Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Fabricación de prototipo de robot solucionador del cubo de Rubik

Autor: Jorge González Aparicio
Tutor: Manuel Gil Ortega Linares

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Fabricación de prototipo de robot solucionador del cubo de Rubik

Autor:
Jorge González Aparicio

Tutor:
Manuel Gil Ortega Linares
Catedrático

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Trabajo Fin de Grado: Fabricación de prototipo de robot solucionador del cubo de Rubik

Autor: Jorge González Aparicio

Tutor: Manuel Gil Ortega Linares

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

A mi madre, a mi padre, a mis amigos.

Agradecimientos

Seré breve. Gracias a todas las personas que de una u otra manera han contribuido a la creación de este trabajo. Gracias por el apoyo, y gracias por tener la paciencia para soportarme.

Jorge González Aparicio

Sevilla, 2022

Resumen

El objetivo de este Trabajo de Fin de Grado es crear un robot que pueda resolver el cubo de Rubik, partiendo de una configuración inicial cualquiera. Hemos decidido crear una estructura envolvente, que aloja 6 motores de paso a paso, los cuales se utilizan para girar cada una de las 6 caras del cubo. Para obtener la solución del cubo, hemos utilizado una implementación del algoritmo de Kociemba en Raspberry Pi, que permite introducir el estado inicial del cubo a partir de una interfaz de usuario, y otorga una solución cuasi-óptima tras un tiempo de procesado programable. La solución consiste en una serie de caracteres que representan movimientos de las caras del cubo, el cual se orienta según un sistema de referencia basado en posiciones relativas. Esta serie de caracteres se introducen en el puerto serie del Arduino Uno encargado del control de los motores, el cual los traduce a instrucciones para los motores, y se encarga de ejecutar la solución del cubo.

Abstract

The objective of this TFG is to create a robot which can solve the Rubik's cube from any starting configuration. We have decided to build an enveloping structure which hosts 6 stepmotors, which are used to twist each of the cube's faces. In order to solve the cube, we've used an implementation of Kociemba's algorithm in Raspberry Pi, which allows us to enter the initial state of the cube via a user interface, and produces a quasi-optimized solution after a programmable processing time. The solution consists of an array of characters, each representing a turn of the faces of the cube, which are oriented following a reference system based on relative positions. This array is then fed to the series port of the Arduino Uno tasked with the motor control, which translates it into instructions for the motors and then executes them.

Índice

ÍNDICE DE ILUSTRACIONES

Notación

Cubo/Puzle	Cubo de Rubik
Kociemba	Algoritmo de Kociemba
QTM	Métrica de Cuarto de Vuelta (Quarter Turn Metric)
HTM	Métrica de Media Vuelta (Half Turn Metric)
U	Up, cara superior del cubo/Giro de 90º en sentido horario de dicha cara.
D	Down, cara inferior del cubo/Giro de 90º en sentido horario de dicha cara.
L	Left, cara izquierda del cubo/Giro de 90º en sentido horario de dicha cara.
R	Right, cara derecha del cubo/Giro de 90º en sentido horario de dicha cara.
F	Front, cara frontal del cubo/Giro de 90º en sentido horario de dicha cara.
B	Back, cara trasera del cubo/Giro de 90º en sentido horario de dicha cara.
U′	Giro de 270º en sentido horario de la cara superior del cubo.
U2	Giro de 180º en sentido horario de la cara superior del cubo.
S.T.A.R.	Solver Tool for Aid with Rubik's. Nombre asignado al robot.
WCA	World Cube Association.
c/u	Cada unidad.

1 INTRODUCCIÓN

In life, unlike chess, the game continues after checkmate.

- Isaac Asimov -

La resolución de un cubo de Rubik es un problema complejo, regido por leyes simples. Para solucionarlo es necesario memorizar algoritmos simples que provocan cambios de posición predecibles de las piezas del rompecabezas, dejando en el mismo sitio otras piezas que ya están correctamente colocadas. Siguiendo estos algoritmos paso a paso, cualquier persona puede resolver un cubo dedicando el tiempo suficiente. Esencialmente, es un problema ideal para ser resuelto computacionalmente, ya que es fácil definir todos los estados posibles del puzle. Adicionalmente, se ha demostrado que es posible solucionar cualquier estado en 26 rotaciones de 90º de las caras, o menos. Sin embargo, no es práctico obtener por fuerza bruta la solución, ya que el número de estados posibles es 43.252.003.274.489.856.000 (4.3×10^{19})[1] [1]. A pesar de esto, se han desarrollado algoritmos que permiten solucionar el rompecabezas. Basándose en la teoría de grupos, el algoritmo de Kociemba, permite obtener soluciones subóptimas con un breve tiempo de computación, partiendo de una descripción del estado inicial del cubo. [2]

El propósito de este TFG es diseñar y fabricar un robot que sea capaz de utilizar el algoritmo de Kociemba y de realizar el giro de las caras del cubo de forma autónoma. Dicho robot se ha bautizado como S.T.A.R. (Solver Tool for Aid with Rubik's).

[1] Si se tiene en cuenta la orientación de los centros, el número de estados asciende a 88 580 102 706 155 225 088 000 (8.9×10^{22}). En el puzle original, y en este trabajo, no se consideran los giros de los centros.

2 OBJETIVOS Y METODOLOGÍA

If knowledge can create problems, it is not through ignorance that we can solve them.

- Isaac Asimov -

El objetivo principal del trabajo es lograr que STAR pueda superar el mejor tiempo personal del autor para solucionar un cubo de Rubik: 1 minuto. Además, se busca lograr integrar una amplia gama de los conocimientos adquiridos en el grado, así como emplear un presupuesto lo más reducido posible.

Temprano en el desarrollo del trabajo, se decidió utilizar un enfoque modular, para poder corregir los problemas que fueran surgiendo, mientras se dejaban intactas las partes que funcionaban, casualmente siguiendo el espíritu de la forma tradicional de solucionar el cubo.

2.1. Estado del arte.

Antes de comenzar con el proyecto, se revisó el estado del arte para tomar decisiones fundadas en ejemplos ya probados.

2.1.1 Albert Beer.

Ingeniero alemán, anterior récord del mundo. Su robot Sub1 Reloaded dejó el récord en 0.637 segundos. [3]

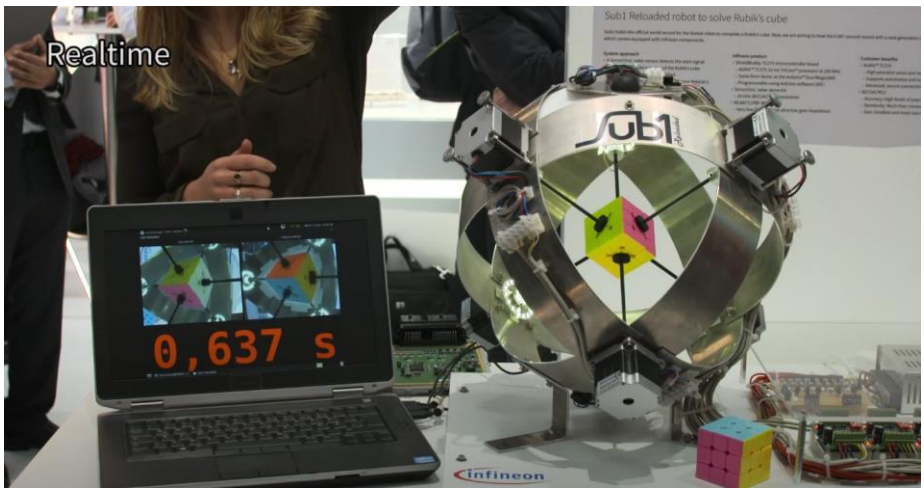

Ilustración 1. Sub1 Reloaded, robot de Albert Beer.

2.1.2 Ben Katz y Jared Di Carlo.

Estos investigadores del MIT (Massachusetts Institute of Technology) lograron en 2018 el actual récord mundial, con un tiempo de resolución de 0.38 segundos. [4] [5]

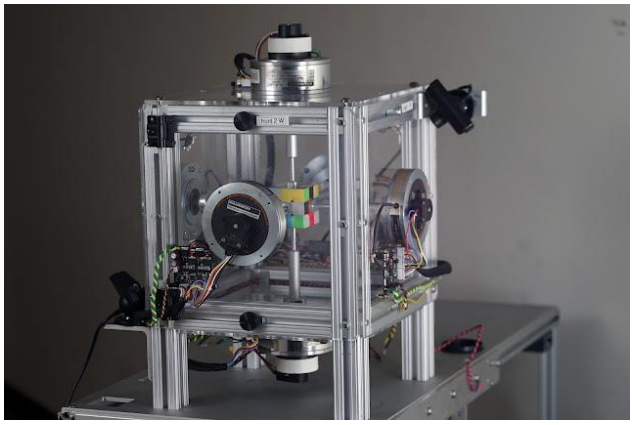

Ilustración 2. Robot de Ben Katz y Jared Di Carlo.

