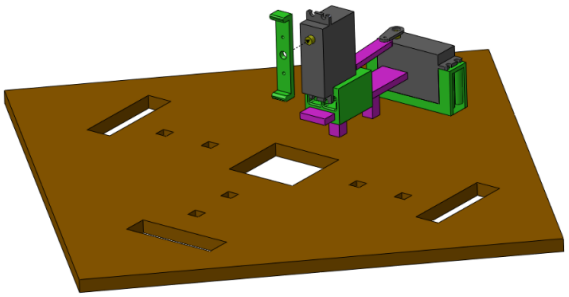


Fig. 5.9: Séptimo paso del montaje del prototipo.

5.3 Conexiones del sistema

Una vez ya tenemos montado el sistema, se procede a la conexión y explicación de todos los componentes electrónicos para el correcto funcionamiento del mismo.

Empezando por los servo motores, estos disponen de tres cables diferenciados por tener distinto color. Por un lado tenemos el cable marrón el cual se conectará al neutro o *ground* (GND), luego, tenemos el cable rojo mediante el cual alimentaremos el servo motor (V+ o VCC) y, por último, tenemos el cable amarillo a través del cual se enviarán las señales PWM.

Cabe destacar que el microcontrolador solo es capaz de alimentar

a dos servo motores y nuestro prototipo dispone de ocho. Por lo que será necesario el uso de un driver a través del cual alimentar y controlar los servo motores. Para ello, se ha elegido el driver PCA9685 con el objeto de conectar una batería externa de cuatro pilas AA de 1.5V cada una.

El driver PCA9685 dispone de seis pines para la conexión a microcontrolador, sin embargo, ha sido conectado al Arduino mediante cuatro de estos seis pines. Uno de ellos irá conectado al neutro o *ground* (GND), otro alimentará el driver a través del pin de alimentación del Arduino (V+ o VCC) y los otros dos pines establecerán la comunicación haciendo uso del protocolo de comunicación I2C.

Por último, el Arduino será alimentado a través del puerto USB B dado que este necesita conexión directa con el ordenador porque será controlado mediante el entorno de programación *Matlab*.

El esquema de las conexiones explicadas se muestra en la figura 5.10 de manera gráfica y concisa.

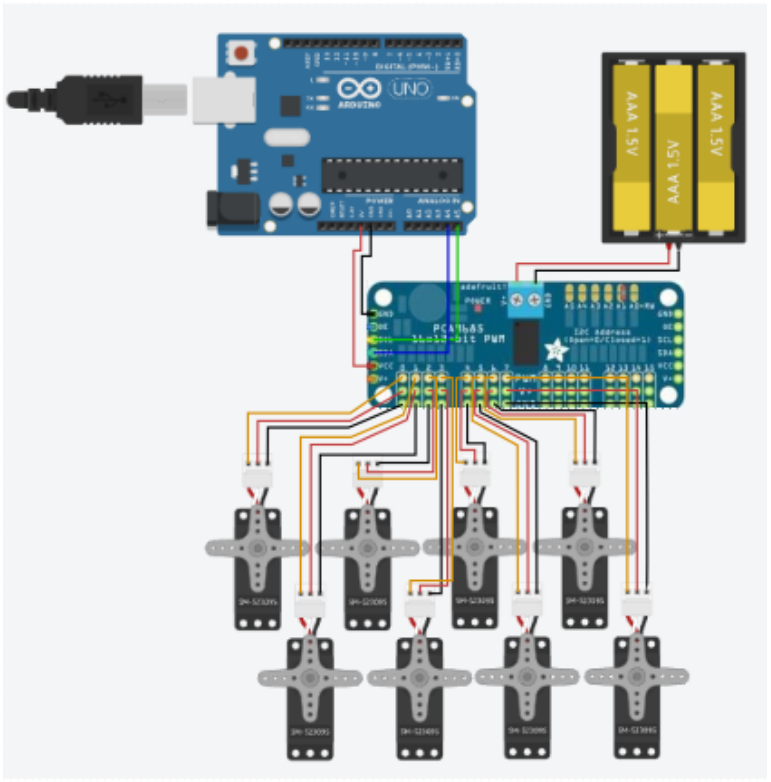


Fig. 5.10: Esquema de conexiones del sistema.

Por otro lado, la tabla 5.1 mostrará, de manera resumida, los pines empleado del Arduino UNO.

Pin Arduino	Pin del Componente	Componente
5V	Vcc	PCA9685
GND	GND	
A4	SDA	
A5	CSL	
USB B	Puerto USB del pc	IDE Arduino

Tabla 5.1: Conexiones del sistema según los pines empleados.

Por otro lado, los servo motores han sido conectados al driver PCA9685 empleando los pines del 0 al 7 siguiendo elcódigo de colores tanto del driver como de los servo motores (GND con GND, VCC con VCC y PWM con PWM).

Una vez el montaje realizado y el sistema conectado, el prototipo queda tal y como se muestra en la figura 5.11.

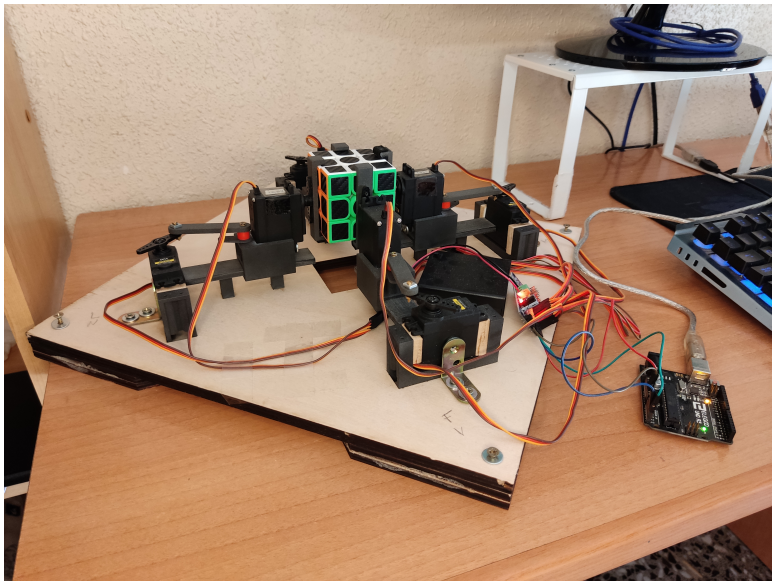


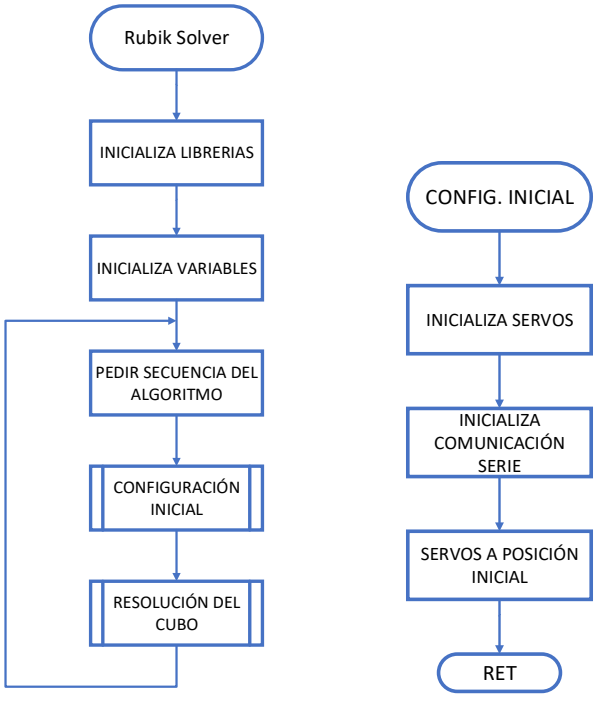
Fig. 5.11: Imagen completa de la implementación del prototipo.

5.4 Programación del microcontrolador

A continuación se procede a explicar detalladamente la programación del sistema la cual ha sido realizada tanto en el entorno de programación *Arduino IDE* como en el entorno de programación *Matlab*.

En un principio el programa iba a ser desarrollado completamente en *Matlab*, sin embargo, no se encontraron extensiones de *Matlab* que soportasen el driver PCA9685. Por ello, se ha optado por realizar los cálculos y operaciones necesarias en *Matlab*, enviar mediante comunicación serie lo que los servo motores han de realizar y que el programa en *Arduino IDE* sea el encargado de mover exclusivamente los servo motores a demanda.

Empezando por el cuerpo principal del programa de la parte realizada en el entorno *Matlab*, en la figura 5.12a se muestra el diagrama de flujo del mismo. Dicho programa comienza inicializando la librerías y variables necesarias para su ejecución. Seguidamente pide la secuencia generada por el algoritmo y realiza la configuración inicial para posteriormente pasar a la resolución del Rubo de Rubik y volver a pedir la nueva secuencia a realizar quedándose a la espera.



(a) Cuerpo principal en *Matlab*

(b) Configuración inicial

Fig. 5.12: Diagramas de flujo.

Pasando a la función de configuración inicial (Figura 5.12b), esta inicializará los servo motores así como la comunicación serie necesaria para comunicar *Matlab* con *Arduino IDE*. Por otro lado, ordenará a los servo motores a ponerse en posición inicial.

Seguidamente tenemos la función principal: Resolución del Cubo (Figura 5.13). Esta función empieza leyendo la secuencia separándola en caras y grados girados. Con estos datos separados, se leerá posición por posición ambos vectores y se enviará el mensaje por el puerto serie para realizar el giro de la cara pertinente con sus respectivos grados de giro. Dando un error en caso de no leer una de las caras.

