Universidad de Concepción

# RESOLVIENDO EL CUBO DE RUBIK CON EL ROBOT BAXTER

**POR CÉSAR ANDRÉS BOLÍVAR SEVERINO**
Departamento de Ingeniería Informática y Ciencias de la Computación
Universidad de Concepción

Tesis presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Informático

**Profesor Guía: Julio Godoy del Campo**
**Comisión: Roberto Asín Achá, Eduardo Méndez Ortiz**

3 de septiembre de 2019

Concepción, Chile

A mi madre, y a mi padre, que en paz descanse.

# AGRADECIMIENTOS

Mirando hacia atrás, siento que toda mi vida he sido un malagradecido. Aunque me encantaría poder decir que todo mis logros son productos de mi propio esfuerzo y de nadie más, la verdad es que muchos si es que no todos hubiesen sido imposibles de realizar sin el apoyo de mis seres queridos. Ha llegado la hora de dar las gracias, así que leed bien los siguientes párrafos que no los escribiré dos veces.

Gracias a mi familia por su amor incondicional, por creer en mí, por apoyar cada una de las decisiones que me llevaron a estudiar en esta prestigiosa universidad y por su apoyo financiero durante tantos años, a pesar de las adversidades. Sin su esfuerzo jamás hubiese llegado donde estoy ahora. Espero algún día poder retribuirles todo.

Gracias a mis amigos, que a pesar de la distancia que nos separa me han acompañado año tras año. Gracias por su lealtad y por sacarme sonrisas hasta en los momentos más difíciles. Ya ansío poder celebrar con ustedes.

Gracias a todos los profesores del Departamento de Ingeniería Informática y Ciencias de la Computación de la Universidad de Concepción que tuvieron parte en mi proceso formativo, en especial al profesor Roberto Asín por su buena disposición y la ayuda que me brindó en multitud de ocasiones. Son todos personas muy inteligentes, amables e inspiradoras.

Gracias a mis compañeros por sufrir la carrera junto conmigo, sacando adelante cada proyecto realizado en estos durísimos años. En especial, gracias a Daniel Ortega, que más que un compañero y un amigo ha sido un hermano para mí.

Gracias a todas las personas que han formado parte de este viaje y que aportaron de una u otra forma a que llegase este momento. Los amo a todos.

# Índice general

# Índice de cuadros

# Índice de figuras

# Índice de figuras

# Resumen

Este documento describe un proyecto de Memoria de Título de Ingeniería Civil Informática, el cual consistió en lograr que Baxter, un robot manipulador industrial humanoide, resuelva el cubo de Rubik, un puzzle mecánico tridimensional. Se describe la implementación de un sistema que realiza esta tarea, desde el recoger el cubo de una mesa, pasando por la detección del estado del cubo, la búsqueda de una solución para el mismo, hasta la manipulación y ejecución de dicha solución. Se propone un esquema de 6 posturas de los brazos del robot para la captura del estado del cubo mediante una cámara externa, y otro también de 6 posturas para la ejecución de rotaciones de caras del cubo. Para la visión, se propone utilizar la transformada de Hough y un modelo de mezcla de gaussianas, para detectar regiones de interés, extracción de colores representantes de dicha región y agrupamiento de los mismos. Para la resolución, se utiliza el algoritmo de 2 fases de Kociemba. El sistema implementado es capaz de detectar el estado de un cubo de Rubik con una exactitud de sobre el 90 % del estado total, y es capaz de ejecutar correctamente secuencias no muy largas de movimientos de caras del cubo. Se describen situaciones problemáticas que se presentan al intentar completar el objetivo, y sus posibles causas.

# 1 Introducción

El cubo de Rubik es un puzzle mecánico, inventado por Erno Rubik en 1974 [13]. Consiste de un sistema de piezas móviles formadas al subdividir cada una de las caras de un cubo en una grilla de $3 \times 3$ cuadrados de igual tamaño, permitiendo que aristas y esquinas se permuten entre sí al rotar alguna de las caras en torno a un eje que atraviesa el centro de dicha cara. Cada cara tiene un color diferente a las demás, y el objetivo del rompecabezas es, una vez permutado el cubo, llevarlo a una configuración donde todos los cuadrados (también llamados "*facelets*") de una misma cara tengan el mismo color. Ésto debe conseguirse sólo mediante rotaciones de caras sobre sus ejes y sin algún tipo de trampa como destornillar los ejes o sacar los stickers de los facelets.