2.2. Análisis del estado del arte.

En ambos casos se emplea una estructura que rodea al cubo, y 6 motores asociados a cada uno de los ejes de rotación de las caras. Ambos proyectos emplean 2 cámaras situadas en vértices opuestos del cubo para captar el estado inicial, un sistema para realizar el cálculo de la solución, basado en el algoritmo de Kociemba, y un sistema para el control de los motores.

2.3. Módulos.

Basándonos en lo observado en el estado del arte, identificamos 4 módulos claramente diferenciados, que han sido utilizados en ambos casos estudiados.

2.3.1 Estructura y motores.

Las estructuras empleadas en ambos casos son metálicas y su longitud característica es en torno a 5 veces mayor que la del cubo, lo cual permite el acceso al puzle dentro del robot, así como la manipulación de este en caso de fallos, a cambio de requerir ejes de mayor longitud. Ambas estructuras son funcionalmente muy similares.

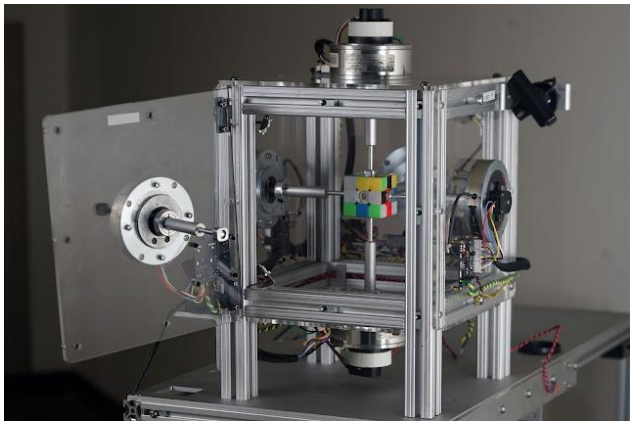

Ilustración 3. Apertura en el robot de Ben Katz, para acceder al cubo.

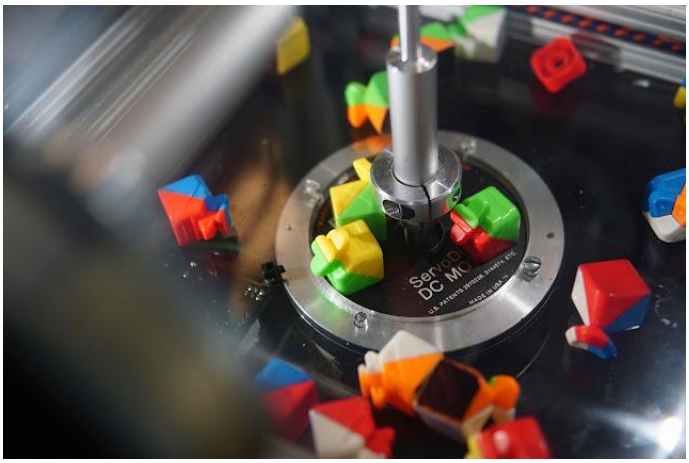

Ilustración 4. Ejemplo de estallido del cubo.[2]

Respecto a los motores, encontramos una diferencia significativa. En el caso de Albert Beer, utiliza motores paso a paso Nema 23. Ben Katz y Jared Di Carlo utilizan motores servodisco con encóders ópticos digitales, capaces de unas aceleraciones mucho mayores. Este es uno de los puntos clave que les permitió mejorar el récord mundial.

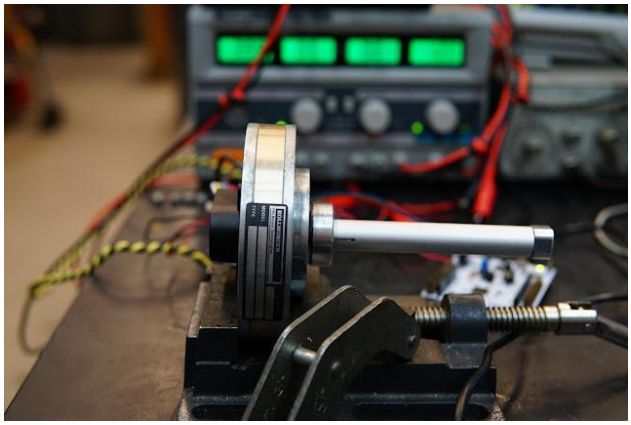

Ilustración 5. Motor ServoDisc N9M4T.

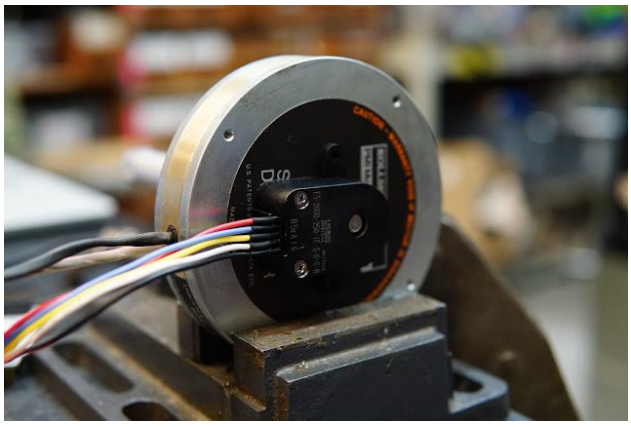

Ilustración 6. E5 Single-Ended optical encoder.

Para este TFG, hemos decidido emplear una estructura compuesta por paneles de madera de 0.5cm de grosor, cortados a láser, para lograr un compromiso entre las distintas especificaciones del trabajo: rigidez, precisión, coste. Se ha procurado reducir el tamaño de la estructura lo más posible, para procurar reducir costes, y aumentar la rigidez.

[2] https://youtu.be/hURpaTfIqQk Video del estallido del cubo del robot de Ben y Jared.

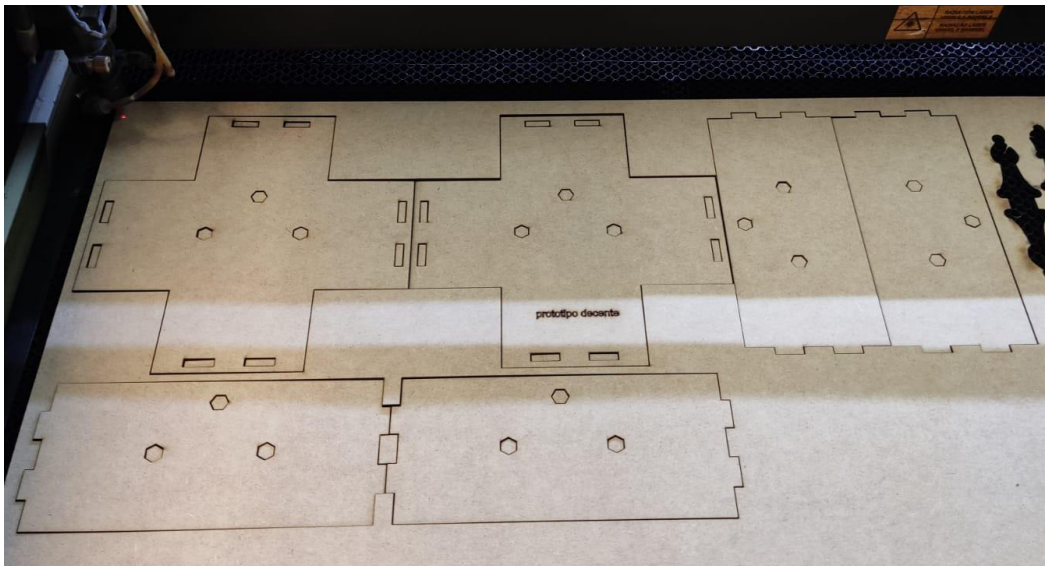

Ilustración 7. Corte de la estructura de madera.

En cuanto a los motores, se han empleado motores paso a paso, Nema 17, recuperados de una antigua impresora 3D. Teniendo en cuenta el Sub1 Reloaded de Albert Beer, es fácil ver que los tiempos de dichos motores son más que adecuados para el alcance de este TFG.

2.3.2 Control de motores.

Albert Beer empleó un procesador de la familia AURIX de Infeon Technologies, tanto para el control de los motores como para el procesamiento de imágenes y la implementación de Kociemba.

Ben Katz y Jared Di Carlo diseñaron controladores propios, empleando microcontroladores STM32F303K8. Crearon un sistema de sincronización y anticolisión de las caras del cubo, basado en una placa de puertas AND que se encargan de garantizar que los movimientos se ejecuten secuencialmente sin que uno comience antes de que acabe el anterior.