Una vez enviado el mensaje, se esperará confirmación por parte del microcontrolador con el fin de no ordenar girar una cara mientras otra esta a mitad operación, evitando así el colapse del sistema.

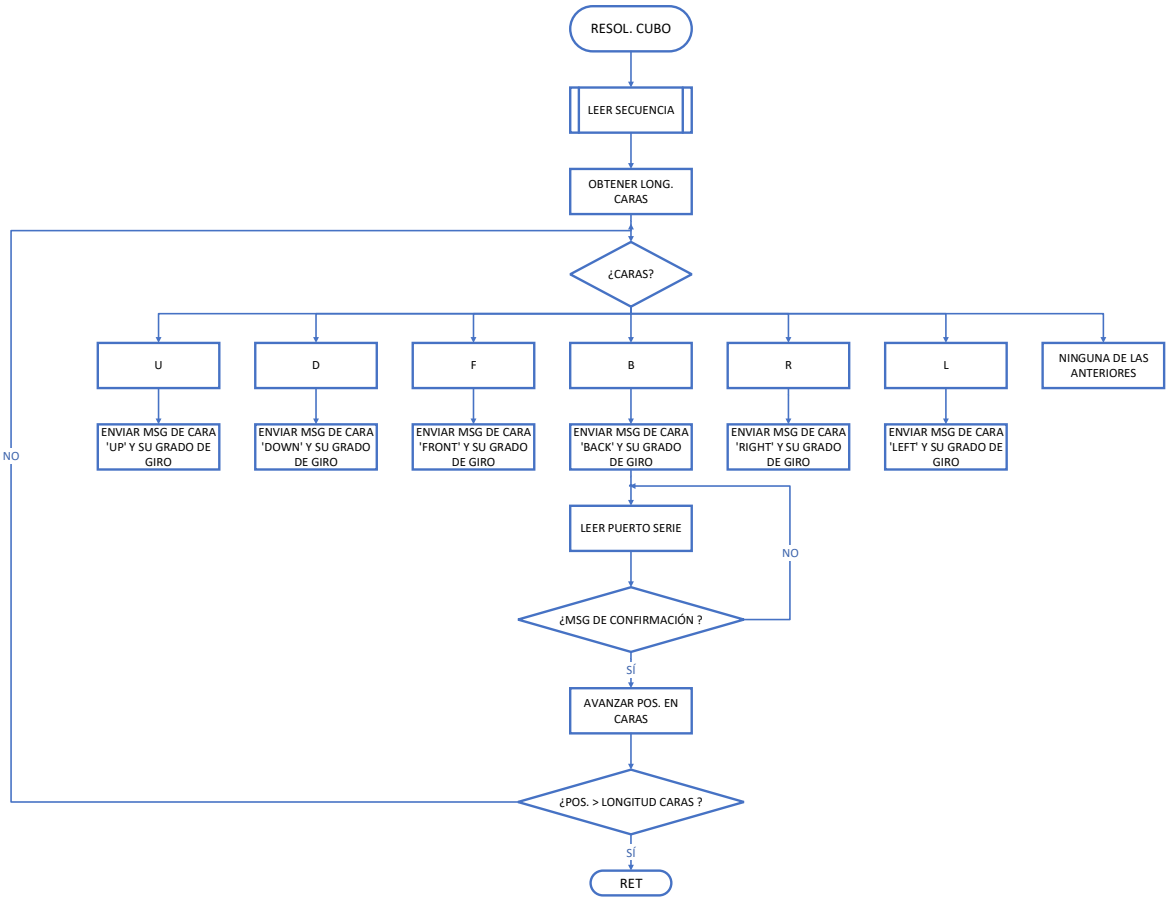


Fig. 5.13: Diagrama de flujo de la función Resolución del Cubo.

La función Leer Secuencia (Figura 5.14) será la encargada de leer la secuencia dada por el algoritmo del Cubo de Rubik y separarla en caras y grados girados. Esto es necesario dado que la secuencia viene dada algo así como: $[' F, B2, U3, R']$, donde las letras hacen referencia a la cara pertinente (Front, Back, Right, Left, Up y Down) y los números a los grados girados (90°, 180° y −90°) entendiendo la ausencia de número como 1.

Por ello, esta función irá leyendo carácter a carácter guardando las letras en la variable caras, los números en la variable giros y, en caso de haber llegado al carácter se separación, se comprobará que la longitud de caras y de giros sea igual. En caso contrario, se añadirá un '1' a giros ya que habrá una ausencia de número.

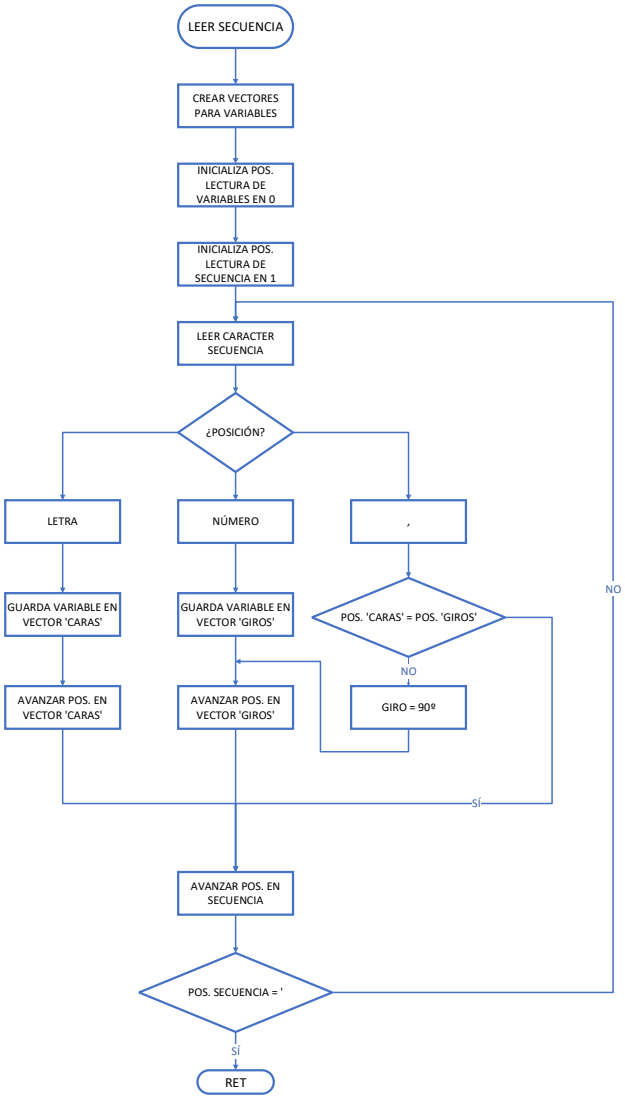
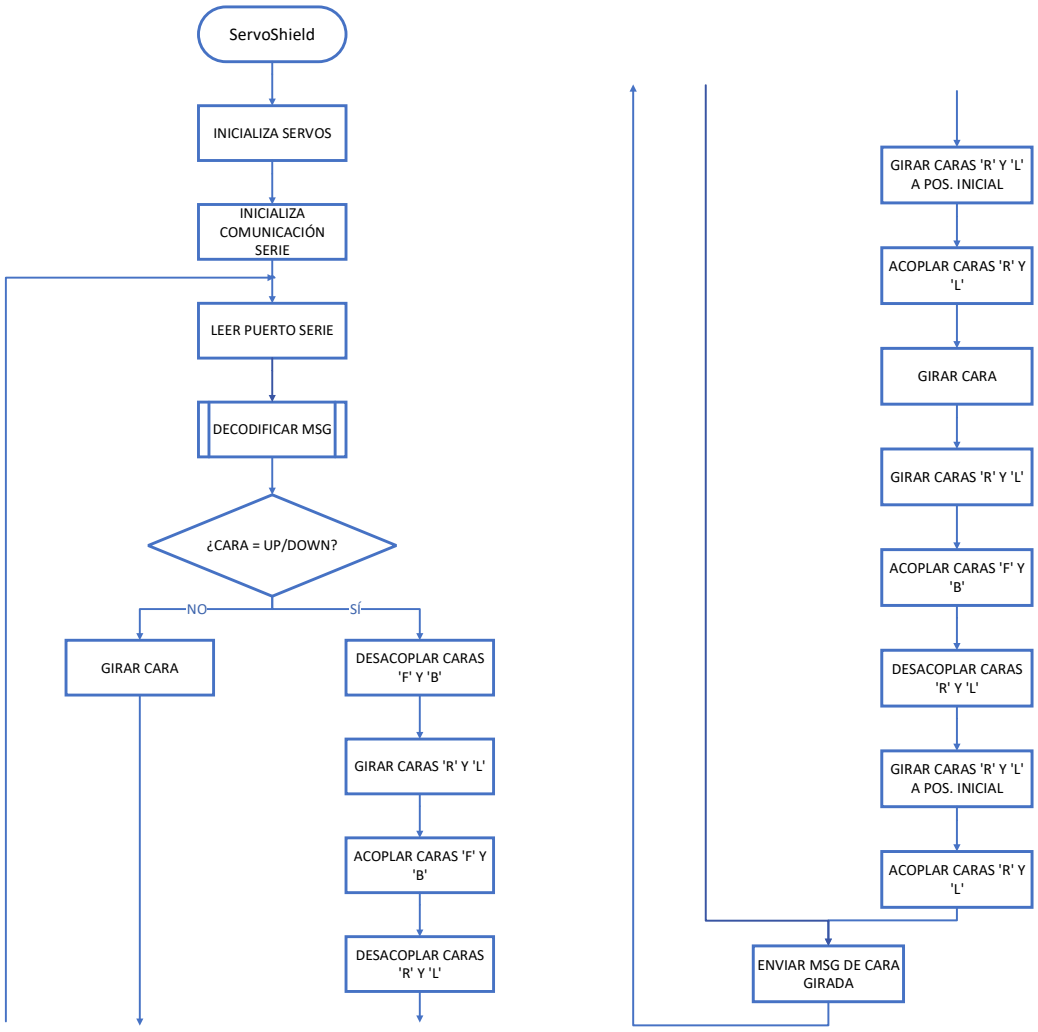


Fig. 5.14: Diagrama de flujo de la función Leer Secuencia.