A pesar de su aparente simplicidad, este juguete oculta una complejidad que ha fascinado a científicos de la computación por décadas. Con nada más ni nada menos que 43 252 003 274 489 856 000 configuraciones posibles[9], la tarea de resolverlo no es para nada trivial. Varios algoritmos de diversa complejidad han sido desarrollados para solucionar el cubo, cada uno con ventajas y desventajas.

El interés en resolver el cubo de Rubik no se detiene en el ámbito del software. Numerosos robots se han diseñado específicamente para resolver cubos, pero también se han utilizado robots con propósitos más generales para esta tarea. Uno de estos robots, es el robot Baxter.

Baxter es un robot humanoide, desarrollado por la compañía Rethink Robotics[11]. Mide entre 122 y 190 centímetros de alto, y pesa 138 kilogramos. Posee 2 brazos con 7 articulaciones cada uno, 4 de tipo rotatoria y 3 "codos" intercaladas entre sí. Un par de pinzas paralelas o "*grippers*" se pueden adjuntar en los extremos de cada brazo. Estas pinzas, disponibles en diversas formas y tamaños, pueden abrirse o cerrarse con el propósito de manipular una variedad objetos. El robot también posee sensores para determinar la posición y torque

Figura 1.1: El robot Baxter.

de cada articulación en cada una de éstas. Además, cuenta con cámaras en sus 2 brazos y en su cabeza. La cabeza de Baxter contiene una pantalla, en la cual se puede mostrar información relevante[35]. Este robot ha sido utilizado ampliamente tanto en la industria como en la academia, realizando tareas desde manipular objetos en líneas de ensamblaje[5], pasando por jugar ajedrez[6] e incluso asistir a humanos en ejercicios diseñados para terapia física[10].

El objetivo general de este trabajo es conseguir que un robot Baxter arme un cubo de Rubik desde cero. Esto incluye los objetivos particulares de (1) ser capaz de recoger el cubo desde algún lugar, (2) utilizar sensores (en particular, cámaras) para detectar el estado del cubo, (3) solucionarlo, (4) planificar una secuencia de movimientos y (5) efectuarlos mediante el hardware del robot. Para esto se emplea el Robot Baxter ubicado en el Laboratorio FabLab de la Facultad de Ingeniería de la Universidad de Concepción, y un cubo de Rubik capaz de "cortar esquinas" (esto es, que sus caras puedan rotar aún cuando no estén perfectamente alineadas) y ligeramente preparado para su uso por el

robot.

Como objetivos específicos se tiene:

- Aplicar técnicas de robótica para programar un robot para una tarea que requiere motricidad fina.
- Aplicar técnicas de visión computacional para detectar el estado de un cubo de Rubik.
- Utilizar métodos de inteligencia artificial en la resolución de un problema combinatorio.
- Diseñar e implementar un sistema que integre todas las tareas anteriores.

Al finalizar este proyecto, se consiguió que el robot Baxter detectara correctamente más del 90 % del total de los colores del cubo sobre todos los experimentos realizados, y más de la mitad de cada experimento individual en su totalidad. El robot es capaz de llevar el cubo desde su estado inicial hasta su estado resuelto realizando la secuencia de giros correcta completamente, siempre y cuando dicha secuencia no sea demasiado larga.

# 2 Discusión Bibliográfica

## Rubik

Un cubo de Rubik válidamente permutado tiene una solución dada por una secuencia de rotaciones de caras que, al aplicarlas en orden, llevan el cubo a su estado resuelto. Usualmente, se utilizan 2 métricas para contar la cantidad de rotaciones: *quarter-turn metric* o "QTM" y *half-turn metric* o "HTM"[36]. La primera considera cada rotación de 90 grados (ya sea en sentido horario o antihorario) como una acción por sí misma. La segunda es similar, salvo que además considera los giros de 180 grados como 1 acción en lugar de 2.

A lo largo de los años, se ha ido reduciendo la cota superior del largo máximo de una solución para cualquier configuración del cubo. A este valor se le conoce como el "número de Dios", en alusión a el "algoritmo de Dios", un algoritmo que sería capaz de resolver cualquier cubo de Rubik en la menor cantidad de movimientos.

En 1981, Thistlethwaite[45] demuestra que 52 movimientos bastan para resolver cualquier cubo. Posteriormente, Kloosterman[15] reduce esta cota a 42 en 1990. En 1992, Reid y Winter disminuyen este valor a 39 y 37 movimientos, transcurriendo sólamente 1 día entre estos dos eventos[30][47]. En 1995, Reid baja aún mas la cota a 29 giros [31].

En 1995, Reid demuestra que la posición "superflip", una configuración en la que el cubo está resuelto, con la excepción de que todas las aristas están volteadas en su lugar, requiere de 20 rotaciones[32] en HTM, dando una cota inferior para el largo de una solución en el peor caso.