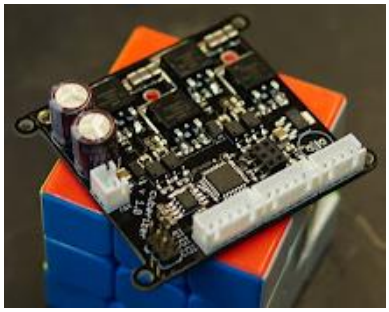

Ilustración 8. Controlador de Ben Katz.

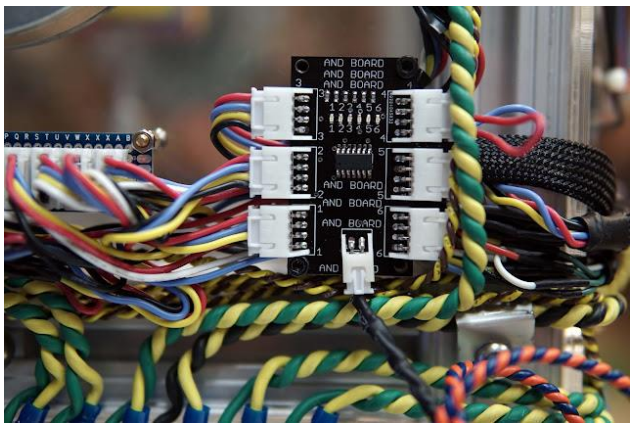

Ilustración 9. Placa de puertas AND para sincronización de los motores.

En este TFG, hemos optado por emplear un Arduino Uno, con un código desarrollado en C, en conjunto con drivers A4988, para recibir por puerto serie la cadena de caracteres que representa los giros de la solución calculada, y traducirla a las instrucciones necesarias para los motores. [6] [7] [8]

2.3.3 Algoritmo de Kociemba.

En ambos casos utilizaron para establecer sus respectivos récords, una implementación del algoritmo de Kociemba.

En este TFG, utilizamos el solucionador en dos fases publicado por el propio Herbert Kociemba en GitHub. [9] Utiliza Python para realizar los cálculos, y es capaz de dar soluciones pseudo-óptimas en fracciones de segundo. Hemos empleado una Raspberry Pi 3 Model B+. Para instalar el algoritmo, solo hay que introducir un comando en el terminal de Raspian:

```
$ pip install RubikTwoPhase
```

En el siguiente capítulo se detallarán las características de esta implementación.

2.3.4 Visión por computador.

Ambos récords emplean 2 cámaras web situadas en vértices opuestos para obtener el estado inicial del cubo, detectando los colores presentes en cada una de las caras, y traduciendo su localización a una cadena de caracteres que representa el estado del cubo.

Ilustración 10. Cámara web del robot de Ben y Jared.

Ilustración 11. Ejemplo de funcionamiento de la visión por computador del robot de Ben y Jared.

Dado que la visión por computador desempeña un papel pequeño dentro del sistema (exclusivamente comunicar al sistema que implementa el algoritmo de Kociemba el estado inicial), y sin embargo involucra múltiples problemas, incluyendo: la programación del script que analice la imagen, la iluminación del cubo de forma idéntica en cada ensayo y la configuración de múltiples cámaras en una misma Raspberry; tras varios intentos se ha decidido omitir este módulo en el TFG, optando por introducir manualmente el estado inicial del cubo usando la interfaz gráfica de usuario que incluye la implementación usada del algoritmo de Kociemba.

3 DESCRIPCIÓN DEL ROBOT

The most exciting phrase to hear in science, the one that heralds new discoveries, is not "Eureka!", but "That's funny…"

- Isaac Asimov -

En este capítulo se realizará una descripción detallada de los distintos componentes de STAR, así como de las decisiones relevantes tomadas durante el desarrollo del proyecto. Se explicará el funcionamiento y se indicarán los pasos necesarios para utilizar STAR.

3.1 Notación del cubo de Rubik.

En el mundillo de los aficionados del cubo de Rubik, se suele seguir la notación Singmaster, que al tener carácter relativo, permite escribir los algoritmos de manera que pueden ser aplicados independientemente de como se asignen las caras o del color que tengan. Se utiliza: U, D, L, R, F, B, que corresponden a Up (Cara superior), Down (Cara inferior), L, (Cara izquierda), R (Cara derecha), F (Cara frontal), B (Cara trasera). Adicionalmente, a la hora de describir los giros en las soluciones, se utilizan las letras mayúsculas para indicar un giro dextrógiro de 90º de la cara correspondiente. A la letra se le añade un símbolo de prima (′) para indicar un giro levógiro de 90º, y colocar un 2 detrás de la letra indica un giro de 180º. [10]

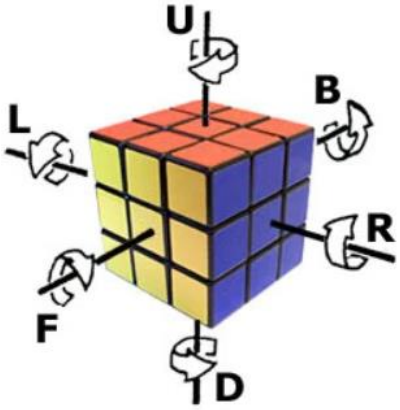

Ilustración 12. Representación de la notación Singmaster.

3.2 Software.

El software que controla el comportamiento del robot se divide en dos partes claramente diferenciadas:

3.2.1 Implementación del algoritmo de Kociemba.

Tras la instalación del GitHub de Herbert Kociemba, se descargaron en la Raspberry Pi las tablas que permiten el funcionamiento del algoritmo, siguiendo las instrucciones del GitHub [9]. Para utilizar la implementación, es recomendable iniciar un servidor local en un terminal en Python, con los comandos:

```
>>> import twophase.server as srv
>>> srv.start(8080, 20, 2)
```

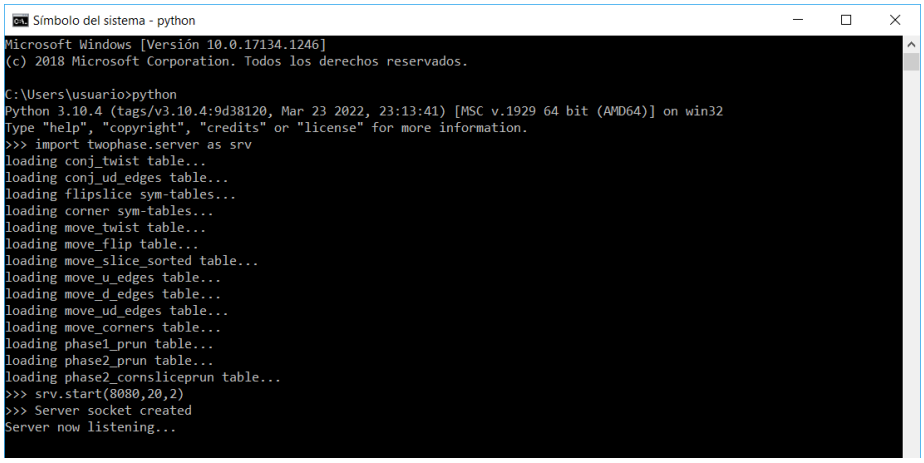

Ilustración 13. Ventana del terminal del servidor local.

Y luego abrir la interfaz gráfica de usuario que permite introducir cómodamente el estado inicial del cubo, con el comando:

```
>>> import twophase.client_gui
```

Ilustración 14. Ventana del terminal del cliente.

Usando la interfaz gráfica se puede introducir el estado inicial, y al pulsar el botón “Solve” se obtiene la solución del cubo en el cuadro de texto, donde se indica adicionalmente el número de movimientos, según la métrica de la media vuelta (HTM, por sus siglas en inglés). Se ha demostrado que cualquier estado inicial se puede resolver en un máximo de 20 movimientos según la HTM, o 26 según la QTM (Quarter Turn Metric).

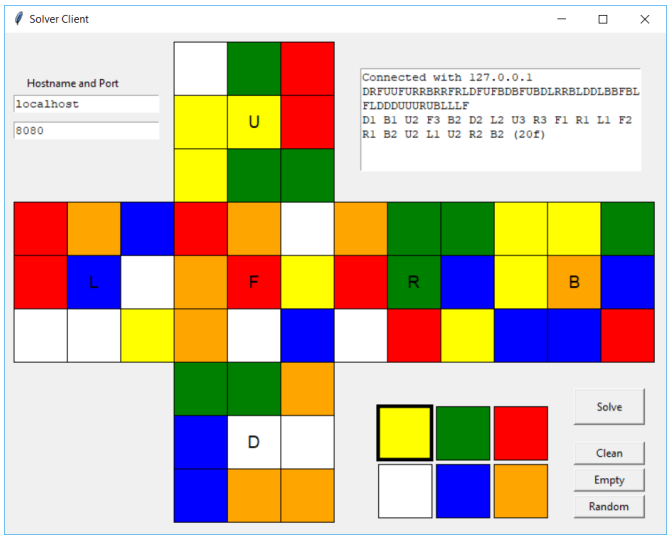

Ilustración 15. Ejemplo de uso de la interfaz de usuario.