Finalmente, por parte de *Arduino IDE*, el programa principal (figura 5.15) dará comienzo inicializando los servo motores así como el driver encargado de darles alimentación. Seguidamente se inicializará la comunicación serie y leerá el puerto serie, quedando a la espera de un mensaje.

Una vez recibido el mensaje, lo 'decodificará' separándolo en si hay que girar la cara superior/inferior o el resto de caras. Esto es necesario dado que para girar las caras superior e inferior es necesario una secuencia especial que ha sido detallada en el apartado 4.2 como segunda referencia. Al finalizar la operación, se enviará un mensaje de confirmación por el puerto serie y se volverá a la espera de leer una orden.



(a) Parte superior (b) Parte inferior

Fig. 5.15: Diagrama de flujo del cuerpo principal en *Arduino IDE*.

5.5 Resultado de la implementación

Una vez realizado el montaje y conexiones del sistema, se han realizado paulatinamente pruebas de las distintas funciones del programa con el objeto de obtener un sistema robusto y evitar la aparición de varios errores simultáneos. Esto nos ha permitido trabajar en los errores según iban surgiendo facilitando así el 'debug' del código.

Se empezó probando la comunicación serie mediante pruebas sencillas de enviar y recibir mensajes. Una vez eso funcionaba, se enviaron mensajes de movimiento de servo motores uno a uno y mediante la ventana de comandos de *Matlab*. En este punto surgieron problemas con algunos de los servo motores dado que no estaban bien configurados o no estaban bien conectados. Sin embargo, gracias a ir probando las funciones por separado, este error se encontró y solucionó de manera rápida y sencilla, haciéndonos perder el mínimo tiempo en localizar el fallo.

Seguidamente, se probó la función de lectura de la secuencia confirmando su correcto funcionamiento y se pasó a probar una secuencia de un único movimiento como podría ser $['B2']$. Comprobado que todo funciona como se espera, se paso a probar uno a uno todos los servo motores con todas sus correspondientes combinaciones de giro. Esta vez, volvió a dar un error en ciertos servo motores a la hora de trabajar con las caras superior e inferior debido a que se intentó realizar la secuencia demasiado rápido y el programa colapsó. El punto positivo es que, gracias a la espera de la confirmación de movimiento mediante la comunicación serie, lo que colapsó fue el puerto serie y no el Cubo de Rubik.

Una vez solucionado este pequeño contratiempo, se procedió a realizar una secuencia completa de mas de siete movimientos y todo funciona correctamente. Por lo que podemos afirmar que el sistema es robusto y con la secuencia adecuada podremos resolver el cubo de Rubik. Sin embargo, debido a la espera de confirmación, el sistema es lento pero seguro.

6 Pliego de Condiciones

6.1 Definición y alcance

El presente proyecto consta de un sistema robotizado con cuatro pinzas de amarre, especialmente diseñado para la resolución del Cubo de Rubik.

El alcance de dicho proyecto abarca todas las disposiciones técnicas y procesos que integran el mismo, los cuales han sido descritos con exactitud a lo largo de esta sección.

6.2 Objeto

El objeto de esta memoria es establecer las condiciones generales y técnicas mínimas requeridas para el desarrollo del proyecto anteriormente mencionado.

En este proyecto no se establecerán condiciones económicas y legales, por ser de tipo académico, y no ser un producto que se vaya a comercializar.

6.3 Condiciones y normas de carácter general

Este proyecto debe cumplir con obligatoriedad todas las condiciones y normas de carácter general por las que se rigen los proyectos industriales.

El objetivo de estas normas tiene como fin la garantía de la seguridad del autor y todos aquellos que participen en cualquier fase del desarrollo del proyecto.

Aquí se destacará la normativa común a todos los proyectos de este tipo, siendo las siguientes:

- UNE 157001:2014. "Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico".

- UNE-EN ISO 11442:2006. "Documentación técnica de productos. Gestión de documentos (ISO 11442:2006)".

6.4 Condiciones de especificaciones técnicas

En lo referente a las condiciones de especificaciones técnicas, estas vienen mucho más definidas por la propia naturaleza del proyecto, correspondiéndose con el aseguramiento de la seguridad y la calidad de los procesos concretos inherentes al mismo.

6.4.1 Normativa aplicable a la generación de planos.

- UNE 1027-95. "Dibujos técnicos. Plegado de planos".

- UNE 1039:1994. "Dibujos técnicos. Acotación. Principios generales, definiciones, métodos de ejecución e indicaciones especiales".

- UNE 1032:1982. "Dibujos técnicos. Principios generales de representación."

6.4.2 Normativa aplicable a la gestión de calidad.

- UNE-EN ISO 8402:1995. "Gestión y aseguramiento de la calidad. (ISO 8402:1994)."

- UNE-EN ISO 10007:1997. "Gestión de la calidad. Directrices para la gestión de la configuración. (ISO 10007:1995)."

- UNE-EN 45020:1998. "Normalización y actividades relacionadas."

- UNE-EN ISO 9001:2000. "Sistemas de gestión de la calidad. Requisitos."

- UNE-EN ISO 9004:2000. "Sistemas de gestión de la calidad. Directrices para la mejora del desempeño"

6.4.3 Normativa aplicable a proyectos eléctricos.

- Reglamento Electrotécnico para Baja Tensión (REBT) (Decreto 2413/1973) y sus Instrucciones Técnicas Complementarias (MIBT).

6.4.4 Normativa sobre fabricación y componentes.

- UNE-EN ISO 16090-1:2017. "Seguridad de las máquinas herramienta. Centros de mecanizado, centros de fresado, máquinas transfer. Parte 1: Requisitos de seguridad".

- UNE-EN ISO 888:2019. "Elementos de fijación. Pernos, tornillos y espárragos. Longitudes nominales y longitudes roscadas".

- UNE-EN ISO 898-1:2015. "Características mecánicas de los elementos de fijación de acero al carbono y de acero aleado. Parte 1: Pernos, tornillos y bulones con clases de calidad especificadas. Rosca de paso grueso y rosca de paso fino".

- UNE-EN ISO/ASTM 52910:2020 "Fabricación aditiva. Diseño. Requisitos, directrices y recomendaciones (ISO/ASTM 52910:2018)".

7 Presupuesto

En esta sección se procede a la descripción y desarrollo de los costes de realización desglosados para una mayor comprensión.

Cabe destacar una serie de consideraciones a tener en cuenta:

- La elaboración del proyecto, así como su fabricación, será realizado por una única persona.

- Se elaborará dos presupuestos finales, separando el coste de fabricación propiamente dicho del coste de diseño. Incluyendo en este segundo los costes de personal, fabricación, instalaciones, licencias de software y horas de diseño.

Esta última consideración es debida a que si se incluyese todo en un único presupuesto, la cifra ascendería a niveles demasiado altos para una sola máquina. Sin embargo, al considerar la fabricación en serie y venta por unidad, el precio del sistema será el del coste de su fabricación más un porcentaje del coste de diseño. De este modo, el coste debido a su diseño se irá amortizando paulatinamente sin tener un producto demasiado caro para el mercado.

7.1 Desglose de costes de material

Artículo	**Coste Unitario (€)**	**Unidades**	**Coste(€)**
Arduino Uno	29.28	1	29.28
Servo Motor DM996	6.9	8	55.20
Driver PCA9685	8.99	1	8.99
Porta Pilas	6.39	1	6.39
Cables Macho-Hembra 200mm	0.05	27	1.35
Paquetes de Pilas AA	1.5	1	1.5
Elementos de Fijación	24.99	1	24.99
Nylon en Polvo	9.7 €/Kg	0.4 Kg	3.88
Tablero de Madera $350 \times 350 \times 20$	8.89	1	8.89
Coste Total			**140.47**

Tabla 7.1: Desglose de los costes de material.

La tabla 7.1 muestra el desglose de los costes debido únicamente a los materiales.

7.2 Desglose de costes repercutidos

Seguidamente se procede a detallar todos aquellos costes que repercuten al proyecto como son los costes de personal, de fabricación e instalaciones y de software.

7.2.1 Personal

En el coste de personal se tendrá en cuenta únicamente el trabajo realizado por el autor como se ha mencionado anteriormente. Esto es debido a que el coste de los operarios para la fabricación se tendrá en cuenta en el siguiente apartado.

Habiéndose dedicado una media de 2.3 horas diarias durante 4 meses, las horas totales de trabajo ascienden a:

- **Horas Totales:** $2h \times 5d\acute{i}as \times 20semanas = 184horas$

Aplicando estas horas la tarifa media de coste de un ingeniero en España, la cual se sitúa en 15.40€/h, se obtendrá el siguiente coste de personal:

- **Coste de Ingeniero:** $184h \times 15.40€/h = 2833.60€$

7.2.2 Fabricación e instalaciones

En el coste de fabricación si que se incluirá el coste del técnico necesario para la fabricación. En este caso, tomaremos un salario anual del técnico como 22000€. Pasando esto a salario mensual, sería cerca de 1800€, con una jornada laboral de 40 horas semanales, obtendríamos un coste por hora de 11.46€.