En el caso de cometer un error al introducir el estado inicial, que hiciese que el cubo fuese irresoluble, porque el número o colocación de los cuadrados de colores fuese incorrecto, al pulsar Solve la interfaz señala que no es posible obtener una solución:

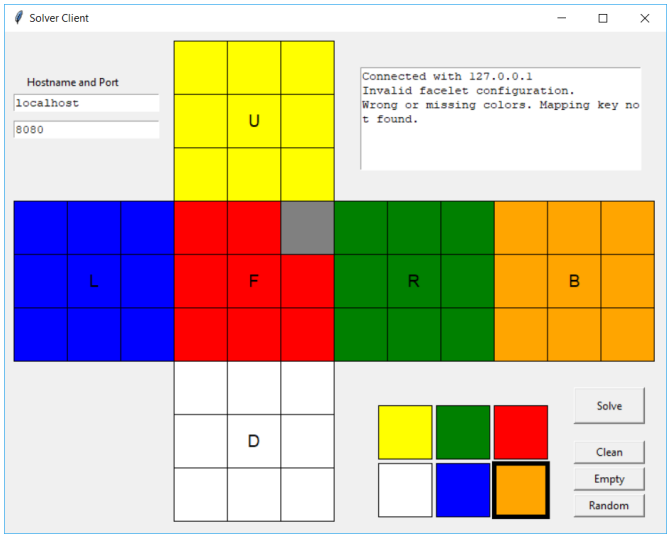

Ilustración 16. Error por estado inicial incompleto.

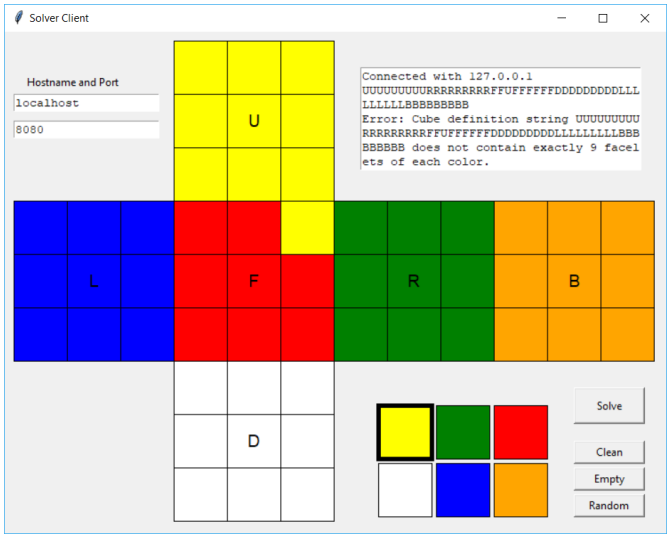

Ilustración 17. Error por número de caras de colores incorrecto.

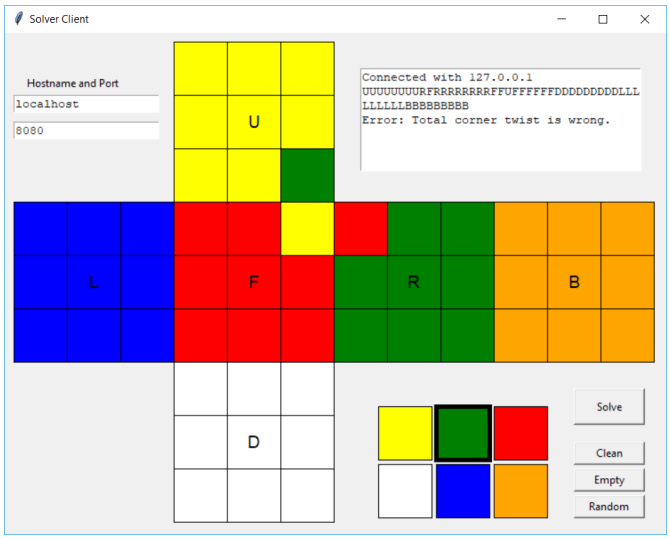

Ilustración 18. Error por estado irresoluble por colocación inalcanzable de un vértice.

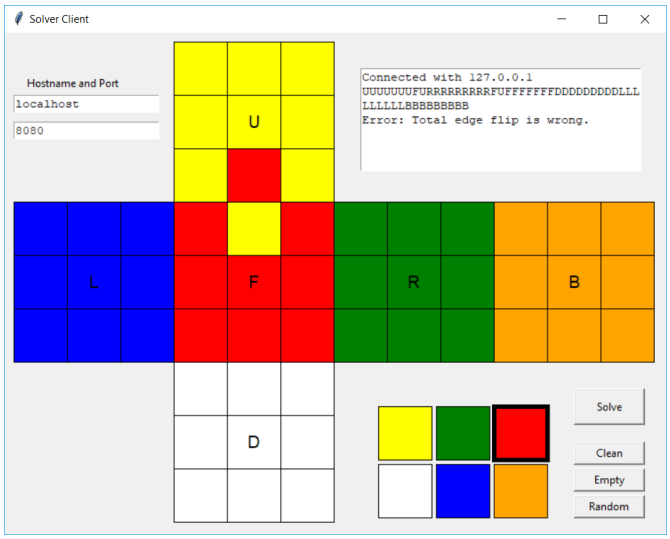

Ilustración 19. Error por estado irresoluble por colocación inalcanzable de arista.

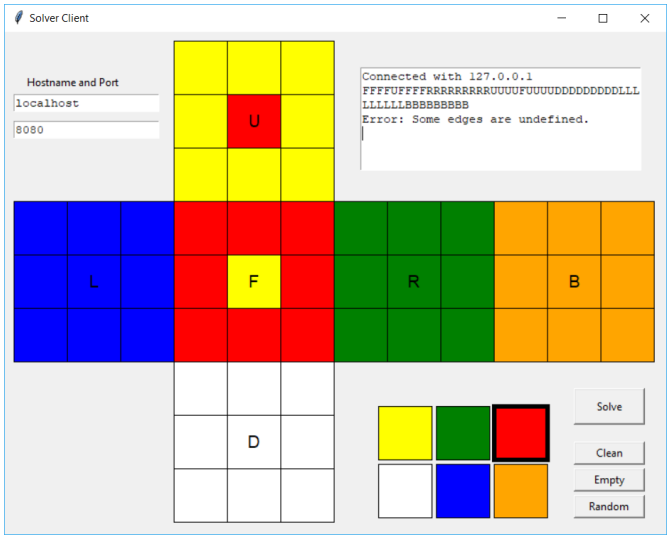

Ilustración 20. Error por definición incorrecta de las aristas.

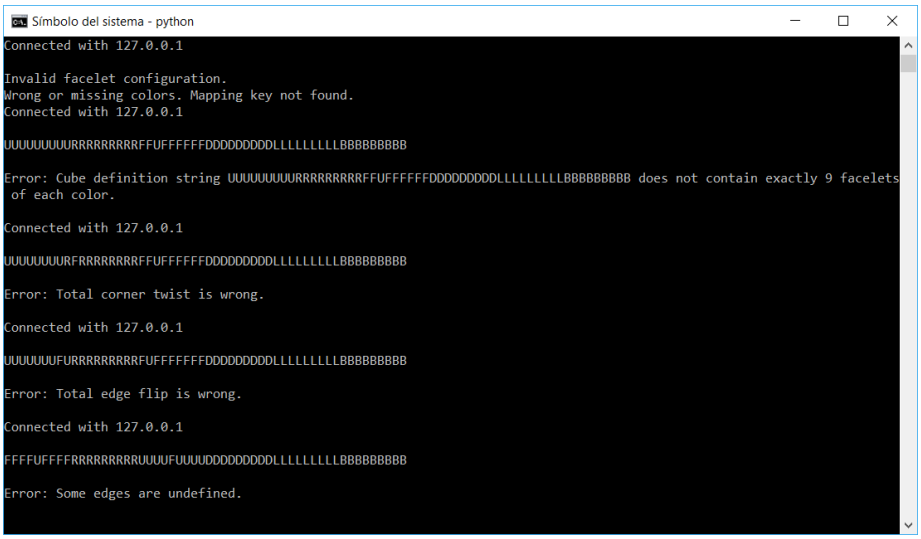

Ilustración 21. Errores, representados en el terminal del cliente.

3.2.2 Control de motores.

Se ha modificado un código en Arduino, para recibir por puerto serie la solución con el formato establecido, y traducir dicha solución a instrucciones para los motores. [7]

La implementación de Herbert Kociemba emplea la siguiente notación para expresar las soluciones: Una letra mayúscula (U,D,F,B,L,R) seguida de 1 para giros de 90º, 2 para giros de 180º, y 3 para giros de -90º. Tras cada instrucción incluye un espacio. El código que hemos desarrollado recorre el vector solución que se introduce por puerto serie, y al detectar una de las letras válidas, primero examina el número que la sigue, para determinar cual de los tres giros debe realizar, y luego da la orden de giro al motor correspondiente.

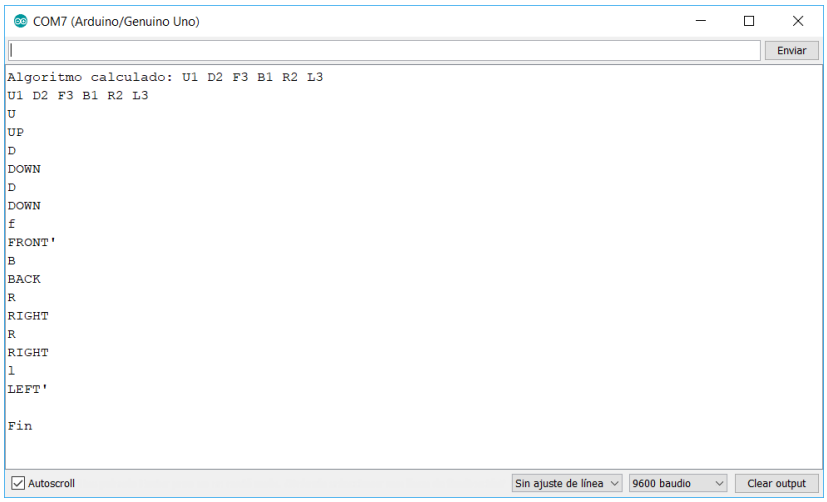

Ilustración 22. Ejemplo de funcionamiento del código de control de motores.

Tras introducir el vector solución, se muestra por pantalla primero la orden recibida (carácter en mayúscula para giro de dextrógiro, y en minúscula para levógiro) y luego tras realizar el giro, se imprime la palabra correspondiente, seguida de una prima en el caso de haber realizado un giro levógiro. En el código se distingue entre giros de 90º y -90º, en el caso de requerir uno de 180º se efectuan 2 giros de 90º seguidos.

El código ignora los caracteres que no se corresponden con un giro, como se muestra en la ilustración 23.

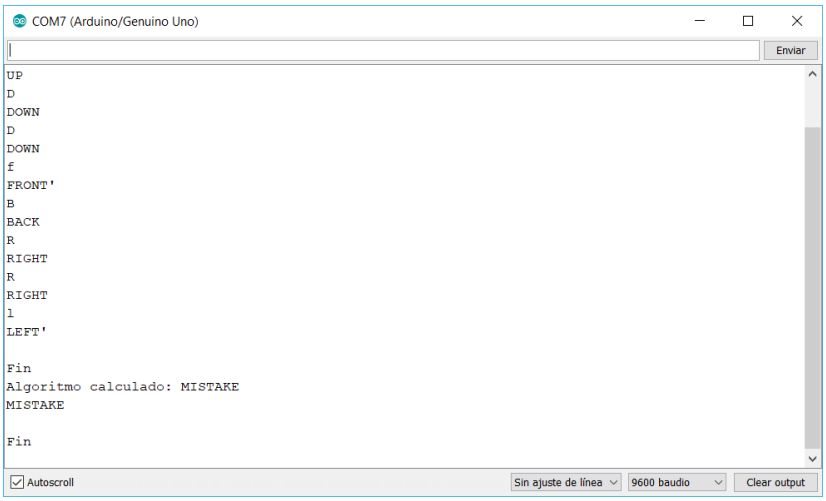

Ilustración 23. Comportamiento del código ante entrada errónea.

Sin embargo, en el caso de recibir un carácter admisible junto a otros erróneos, ejecuta los movimientos detectados, como se muestra en la ilustración 24.

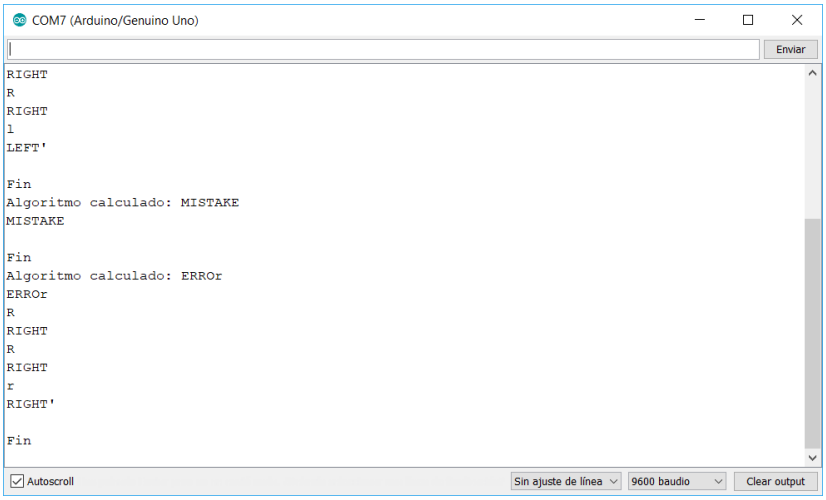

Ilustración 24. Comportamiento frente a entrada errónea que incluye caracteres admisibles.

Este comportamiento no supone un problema para la integridad del sistema.

3.3 Hardware.

Se han utilizado los siguientes elementos físicos:

- Cubo de Rubik marca QiYi.
- Raspberry Pi 3 Model B+.
- Periféricos (ratón, teclado, pantalla).
- Arduino Uno.
- Fuente de alimentación.
- Motores Nema17.
- Drivers A4988.
- Estructura de madera realizada por corte láser.
- Piezas personalizadas modeladas en Catia y fabricadas por impresión 3D.

3.3.1 Cubo de Rubik.

Se ha comprado un cubo económico, pero de respetable calidad, de la marca QiYi. Permite acceder a los tornillos que sujetan el núcleo a través de las tapas de las piezas centrales, y esto mismo permite sustituir las tapas, por las piezas diseñadas para enlazar las caras con los ejes de los motores. Cualquier cubo que permita realizar esta conexión de forma sencilla es adecuado para la construcción de un robot solucionador.

Ilustración 25. Cubo de la marca QiYi.

3.3.2 Raspberry Pi, periféricos.

Se ha escogido una Raspberry Pi ya que cumple con los requisitos mínimos para utilizar la implementación de Herbert Kociemba, y es una opción económica y muy bien documentada. Los periféricos utilizados han sido cedidos por el FABLAB de Sevilla, incluyendo ratón, teclado y minipantalla compatible con Raspberry Pi.

Ilustración 26. Periféricos utilizados.

3.3.3 Arduino Uno.

Aunque inicialmente se escogió un Arduino Mega, tras realizar el conexionado se observó que los pines digitales del Arduino Uno eran suficientes para controlar los motores gracias a los drivers A4988.

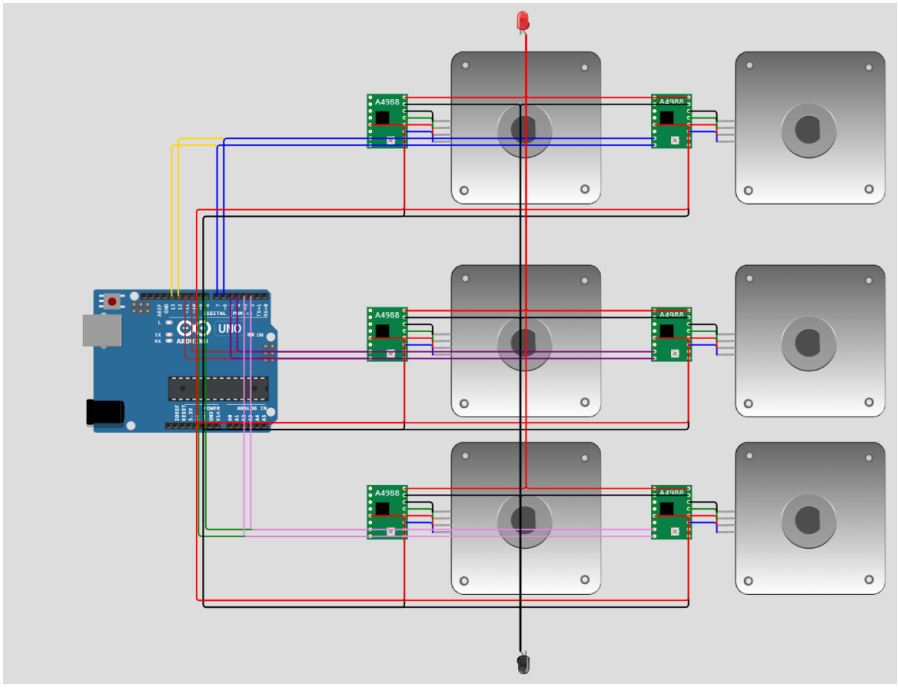

Ilustración 27. Esquema de conexiones del Arduino Uno.