Artículo	Coste Horario (€/h)	Unidades (h)	Coste (€)
Sinterizado Láser	11.46	8	91.70
Corte Láser	11.46	1.5	17.19
Coste Total			**108.85**

Tabla 7.2: Desglose de los costes de fabricación.

Para el cálculo de costes de instalaciones se estimará la parte proporcional del coste de la maquinaria empleada por el proyecto. Todo esto teniendo en cuenta que se realizará el mismo proyecto durante toda la vida útil de la maquinaria empleada. Por ello, en la tabla 7.3 queda desglosado este coste.

Artículo	Coste del Artículo (€)	Vida Útil (h)	Uso (h)	Coste (€)
Máquina Sinterizado Láser	20000	20000	8	8
Máquina Corte Láser	15000	30000	1.5	0.75
Ordenador	2000	10000	184	36.8
Coste Total				**45.55**

Tabla 7.3: Desglose de los costes de instalación.

finalmente, juntando los costes de fabricación e instalaciones obtenemos los costes totales mostrados en la tabla 7.4.

Artículo	Coste (€)
Fabricación	108.85
Instalaciones	45.55
Coste Total	**154.4**

Tabla 7.4: Desglose de los costes totales de Fabricación e Instalaciones.

7.2.3 Software y licencias

En los gastos de Software y licencias emplearemos el mismo método que en el apartado anterior, estimando el precio por proyecto realizado. En nuestro caso particular se han empleado tres softwares diferentes los cuales son: *Matlab*, *Solid Works* y *Microsoft Office*.

- Teniendo en cuenta que una licencia de **Matlab** ronda los 2020€ anuales. Contando con las horas realizadas con esta licencia, se estima que se pueden realizar cerca de 8 proyectos al año de la duración de este, lo que nos deja con un coste por licencia de **252.5€**.

- Una licencia anual de **Solid Works** estándar cuesta cerca de 3100€ y, aproximadamente, se puede realizar el diseño de cerca de 12 proyectos, dado que en esta licencia se han empleado muchas menos horas que en la anterior. Esto nos deja con un coste de **258.3€**.

- Finalmente, la licencia anual de **Microsoft Office** ronda los 70€. Realizando unos 8 proyectos anuales, se tendría un coste de **7.78€**.

Esto nos dejaría con un **coste total** de software y licencias de aproximadamente **518.61€**.

7.3 Presupuesto final

Finalmente, se procede a resumir el presupuesto completo del proyecto teniendo en cuenta los costes de producción de un único artículo y estimando que se fabricará mas de una unidad al año.

7.3.1 Presupuesto de costes de material

Contando solo con los costes de materiales, este proyecto tiene un presupuesto de **140.47€**. Sin embargo, cabe destacar que este es el coste por unidad teniendo en cuenta que es fabricado por el usuario y que este dispone de la maquinaria necesaria para llevarlo a cabo.

7.3.2 Presupuesto de costes total

Juntando todos los costes del proyecto, obtenemos un total de **967.88€** tal y como se muestra en la tabla 7.5. Sin embargo, este sería el precio si se vendiese una única unidad dado que el Software, así como la de personal de diseño, se emplea exclusivamente en la fase de diseño, por lo que ese coste se comparte entre todas las unidades vendidas.

Por otro lado, al realizar una producción en serie, los costes de fabricación, instalaciones y personal se reducen drásticamente ya que se optimizaría el tiempo y recursos empleados.

Artículo	**Coste(€)**
Materiales	140.47
Personal	2833.60
Fabricación e Instalaciones	154.4
Software y Licencias	518.61
Coste Total	**3647.08**

Tabla 7.5: Desglose de los costes totales del proyecto.

Con todo esto y contando una producción en serie, se podría estimar un precio de 283 €/unidad, 257 €/unidad y 220 €/unidad teniendo en cuenta que al año se fabrican 1000 unidades, 5000 unidades y 10000 unidades respectivamente. Como podemos observar, al repartir los costes de personal y software entre las unidades fabricadas, el coste se reduce drásticamente, llegando a un 6% del coste total en caso de fabricar 10000 unidades.

8 Conclusiones y Propuestas de Mejora

En este Trabajo de Fin de Máster se ha desarrollado exitosamente el diseño, programación e implementación de un prototipo capaz de resolver el Cubo de Rubik. Por otro lado, se ha demostrado la capacidad del sistema de seguir una serie de movimientos dados manteniendo una alta precisión a la hora de girar las caras del cubo, tanto en la simulación como en la implementación real.

Por tanto, se consideran todos los objetivos propuestos inicialmente como completados con éxito dado que las diferentes fases del proyecto han sido completadas de manera satisfactoria. Se ha realizado un diseño sencillo y funcional capaz de mover cualquier cara del cubo, diseño el cual ha permitido la realización de un modelo CAD para la simulación y predicción de su futuro comportamiento. Se han solventado problemas a la hora de fabricar de manera rápida y eficaz, así como errores en la programación, hechos que han ayudado al desarrollo y mejora del pensamiento crítico y resolutivo, haciendo crecer como ingeniero a su creador.

Sin embargo, cabe destacar que el sistema dispone de una serie de limitaciones y carencias que podrían solventarse en un futuro. Por ello, se desarrollarán una serie de recomendaciones y **propuestas de mejora** las cuales permitirán mejorar el rendimiento de sistema:

- Aunque, de manera manual, es posible obtener la secuencia a realizar para la resolución del cubo, sería útil implementar un sistema de **visión artificial** capaz de reconocer la situación del cubo, mandar la información al programa que corre el algoritmo de resolución y obtener la secuencia de manera automática. Este sistema eliminaría la intervención del usuario automatizando aún más el proceso de resolución.
- El sistema resuelve el cubo de manera robusta, sin embargo, es un sistema muy lento dado que tiene que esperar constantemente la confirmación de realización de cada movimiento. Esto podría solventarse creando una función en el entorno *Matlab* capaz de manejar el driver PCA9685, **eliminando** así la **comunicación serie** y aumentando drásticamente la velocidad del sistema.

9 Anexo: Código

9.1 Función principal en *Matlab*

```
clear all;
clear all port;
clc

COM = 'COM3';
delete(instrfind({'Port'},{COM}));
Arduino = serialport(COM,9600);    %Puerto Serie

flush(Arduino);

Algoritmo_Rubik =['B3,L,R,F2,L3,B2,F3,'];

[Caras Giro]=Algoritmo(Algoritmo_Rubik);

%Girar -90º -> '
%Girar 180º -> 2

servo_Front = "F";
servo_Back = "B";

bucles = length(Caras);
for i=1:+1:bucles

    switch Caras(i)
        case 'U'
            Girar_Cara_Up(Arduino,servo_Back, Giro(i));
        case 'D'
            Girar_Cara_Up(Arduino,servo_Front, Giro(i));
        case 'F'
            Servo_Movimiento = "V";
            Girar_Cara(Arduino,Caras(i), Giro(i), Servo_Movimiento);
        case 'B'
            Servo_Movimiento = "A";
            Girar_Cara(Arduino,Caras(i), Giro(i), Servo_Movimiento);
        case 'R'
            Servo_Movimiento = "M";
            Girar_Cara(Arduino,Caras(i), Giro(i), Servo_Movimiento);
        case 'L'
            Servo_Movimiento = "N";
            Girar_Cara(Arduino,Caras(i), Giro(i), Servo_Movimiento);
    end

    flush(Arduino);

end
```

9.2 Función lectura de la secuencia del Algoritmo en *Matlab*

```
function [Caras Giro]=Algoritmo(Algoritmo_Rubik)
    %Contadores para saber qué posición de Caras y Giro toca
    i_Caras=1;
    i_Giro=1;
    Pos = 0;
    for i=1:+1:length(Algoritmo_Rubik)
        Pos = Pos+1;

        %Pos_actual=Pos-Pos_ini
        if Algoritmo_Rubik(i) == ','
            switch Pos
                case 2
                    Giro(i_Giro) = "1";
                    i_Giro = i_Giro+1;
                    Pos = 0;
                case 3
                    if Algoritmo_Rubik(i-1) == '2'
                        Giro(i_Giro) = "2";
                        i_Giro = i_Giro+1;
                        Pos = 0;
                    else
                        Giro(i_Giro) = "3";
                        i_Giro = i_Giro+1;
                        Pos = 0;
                    end %END IF
                end %END SWITCH
        elseif Pos == 1
            Caras(i_Caras) = Algoritmo_Rubik(i);
            i_Caras = i_Caras+1;
        end %END IF
    end %END FOR

    L_Giro = length(Giro);
    Caras;
    Giro;

end
```

9.3 Función Girar Cara en *Matlab*

```
%Funcion para girar caras del cubo de rubik
function Done = Girar_Cara(Arduino,Caras, Giro, Servo_Movimiento)
    %La cara solo puede girar 0º(1), 90º(2) o 180º(3). Si no introducimos
    %uno de esos 3 valores se puede romper el cubo

    Instruccion = Caras + Giro;
    Instruccion_0 = Caras + "0";
    Movimiento_0 = Servo_Movimiento + "0";
    Movimiento_90 = Servo_Movimiento + "1";

    if(Giro == "3")
        Instruccion = Caras + "1";

        EnviarArduino(Arduino, Movimiento_90);   %Desacoplamos la pinza

        EnviarArduino(Arduino, Instruccion);       %Giramos la pinza a posicion 90ª

        EnviarArduino(Arduino, Movimiento_0);   %Acoplamos la pinza

        EnviarArduino(Arduino, Instruccion_0);       %Giramos la pinza a posicion 0ª

    else
        EnviarArduino(Arduino, Instruccion);

        EnviarArduino(Arduino, Movimiento_90);   %Desacoplamos la pinza

        EnviarArduino(Arduino, Instruccion_0);       %Giramos la pinza a posicion 0ª

        EnviarArduino(Arduino, Movimiento_0);   %Acoplamos la pinza
    end
end
```

9.4 Función Girar Cara Superior e Inferior en *Matlab*

```
%Funcion para girar caras del cubo de rubik

function Girar_Cara_Up(Arduino,Servo_Cara, Grados)
    %La cara solo puede girar 0º(0), 90º(0.5) o 180º(1). Si no introducimos
    %uno de esos 3 valores se puede romper el cubo
%     R_0 = "R0";
%     R_90 = "R1";
    L_0 = "L0";
    L_90 = "L1";
    F_0 = "F0";
%     F_90 = "F1";
    B_0 = "B0";
%     B_90 = "B1";

    %Mensaje a enviar para acoplar/desacoplar las pinzas de cada una de las caras
    Mov_R_0 = "M0";
    Mov_R_90 = "M1";
    Mov_L_0 = "N0";
    Mov_L_90 = "N1";
    Mov_F_0 = "V0";
    Mov_F_90 = "V1";
    Mov_B_0 = "A0";
    Mov_B_90 = "A1";
    Girar_Cubo = "U4";
    Cubo_Default = "D5";

    if(Grados ~= '1' && Grados ~= '2' && Grados ~= '3')
        display('Error:');
    else

        %Giramos el cubo para tener la cara UP y DOWN accesibles
        EnviarArduino(Arduino, Mov_L_90);
        pause(0.5);
        EnviarArduino(Arduino, L_90);
        pause(0.5);
        EnviarArduino(Arduino, Mov_L_0);
        pause(0.5);

        %Desacoplamos las caras que no usaremos
        EnviarArduino(Arduino, Mov_F_90);
        EnviarArduino(Arduino, Mov_B_90);
        pause(0.5);

        EnviarArduino(Arduino, Girar_Cubo);
        pause(0.5);

        %Acoplamos las caras que no usaremos
        EnviarArduino(Arduino, Mov_F_0);
        EnviarArduino(Arduino, Mov_B_0);
        pause(0.5);

        %Desacoplamos las caras giradas
        EnviarArduino(Arduino, Mov_R_90);
        pause(0.5);

```

```
56          %Giramos la cara correspondiente
57          if(Servo_Cara == "F")
58              Servo_Movimiento = "V";
59              Girar_Cara(Arduino,Servo_Cara, Grados, Servo_Movimiento);
60          else
61              Servo_Movimiento = "A";
62              Girar_Cara(Arduino,Servo_Cara, Grados, Servo_Movimiento);
63          end
64
65          %Giramos la pinza usada a posicion 0ª
66          if(Servo_Cara == "F")
67              EnviarArduino(Arduino, Mov_F_90);
68              pause(0.5);
69              EnviarArduino(Arduino, F_0);
70          else
71              EnviarArduino(Arduino, Mov_B_90);
72              pause(0.5);
73              EnviarArduino(Arduino, B_0);
74          end
75
76          %Giramos el cubo a posicion inicial
77          EnviarArduino(Arduino, Mov_R_0);
78          pause(0.5);
79
80          %Desacoplamos las caras que no usaremos
81          if(Servo_Cara == "F")
82              EnviarArduino(Arduino, Mov_B_90);
83          else
84              EnviarArduino(Arduino, Mov_F_90);
85          end
86          pause(0.5);
87
88          EnviarArduino(Arduino, Cubo_Default);
89          pause(0.5);
90
91          %Acoplamos las caras que no usaremos
92          EnviarArduino(Arduino, Mov_F_0);
93          EnviarArduino(Arduino, Mov_B_0);
94          pause(0.5);
95
96          %Desacoplamos las caras giradas
97          EnviarArduino(Arduino, Mov_L_90);
98          pause(0.5);
99
100         %ponemos los servos de giro en posicion 0
101         EnviarArduino(Arduino, L_0);
102         pause(0.5);
103
104         %Acoplamoss las caras giradas
105         EnviarArduino(Arduino, Mov_L_0);
106         pause(0.5);
107
108     end
109
110 end
```

9.5 Función principal en *Arduino*

ServoShield_V2

```
//SCL -> A5
//SDA -> A4

#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>


#define PERIODO 10

//Definimos los servos

#define Servo_Off 8
#define Servo_Giro_Cubo 9
#define Servo_Default_Cubo 10
#define Servo_Front 0
#define Servo_Back 1
#define Servo_Right 2
#define Servo_Left 3
#define Servo_mov_Front 4      //Verde
#define Servo_mov_Back 5       //Azul
#define Servo_mov_Right 6      //Magenta
#define Servo_mov_Left 7       //Naranja

int EstadoServo = Servo_Off;

#define GRADOS_MAX 4096
#define GRADOS_MIN 0


//Desfases de los Servos
int Desfase_R = 30;    //0.04  --> 163 (duty)
int Desfase_R_180 = 47;    //Tiene un desfase para 180 distinto que para 0
int Desfase_L_0 = 30;    //0.04
int Desfase_L_180 = 54;    //Tiene un desfase para 180 distinto que para 0
int Desfase_L_90 = 48;    //Tiene un desfase para 180 distinto que para 0
int Desfase_F_0 = 23;
int Desfase_F_90 = 20;
int Desfase_F_180 = 33;    //Tiene un desfase para 180 distinto que para 0
int Desfase_B_0 = 20;
int Desfase_B_180 = 45;    //Tiene un desfase para 180 distinto que para 0
int Desfase_B_90 = 35;    //Tiene un desfase para 180 distinto que para 0

int Desfase_V = 10;
int Desfase_A = 0;
int Desfase_M = -25; //-15
int Desfase_N = 30;


//Variables para periodo de cilo
int t1 = 0 ;
int t2 = 0 ;

String mensaje;
char servoUsar;
char GirarCara;

int Grados; //Entre 0 y 4096 para duty
int GradosD;  //Grados + Desfase
int GradosL;
int GradosR;
```

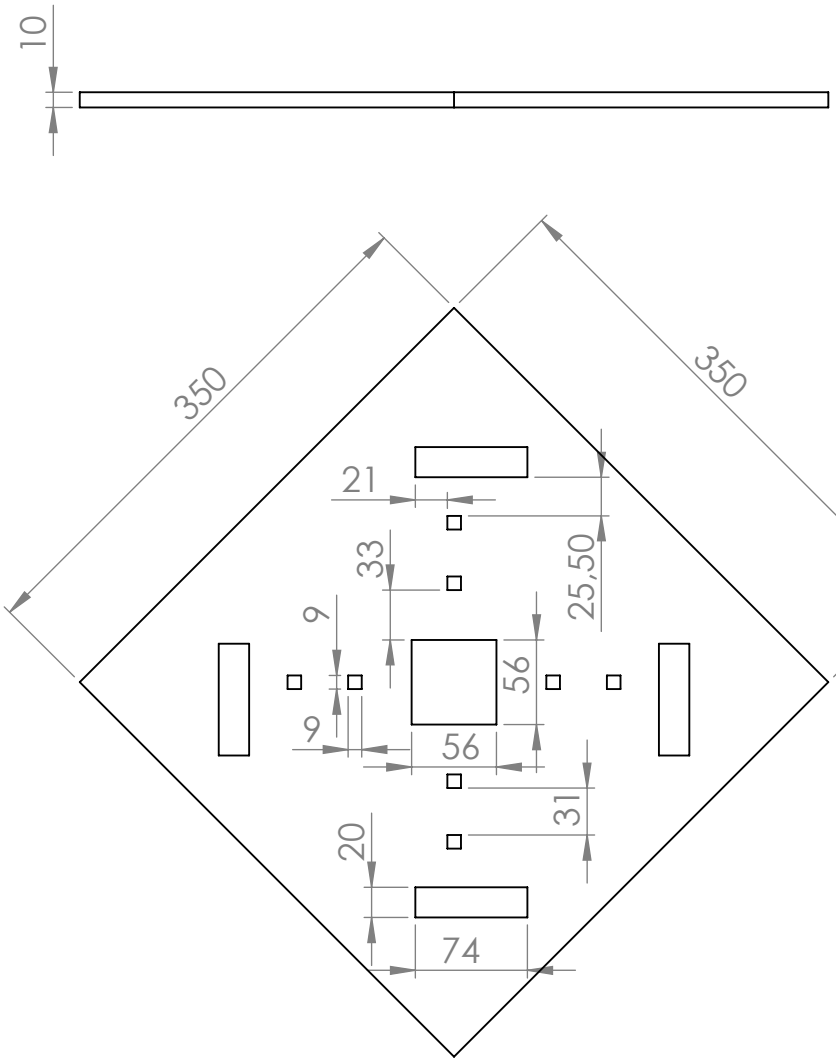
```
Adafruit_PWMServoDriver servoController = Adafruit_PWMServoDriver(0x40);
//const uint8_t frequency = 1600;
const uint8_t frequency = 50;    // Frecuencia PWM de 50Hz o T=20ms
const uint16_t ServoMinTicks = 102; // ancho de pulso en ticks para pocicion 0°
const uint16_t ServoMaxTicks = 512; // ancho de pulso en ticks para la pocicion 180°
const uint16_t Servo90Ticks = ServoMinTicks + ((ServoMaxTicks-ServoMinTicks)/2);


void setup() {
  Serial.begin(9600);
  servoController.begin();
  servoController.setPWMFreq(frequency );
}

void loop() {

  if (Serial.available()>0){
    mensaje = Serial.readStringUntil("\n");
//    int lengthMessage = mensaje.length();
//    for (int i = 1; i <= lengthMessage; i++){
//      if (i % 2 == 0){ // Es par
//          GirarCara = message[i-1];}
//      else {
//        servoUsar = message[i-1];}
//    }

    servoUsar = mensaje[0];
    GirarCara = mensaje[1];

    switch (GirarCara) {
      case '0':
              Grados = ServoMinTicks;
             //Grados = ServoMaxTicks;
             break;
      case '1':
             Grados = Servo90Ticks;
             break;
      case '2':
             //Grados = ServoMinTicks;
             Grados = ServoMaxTicks;
             break;
      case '3':
             Grados = ServoMaxTicks;
//             Serial.print("Error_3"); //Error, IDE no gira -90°
//             Serial.print('\n');
//             EstadoServo = Servo_Off;
             break;
      case '4':
             EstadoServo = Servo_Giro_Cubo;
             break;
      case '5':
             EstadoServo = Servo_Default_Cubo;
             break;
      default:
             Serial.print("Error_G");  //Error, No se ha detectado Giro
             Serial.print('\n');
             EstadoServo = Servo_Off;
      break;
    }
```