Se emplean los pines digitales del Arduino, del 2 al 13 para enviar las señales de control a los drivers. Adicionalmente se conecta el pin de 5V con los Vdd de los drivers, y la tierra del arduino con el pin destinado a la tierra del microcontrolador en los drivers. Se alimenta el Arduino a través de USB.

3.3.4 Fuente de alimentación.

Los motores requieren 12V, la Raspberry Pi requiere 5V y 2.5A, y el Arduino Uno se puede alimentar por USB. Por comodidad, se ha empleado una fuente programable para alimentar tanto los motores como la Raspberry Pi.

3.3.5 Motores, drivers.

Se han reutilizado los motores Nema 17, así como los A4988, que pertenecían a una antigua impresora 3D. Los drivers tienen dos entradas de control, el pin DIR selecciona el sentido de giro, y el pin STEP, que al recibir un pulso provoca un paso en el motor. Para lograr un cuarto de giro, ya que cada paso se correponde con 1.8º, deben darse 50 pasos. Se ha logrado esto por software, empleando bucles for que envían los 50 pulsos necesarios a los pines STEP tras recibir la orden de realizar el correspondiente giro.

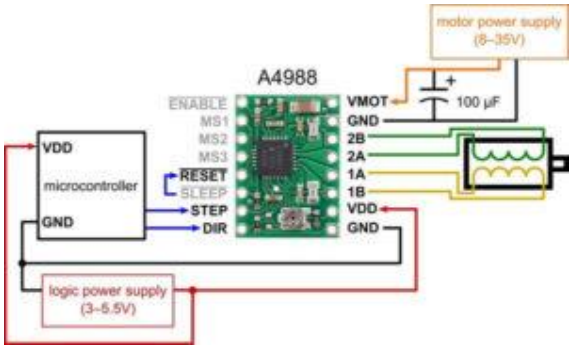

Ilustración 28. Esquema de conexión de los drivers y motores.

Los pines DIR y STEP se conectan con las salidas digitales del Arduino Uno, VMOT con el terminal positivo de la fuente de alimentación, y GND con su tierra. Se ha empleado un condensador entre ambos pines, para lidiar con los picos de corriente. [8]

3.3.6 Estructura de madera.

Se ha empleado una cortadora láser para crear la estructura que se utiliza en el proyecto. Tras tomar las medidas necesarias, se diseñó la estructura teniendo en cuenta el ancho de la placa a cortar. Al terminar el archivo .svg, se introduce en la cortadora, y obtenemos las piezas de la estructura deseada.[3]

Ilustración 29. Imagen de la estructura final.

Antes de disponer de la estructura cortada a láser, se hicieron pruebas y se tomaron medidas con una estructura temporal.

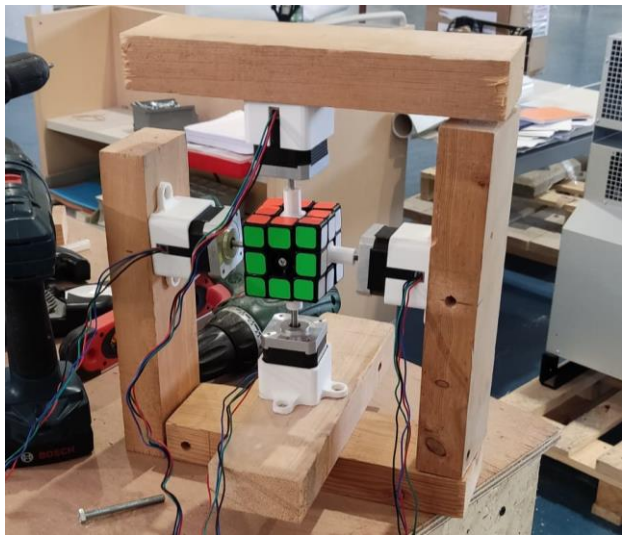

Ilustración 30. Imagen de la estructura temporal.

3.3.7 Piezas 3D.

Uno de los primeros problemas que hubo que solucionar al comienzo del proyecto, fue como enlazar las caras del cubo a los ejes de los motores. Con ese propósito se diseñaron unas agarraderas a medida que encajan en el hueco que queda al levantar las tapas de las piezas centrales, y conectan con el eje de los motores, permitiendo el giro solidario de las caras con los ejes.

[3] https://youtu.be/r8LHQQdPwh8 Vídeo que muestra la cortadora láser en funcionamiento.

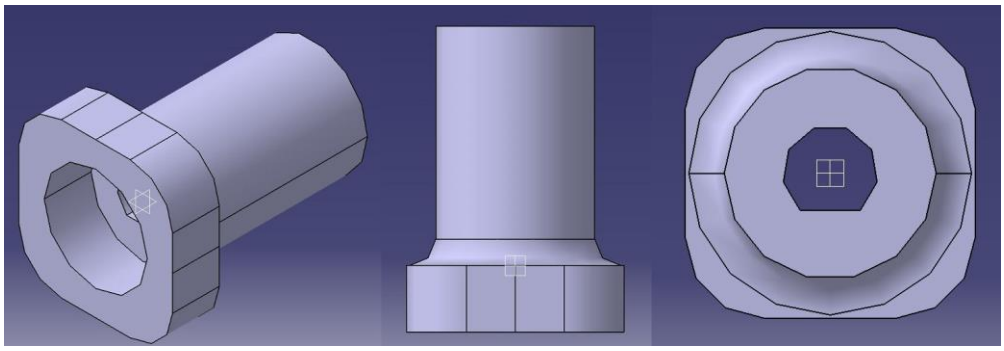

Ilustración 31. Modelo 3D de las agarraderas diseñadas.

Adicionalmente, se diseñaron unos enganches para asegurar los motores a la estructura de madera, empleando tornillos y tuercas.

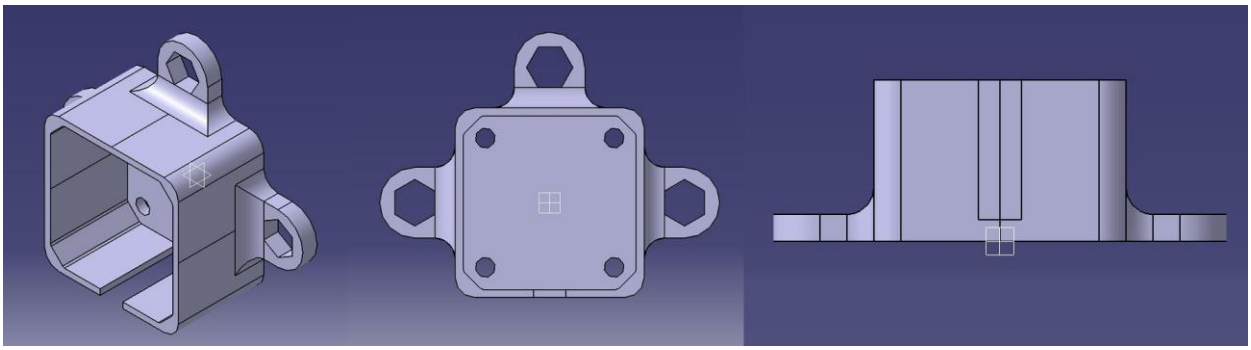

Ilustración 32. Modelo 3D de los enganches de los motores.

Por último, se creó un soporte para el motor de la cara superior, para asegurarlo a la estructura y evitar que su peso recayese sobre el cubo.

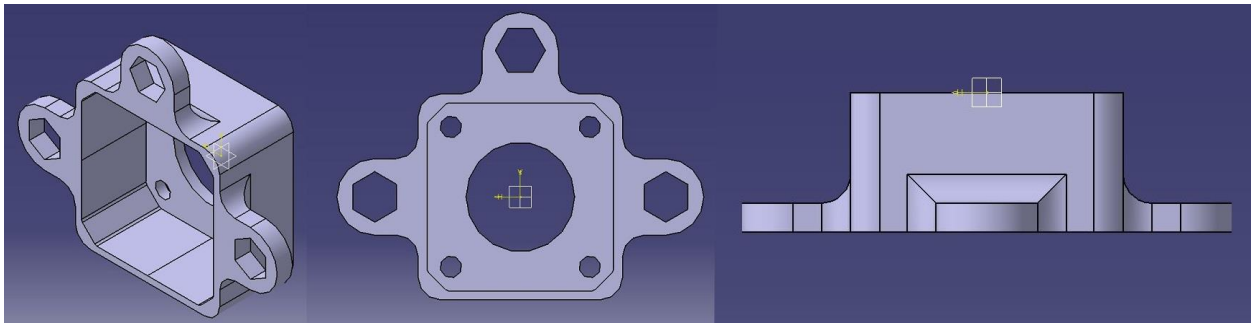

Ilustración 33. Modelo 3D del soporte para el motor superior.