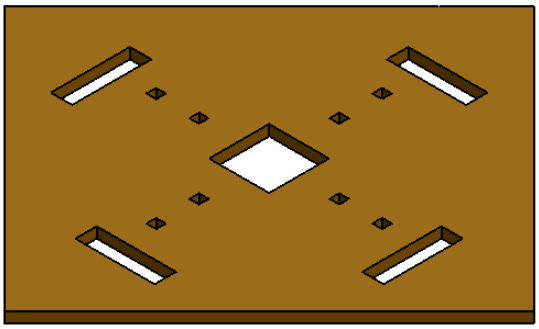
```
    switch (servoUsar) {
      case 'F':
             EstadoServo = Servo_Front;
             break;
      case 'B':
             EstadoServo = Servo_Back;
             break;
      case 'R':
             EstadoServo = Servo_Right;
             break;
      case 'L':
             EstadoServo = Servo_Left;
             break;
      case 'V':
             EstadoServo = Servo_mov_Front;
             break;
      case 'A':
             EstadoServo = Servo_mov_Back;
             break;
      case 'M':
             EstadoServo = Servo_mov_Right;
             break;
      case 'N':
             EstadoServo = Servo_mov_Left;
             break;
      case 'U':
             EstadoServo = Servo_Giro_Cubo;
             break;
      case 'D':
             EstadoServo = Servo_Default_Cubo;
             break;
      default:
             Serial.print("Error_S"); //Error, No se ha detectado Servo
             Serial.print('\n');
             EstadoServo = Servo_Off;
      break;
    }

switch (EstadoServo) {
      case Servo_Off:
             //
             break;
      case Servo_Front:
             if (Grados>500){
                GradosD = Grados + Desfase_F_180;
             }else if (Grados<200){
                GradosD = Grados + Desfase_F_0;
             }else{
                GradosD = Grados + Desfase_F_90;
             }
             //GradosD = Grados + Desfase_F;
             servoController.setPWM(Servo_Front, 0, GradosD);
             delay(50);
             Serial.print("S_D");  //Servo Front Done
             Serial.print('\n');
             break;
```

```
case Servo_Back:
        if (Grados>500){
           GradosD = Grados + Desfase_B_180;
        }else if (Grados<200){
           GradosD = Grados + Desfase_B_0;
        }else{
           GradosD = Grados + Desfase_B_90;
        }
        servoController.setPWM(Servo_Back, 0, GradosD);
        delay(50);
        Serial.print("S_D");  //Servo Back Done
        Serial.print('\n');
        break;
case Servo_Right:
        if (Grados>500){
           GradosD = Grados + Desfase_R_180;
        }else{
           GradosD = Grados +  Desfase_R;
        }
        //GradosD = Grados + Desfase_R;
        servoController.setPWM(Servo_Right, 0, GradosD);
        delay(50);
        Serial.print("S_D");  //Servo Right Done
        Serial.print('\n');
        break;
case Servo_Left:
        if (Grados>500){
           GradosD = Grados + Desfase_L_180;
        }else if (Grados<200){
           GradosD = Grados + Desfase_L_0;
        }else{
           GradosD = Grados +  Desfase_L_90;
        }
        servoController.setPWM(Servo_Left, 0, GradosD);
        delay(50);
        Serial.print("S_D");  //Servo Left Done
        Serial.print('\n');
        break;
case Servo_mov_Front:
        GradosD = Grados + Desfase_V;
        servoController.setPWM(Servo_mov_Front, 0, GradosD);
        delay(50);
        Serial.print("S_D");  //Servo Verde Done
        Serial.print('\n');
        break;
case Servo_mov_Back:
        GradosD = Grados + Desfase_A;
        servoController.setPWM(Servo_mov_Back, 0, GradosD);
        delay(50);
        Serial.print("S_D");  //Servo Azul Done
        Serial.print('\n');
        break;
case Servo_mov_Right:
        GradosD = Grados + Desfase_M;
        servoController.setPWM(Servo_mov_Right, 0, GradosD);
        delay(50);
        Serial.print("S_D");  //Servo Magenta Done
        Serial.print('\n');
        break;
```

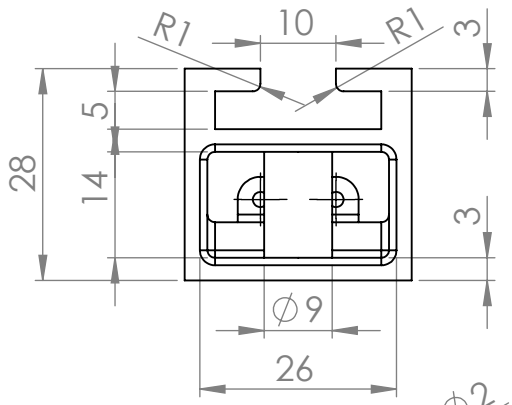
```
    case Servo_mov_Left:
            GradosD = Grados + Desfase_N;
            servoController.setPWM(Servo_mov_Left, 0, GradosD);
            delay(50);
            Serial.print("S_D");  //Servo Naranja Done
            Serial.print('\n');
            break;
    case Servo_Giro_Cubo:
            GradosL = ServoMinTicks+Desfase_L_0;
            GradosR = Servo90Ticks+Desfase_R;
            servoController.setPWM(Servo_Left, 0, GradosL);
            servoController.setPWM(Servo_Right, 0, GradosR);
            delay(50);
            Serial.print("S_D");  //Giro Cubo Done
            Serial.print('\n');
            break;
    case Servo_Default_Cubo:
            GradosR = ServoMinTicks+Desfase_R;
            GradosL = Servo90Ticks+Desfase_L_90;
            servoController.setPWM(Servo_Left, 0, GradosL);
            servoController.setPWM(Servo_Right, 0, GradosR);
            delay(50);
            Serial.print("S_D");  //Default Cubo Done
            Serial.print('\n');
            break;
    default:
    break;
  }
}
```

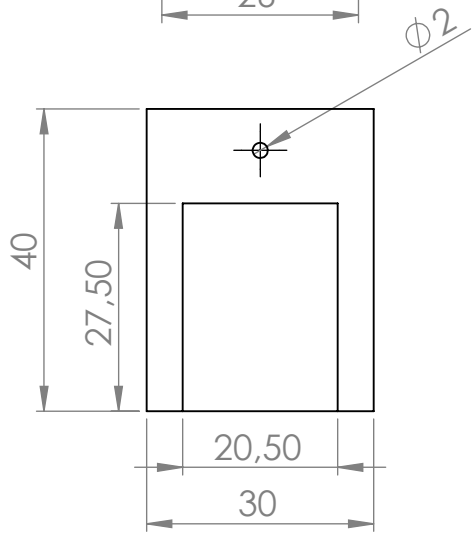
10 Anexo: Planos

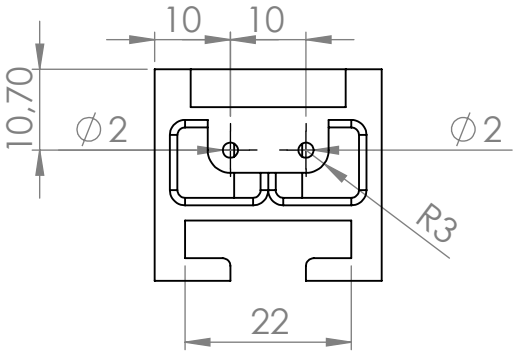


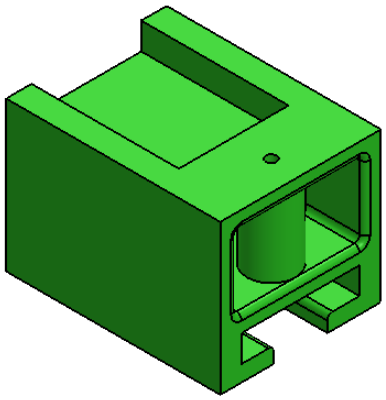


FECHA: 09/02/2023		PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO PARA RESOLUCIÓN AUTOMÁTICA DEL CUBO DE RUBIK	
PLANO: 1 DE 6			
MATERIAL: MADERA			
DIBUJADO POR: **Agustín Monzón Vivas**	N.º DE DIBUJO: BASE		A4
	ESCALA: 1:5	HOJA 1 DE 1	

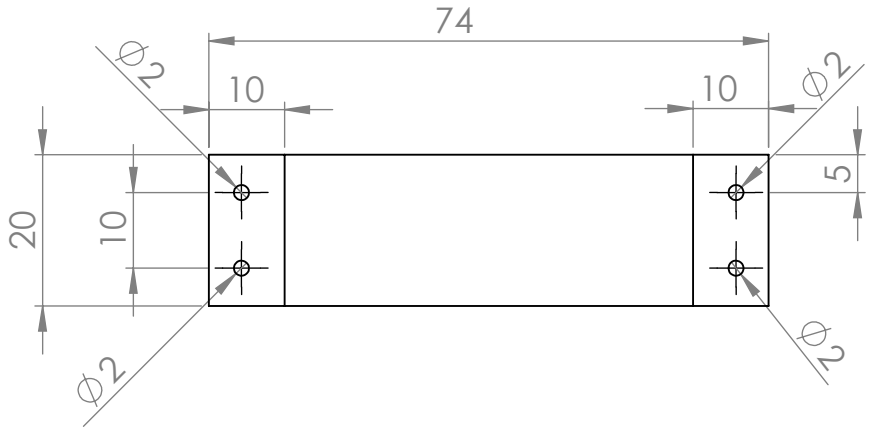


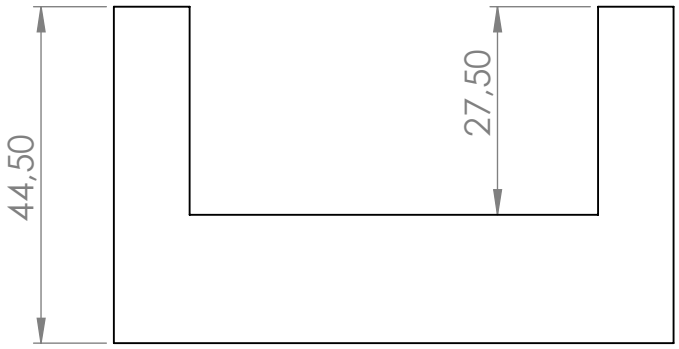


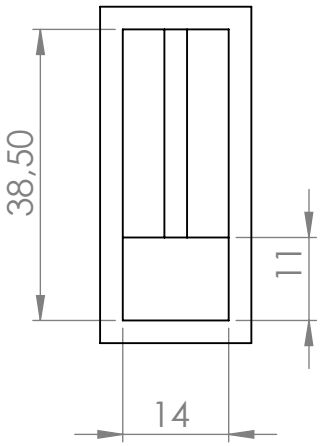


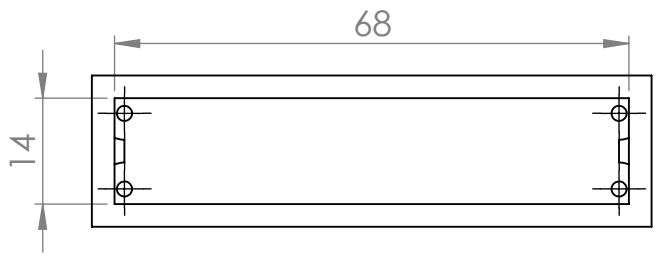


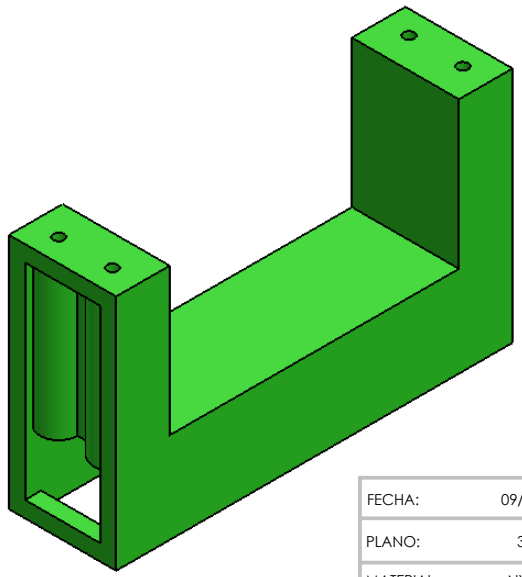
FECHA: 09/02/2023		PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO PARA RESOLUCIÓN AUTOMÁTICA DEL CUBO DE RUBIK	
PLANO: 2 DE 6			
MATERIAL: NYLON			
DIBUJADO POR: **Agustín Monzón Vivas**	N.º DE DIBUJO BASE SERVO		A4
	ESCALA: 1:1	HOJA 1 DE 1	