3.4 Funcionamiento.

Tras alimentar las placas y los motores, el primer paso para usar STAR es utilizar el monitor serie del Arduino para mezclar el cubo. Se pueden introducir movimientos aleatorios, o seguir los dictados por el programa oficial de mezclado de la WCA. Una vez se ha llegado a una mezcla aceptable, se debe utilizar la interfaz de usuario de la implementación de Kociemba para comunicar el estado inicial del cubo. Tras pulsar el botón “Solve” se obtienen los movimientos a realizar. Esta solución se introduce en el monitor serie del Arduino Uno, y tras enviarlo, S.T.A.R. comienza a mover las caras automáticamente.

Debido a la naturaleza acumulativa y catastrofica de los errores de posición de las caras del cubo en la ejecución, es aconsejable agrupar los giros de la solución en grupos de 3, y comprobar que los giros se ejecutan correctamente antes de enviar el siguiente grupo al monitor serie.

4 RESULTADOS

Part of the inhumanity of the computer is that, once it is competently programmed and working smoothly, it is completely honest.

- Isaac Asimov -

Volviendo a los objetivos señalados al principio de esta memoria, procederemos a comprobar si S.T.A.R. cumple con ellos. Además analizaremos lo puntos que permitirían mejorar dichos objetivos.

4.1 Tiempo de resolución.

Se ha logrado solucionar el cubo en menos de un minuto.[4] Para lograr un tiempo más rápido, la construcción de la estructura, el cubo escogido, y, el hecho de que la posición de los motores se controla con un lazo abierto, deben ser corregidos, lo que permitirá reducir el tiempo de espera en el código entre las ejecuciones de los giros.

4.2 Presupuesto.

El coste total del trabajo se ha visto reducido enormemente gracias a disponer de distintos préstamos. Se va a realizar una estimación del coste para el público, empleando materiales de tiendas online.

Objeto	Precio estimado
Cubo QiYi	5€
Raspberry Pi	Entre 35€ y 65€
Monitor/Ratón/Teclado/Cable HDMI	25€/5€/15€/3€
Arduino Uno	24€
Motores Nema 17	8€ c/u

[4] https://youtu.be/YaJpTkkor5Q Video de resolución. El tiempo total de movimiento de STAR es muy inferior al minuto.

Drivers A4988	1.7€ c/u
Corte estructura madera (1.2€/minuto, mínimo 25€)	25€ [11]
Impresión piezas 3D (aproximadamente 4h)	28€ [11]

Estimamos que el coste mínimo puede variar entre 220€ y 250€, dependiendo de la disponibilidad de los materiales, la urgencia con la que se busquen, y el acceso a tiendas que permitan realizar el corte láser e impresión 3D.

El coste de fabricación de nuestro prototipo ha sido de 0.61€, correspondientes exclusivamente al coste del cubo, adquirido por AliExpress e importado desde China. El resto de materiales han sido cedidos.

4.3 Conocimientos integrados.

Se han utilizado conocimientos de electrónica, programación en C, uso de GitHub y diseño en 3D. No solo se han aplicado conocimientos previos, sino que se han descubierto nuevas herramientas que se emplearán en futuros protectos.

5 AMPLIACIONES

Education is not something you can finish.

- Isaac Asimov -

En el transcurso del proyecto se ha decidido dejar de lado algunos aspectos, por complejidad o falta de tiempo. En este capítulo se señalan los esfuerzos realizados para que puedan ser considerados en el caso de querer ampliar el trabajo realizado.

5.1 Visión por computador.

Se ha intentado repetidas veces hacer funcionar el módulo de visión por computador que incluye la implementación en GitHub del algoritmo de Kociemba, sin embargo no se han logrado resultados consistentes, a pesar de realizar múltiples pruebas bajo una iluminación constante proporcionada por una lámpara LED de color blanco, y a pesar de haber modificado, siguiendo las indicaciones presentadas en el GitHub, los parámetros que controlan el módulo en la interfaz que lo acompaña.

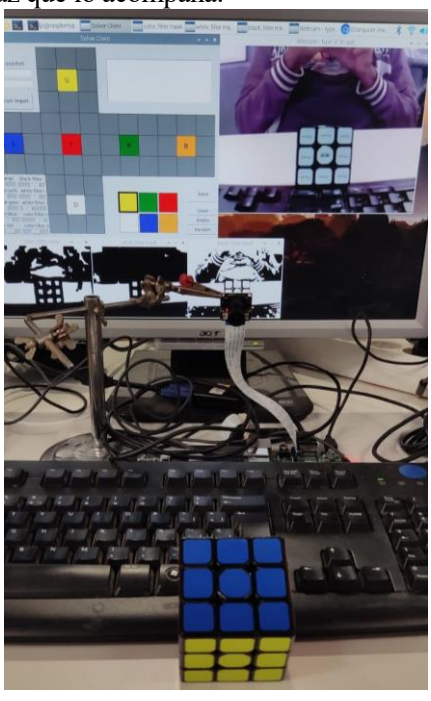

Ilustración 34. Prueba de funcionamiento de la visión por computador.

Uno de los factores que ha influido en el mal resultado de la visión por computador, puede haber sido el uso de una cámara inadecuada. Se realizaron las pruebas con un módulo de cámara de bajo coste, compatible con la Raspberry Pi. Otro factor que se ha observado, es que las pegatinas del cubo comprado reflejan bastante luz, provocando errores en el módulo de visión. Se ha considerado la opción de crear un sistema propio, basado en el análisis de imágenes por MATLAB, pero finalmente, se ha decidido omitir el módulo por completo.

5.2 Conexión por puerto serie.

Para realizar la conexión de la Raspberry Pi con el Arduino Uno, se han considerado diversas opciones. Desde emplear el programa PuTTY tras instalarlo en la Raspberry Pi, hasta hacer una conexión directa de los pines de ambas placas y crear un script que envíe la solución a través de comandos en un terminal en Raspian.

Finalmente, ya que se puede instalar la IDE de Arduino en la propia Raspberry Pi, se ha optado por omitir la conexión automática, en favor de introducir la solución a través del monitor serie de Arduino.

5.3 Observaciones adicionales.

El tiempo logrado en el trabajo es mucho mayor al mínimo teórico alcanzable por STAR. Sin embargo la calibración de los parámetros es un trabajo tedioso y los errores de los motores y de alineación del cubo en la ejecución de la solución se acumulan hasta provocar que se bloquee STAR o que se salte uno de los giros de la solución, haciendo que el resultado final no sea el cubo resuelto.[5] Revisando la estructura, y probando con distintos niveles de soltura del núcleo del cubo, debería ser posible lograr una mayor consistencia en la ejecución. Emplear un cubo de mayor precio, que incluya imanes de neodimio para facilitar la alineación, podría contribuir a evitar los errores en la resolución. Adicionalmente, el empleo de un bucle de control cerrado con motores paso a paso asociados a encóders, u otro tipo de tecnología que permita eliminar el error de la posición de los motores en régimen permanente, permitiría reducir o incluso eliminar por completo los fallos descritos.

[5] https://youtube.com/shorts/aBJOcr5KNSQ?feature=share Vídeo de fallo del robot.

6 REFERENCIAS

[1] Wikipedia, «Rubik's Cube,» 2022. [En línea]. Available: https://en.wikipedia.org/wiki/Rubik%27s_Cube. [Último acceso: 12 Septiembre 2022].

[2] Wikipedia, «Optimal solutions for Rubik's Cube,» 2022. [En línea]. Available: https://en.wikipedia.org/wiki/Optimal_solutions_for_Rubik%27s_Cube. [Último acceso: 12 Septiembre 2022].

[3] R. Swatman, «Guiness World Records,» 3 Marzo 2017. [En línea]. Available: https://www.guinnessworldrecords.com/news/2017/3/video-robot-breaks-world-record-solving-rubiks-cube-in-0-637-seconds-464392. [Último acceso: 12 Septiembre 2022].

[4] B. Katz, «BuildIts,» 7 Marzo 2018. [En línea]. Available: https://build-its-inprogress.blogspot.com/2018/03/the-rubiks-contraption.html. [Último acceso: 12 Septiembre 2022].

[5] J. Di Carlo, «The Cactus Zone,» 7 Marzo 2018. [En línea]. Available: http://cactus-zone.blogspot.com/2018/03/rubiks-solver-software.html. [Último acceso: 12 Septiembre 2022].

[6] Arduino, «Language Reference,» 2022. [En línea]. Available: https://www.arduino.cc/reference/en/. [Último acceso: 12 Septiembre 2022].

[7] carlini, «Carlini's Blog,» 20 Enero 2017. [En línea]. Available: http://carlini.es/manejar-un-motor-stepper-con-un-driver-drv8825-y-arduino/. [Último acceso: 12 Septiembre 2022].

[8] PololuCorportation, «Pololu.com,» 2019. [En línea]. Available: https://www.pololu.com/docs/pdf/0J16/destructive_LC_voltage_spikes.pdf. [Último acceso: 12 Septiembre 2022].

[9] H. Kociemba, «GitHub,» 27 Abril 2017. [En línea]. Available: https://github.com/hkociemba/RubiksCube-TwophaseSolver. [Último acceso: 12 Septiembre 2022].

[10] WikiCube, «Notation,» 2020. [En línea]. Available: https://rubiks.fandom.com/wiki/Notation. [Último acceso: 12 Septiembre 2022].

[11] Atta33, «Tarifas,» 2021. [En línea]. Available: https://atta33.com/servicios-tarifas/. [Último acceso: 12 Septiembre 2022].

CÓDIGO

```
//CÓDIGO DESARROLLADO PARA EL CONTROL DE LOS MOTORES.
// Declaración de variables
const int dirU  = 13; //Pines para controlar la rotación de las caras. dirX controla la dirección de rotación de la cara X, y stepX da la señal de avance.
const int stepU = 12;
const int dirD  = 11;
const int stepD = 10;
const int dirF  = 9;
const int stepF = 8;
const int dirB  = 7;
const int stepB = 6;
const int dirL  = 5;
const int stepL = 4;
const int dirR  = 3;
const int stepR = 2;

const int steps = 50; // Pasos equivalentes a un cuarto de giro. Los A4988 están configurados para que cada paso avance 1.8º, luego 50 pasos son 90º.

int microPausa = 1000, j = 0;
String sol,prev;
char mov = 'I';
int len = 0;
int pausa = 300;      // Duración en milisegundos de la pausa después de cada movimiento.

// Inicialización
void setup() {
  Serial.begin(9600);
  pinMode(dirU, OUTPUT);
  pinMode(stepU, OUTPUT);
  pinMode(dirD, OUTPUT);
  pinMode(stepD, OUTPUT);
  pinMode(dirF, OUTPUT);
  pinMode(stepF, OUTPUT);
  pinMode(dirB, OUTPUT);
  pinMode(stepB, OUTPUT);
```

```
  pinMode(dirL, OUTPUT);
  pinMode(stepL, OUTPUT);
  pinMode(dirR, OUTPUT);
  pinMode(stepR, OUTPUT);
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  while(!Serial);
}

// Funciones creadas para cada posible instrucción recibida por puerto serie.
void UP(){
  digitalWrite(dirU, HIGH);  // Establezco una dirección
  for (int x = 0; x < steps ; x++) {
    digitalWrite(stepU, HIGH);
    delayMicroseconds(microPausa);
    digitalWrite(stepU, LOW);
    delayMicroseconds(microPausa);
  }
  Serial.println("UP");
  delay(pausa);
}

void DOWN(){
  digitalWrite(dirD, HIGH);  // Establezco una dirección
  for (int x = 0; x < steps ; x++) {
    digitalWrite(stepD, HIGH);
    delayMicroseconds(microPausa);
    digitalWrite(stepD, LOW);
    delayMicroseconds(microPausa);
  }
  Serial.println("DOWN");
  delay(pausa);
}

void FRONT(){
  digitalWrite(dirF, HIGH);  // Establezco una dirección
  for (int x = 0; x < steps ; x++) {
    digitalWrite(stepF, HIGH);
    delayMicroseconds(microPausa);
    digitalWrite(stepF, LOW);
```

```
    delayMicroseconds(microPausa);
  }
  Serial.println("FRONT");
  delay(pausa);
}

void BACK(){
  digitalWrite(dirB, HIGH);  // Establezco una dirección
  for (int x = 0; x < steps ; x++) {
    digitalWrite(stepB, HIGH);
    delayMicroseconds(microPausa);
    digitalWrite(stepB, LOW);
    delayMicroseconds(microPausa);
  }
  Serial.println("BACK");
  delay(pausa);
}

void LEFT(){
  digitalWrite(dirL, HIGH);  // Establezco una dirección
  for (int x = 0; x < steps ; x++) {
    digitalWrite(stepL, HIGH);
    delayMicroseconds(microPausa);
    digitalWrite(stepL, LOW);
    delayMicroseconds(microPausa);
  }
  Serial.println("LEFT");
  delay(pausa);
}

void RIGHT(){
  digitalWrite(dirR, HIGH);  // Establezco una dirección
  for (int x = 0; x < steps ; x++) {
    digitalWrite(stepR, HIGH);
    delayMicroseconds(microPausa);
    digitalWrite(stepR, LOW);
    delayMicroseconds(microPausa);
  }
  Serial.println("RIGHT");
  delay(pausa);
```

```
}

void UP_(){
  digitalWrite(dirU, LOW);  // Establezco una dirección
  for (int x = 0; x < steps ; x++) {
    digitalWrite(stepU, HIGH);
    delayMicroseconds(microPausa);
    digitalWrite(stepU, LOW);
    delayMicroseconds(microPausa);
  }
  Serial.println("UP'");
  delay(pausa);
}

void DOWN_(){
  digitalWrite(dirD, LOW);  // Establezco una dirección
  for (int x = 0; x < steps ; x++) {
    digitalWrite(stepD, HIGH);
    delayMicroseconds(microPausa);
    digitalWrite(stepD, LOW);
    delayMicroseconds(microPausa);
  }
  Serial.println("DOWN'");
  delay(pausa);
}

void FRONT_(){
  digitalWrite(dirF, LOW);  // Establezco una dirección
  for (int x = 0; x < steps ; x++) {
    digitalWrite(stepF, HIGH);
    delayMicroseconds(microPausa);
    digitalWrite(stepF, LOW);
    delayMicroseconds(microPausa);
  }
  Serial.println("FRONT'");
  delay(pausa);
}

void BACK_(){
  digitalWrite(dirB, LOW);  // Establezco una dirección
```

```
  for (int x = 0; x < steps ; x++) {
    digitalWrite(stepB, HIGH);
    delayMicroseconds(microPausa);
    digitalWrite(stepB, LOW);
    delayMicroseconds(microPausa);
  }
  Serial.println("BACK'");
  delay(pausa);
}

void LEFT_(){
  digitalWrite(dirL, LOW);  // Establezco una dirección
  for (int x = 0; x < steps ; x++) {
    digitalWrite(stepL, HIGH);
    delayMicroseconds(microPausa);
    digitalWrite(stepL, LOW);
    delayMicroseconds(microPausa);
  }
  Serial.println("LEFT'");
  delay(pausa);
}

void RIGHT_(){
  digitalWrite(dirR, LOW);  // Establezco una dirección
  for (int x = 0; x < steps ; x++) {
    digitalWrite(stepR, HIGH);
    delayMicroseconds(microPausa);
    digitalWrite(stepR, LOW);
    delayMicroseconds(microPausa);
  }
  Serial.println("RIGHT'");
  delay(pausa);
}

//Bucle principal
void loop() {
  if(Serial.available()>0){     // Lectura de solución por puerto serie
    sol = Serial.readString();
    Serial.print("Algoritmo calculado: ");
    Serial.println(sol);
```

```
  }
  if(sol != prev){
    len = sol.length();
    Serial.println(sol);
    /*Serial.println(len);*/  // Longitud de vector solución, solo para debugging
    for(j=0;j<=len;j++){    // Bucle que recorre el vector de la solución, y lo modifica para aplicar los giros adecuados.
      mov = sol[j];
      if(sol[j+1] == 51){        // SI LUEGO VIENE 3, modifico el caracter actual y salto el 3.
        mov = mov-('A'-'a');
        j = j+1;
      }
      else if(sol[j+1] == 50){    // SI LUEGO VIENE 2, lo cambio por una rotación de la misma cara en el mismo sentido.
        sol[j+1] = sol[j];
      }
      else if(sol[j+1] == 49){    // SI LUEGO VIENE 1, lo salto.
        j = j+1;
      }

if(mov=='U'||mov=='D'||mov=='F'||mov=='B'||mov=='L'||mov=='R'||mov=='u'||mov=='d'||mov=='f'||mov=='b'||mov=='l'||mov=='r'||mov=='\0'){               // Imprimo por pantalla el movimiento a realizar.
        Serial.println(mov);
        switch (mov){
          case '\0':
          Serial.println("Fin");
          sol = "";
          break;
          case 'U':
            UP();
            break;
          case 'u':
            UP_();
            break;
          case 'D':
            DOWN();
            break;
          case 'd':
            DOWN_();
            break;
```

```
          case 'F':
            FRONT();
            break;
          case 'f':
            FRONT_();
            break;
          case 'B':
            BACK();
            break;
          case 'b':
            BACK_();
            break;
          case 'L':
            LEFT();
            break;
          case 'l':
            LEFT_();
            break;
          case 'R':
            RIGHT();
            break;
          case 'r':
            RIGHT_();
            break;
          /*case ' ':       //Casos exclusivos para debugging
            Serial.println("ESPACIO");
            break;
          default:
            Serial.println("Error");
          break;*/
        }
      }
      prev = sol;
    }
    delay(50);
  }
}
```