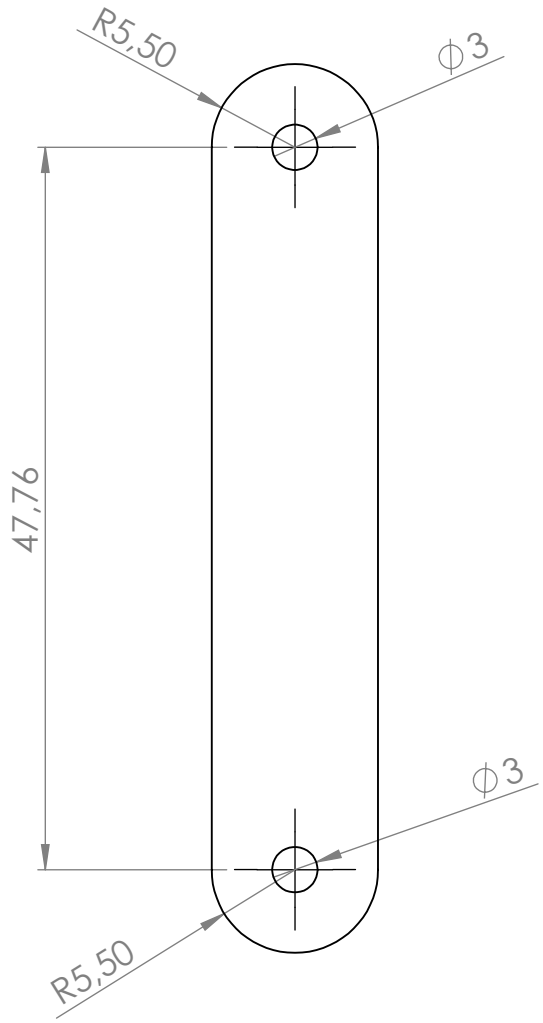


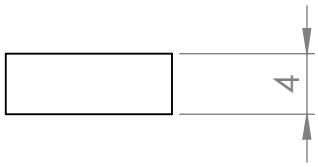


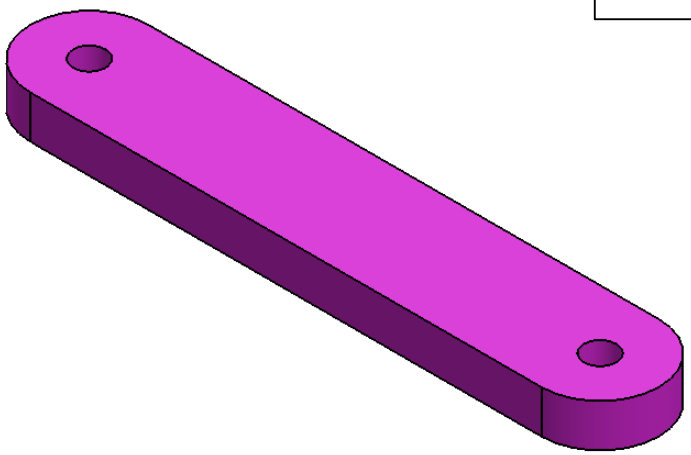




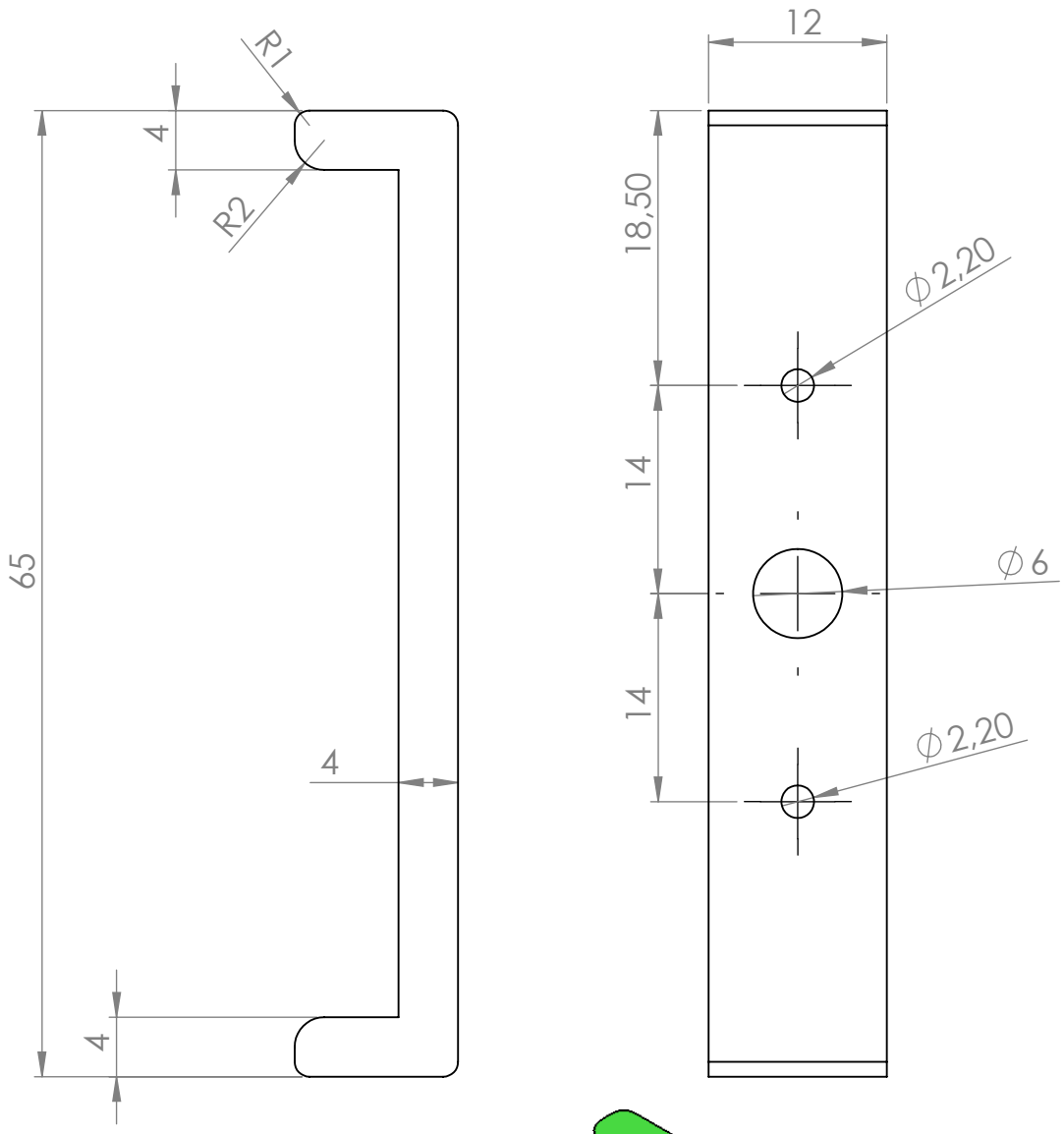
FECHA:	09/02/2023	PROYECTO:	DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO PARA RESOLUCIÓN AUTOMÁTICA DEL CUBO DE RUBIK
PLANO:	3 DE 6		
MATERIAL:	NYLON		
DIBUJADO POR:	**Agustín Monzón Vivas**	N.º DE DIBUJO	BASE SERVO EXT — A4
		ESCALA: 1:5	HOJA 1 DE 1

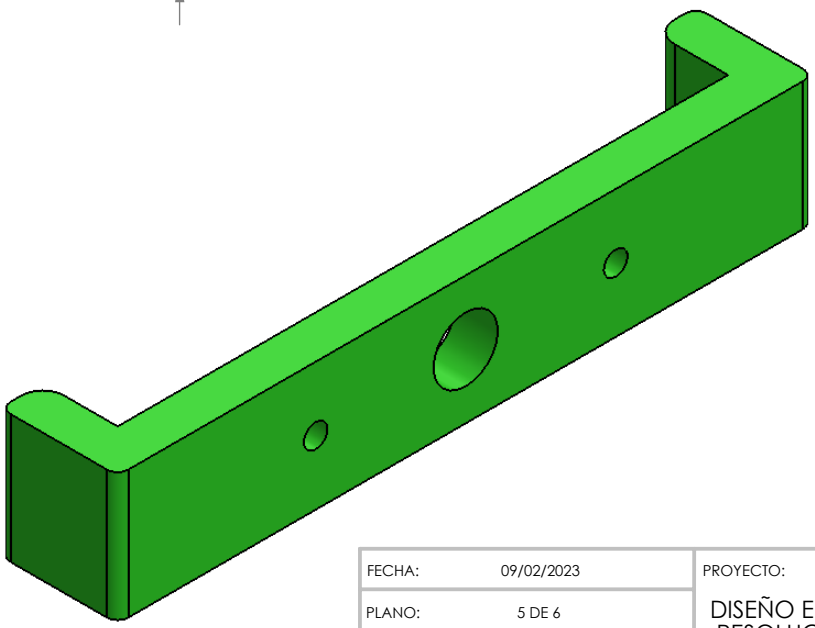




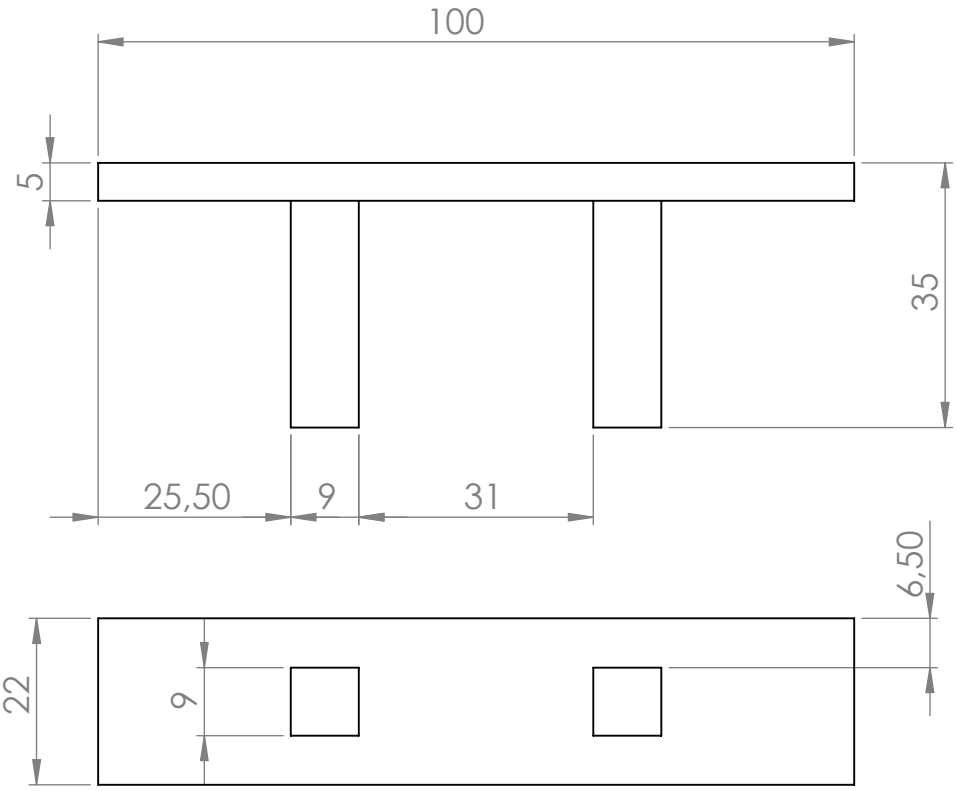


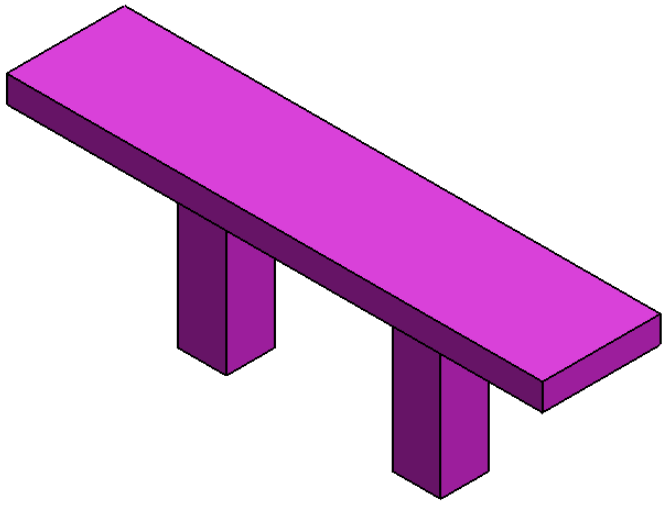
FECHA: 09/02/2023		PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO PARA RESOLUCIÓN AUTOMÁTICA DEL CUBO DE RUBIK	
PLANO: 4 DE 6			
MATERIAL: NYLON			
DIBUJADO POR: **Agustín Monzón Vivas**	N.º DE DIBUJO BRAZO		A4
	ESCALA: 2:1	HOJA 1 DE 1	





FECHA: 09/02/2023		PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO PARA RESOLUCIÓN AUTOMÁTICA DEL CUBO DE RUBIK	
PLANO: 5 DE 6			
MATERIAL: NYLON			
DIBUJADO POR: **Agustín Monzón Vivas**	N.º DE DIBUJO PINZA		A4
	ESCALA: 2:1	HOJA 1 DE 1	





FECHA: 09/02/2023		PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO PARA RESOLUCIÓN AUTOMÁTICA DEL CUBO DE RUBIK	
PLANO: 6 DE 6			
MATERIAL: NYLON			
DIBUJADO POR: **Agustín Monzón Vivas**	N.º DE DIBUJO RAÍL		A4
	ESCALA: 1:1	HOJA 1 DE 1	

REFERENCES

[1] Hope Reese U. A brief history of the rubik's cube, September 25, 2020.
URL https://www.smithsonianmag.com/innovation/brief-history-rubiks-cube-180975911/

[2] Alter A. He invented the rubik's cube. he's still learning from it., Sept 16, 2020.
URL https://www.nytimes.com/2020/09/16/books/erno-rubik-rubiks-cube-inventor-cubed.html

[3] Sadurní J M. El curioso origen húngaro del cubo de rubik, 29 de agosto de 2022.
URL https://historia.nationalgeographic.com.es/a/curioso-origen-hungaro-cubo-rubik_17584

[4] Antonio. Cubo de rubik y matemáticas, 12 abril, 2020.
URL https://divulgadores.com/cubo-de-rubik-y-matematicas/