La cota superior continuó descendiendo, demostrándose de 28 en y 27 por Radu[29][28] en 2005 y 2006 respectivamente, 25 por Kunkle[21] en 2007 y 25, 23 y luego 22 por Rokicki[22][37][38] et al en 2008.

Finalmente, en julio de 2010 Rokicki, Kociemba, Davidson y Dethridge demuestran que el número de Dios es 20[39].

En cuanto a métodos concretos para resolver el cubo, varias alternativas existen actualmente. El "método del principiante"[34] es el que suele ser utilizado por los cuberos novatos. Consiste en armar el cubo por niveles: primero se resuelve la capa inferior, luego la del medio y finalmente la superior. Esto se lleva a cabo utilizando secuencias de giros que preserven lo que ya ha sido armado en los pasos anteriores. Es un método amigable para humanos, ya que requiere memorizar pocas secuencias para ubicar correctamente todas las piezas del cubo, pero suele incurrir en pasos redundantes lo que implica soluciones largas.

En competencias de speedcubing, torneos donde gente intenta armar cubos de Rubik lo más rápido posible, otros métodos suelen ser utilizados, que favorecen el tiempo de resolución y reducen el largo de las soluciones, pero que requieren memorizar más secuencias de rotaciones. El método "CFOP" (abreviación inglesa de "Cross, First 2 layers, Orientation last layer, Permutation last layer"), propuesto por Jessica Fridrich[14] y en menor medida el método "Petrus", propuesto por Lars Petrus[26] son métodos que caen en esta categoría, pero que suelen ser considerados de dificultad intermedia o avanzada para humanos.

Otros métodos menos amigables para seres humanos se basan en búsqueda. Algunos funcionan descomponiendo el cubo de Rubik usando teoría de grupos en varios subgrupos para reducir el espacio de búsqueda, como es el caso del algoritmo de Thistlethwaite[45] (5 subgrupos) y su versión mejorada, el algoritmo de Kociemba[17] (3 subgrupos). Otro enfoque es el algoritmo de Korf[19], que se basa en una búsqueda informada con profundidad iterativa (IDA*)[18] y bases de datos de patrones[7]. Este último es una instancia del algoritmo de Dios, ya que entrega siempre la solución óptima, a cambio de un costo computacional tremendo, exponencial en el largo de la solución[20]. DeepCubeA[2], un método basado en aprendizaje por refuerzo profundo, es capaz de encontrar soluciones cortas sin conocimiento específico para el cubo de Rubik, y además generaliza a otros puzzles como el puzzle 15 y luces fuera.

# Visión Computacional

Si bien encontrar una solución para un cubo de Rubik se puede considerar un problema resuelto, hay muchos pasos previos que se deben realizar antes de llegar a una representación del estado apta para su resolución. Para obtener el estado del cubo, bien se puede ingresar manualmente (por ejemplo, mediante teclado) o se puede detectar automáticamente. Si se desea esto último, como es el caso de este proyecto, se deben aplicar técnicas de visión computacional.

La visión computacional es el área de la inteligencia artificial que estudia cómo automatizar las tareas realizadas por el sistema visual humano. En este campo se desarrollan métodos para que computadores puedan percibir, analizar, procesar e interpretar imágenes del mundo real[43].

Algunos de los problemas que vuelven a la visión computacional una tarea muy compleja son la perspectiva, la orientación, las oclusiones y la deformación, pues pueden causar formas totalmente distintas para el mismo objeto. Otros aspectos dificultan la percepción de color, como las sombras, la reflexión especular y las interreflexiones[40].

El problema de detectar los colores del cubo de Rubik puede atacarse con modelos de aprendizaje supervisado, pero aún con numerosos ejemplos es difícil generalizar a diferentes entornos, cambios en iluminación o desgaste del material [24].

# Robótica

Un robot es un agente que realiza tareas en el mundo físico por medio de sus efectores[40]. La robótica es el estudio y diseño de estos agentes. La mayoría de los robots puede clasificarse como robot manipulador o robot móvil. Baxter es un robot del tipo manipulador, pues sus únicos efectores son sus brazos y su cabeza, siendo incapaz de desplazarse.

Un problema de importancia en robótica es el de las cinemáticas inversas[4] (o “IK”, del inglés “Inverse Kinematics”), que consiste en obtener coordenadas en el espacio de configuración propio del robot -que en el caso de Baxter, corresponde a encontrar los ángulos de cada articulación de un brazo- a partir

de coordenadas cartesianas del espacio tridimensional. Esto no siempre es posible, ya que el mismo brazo del robot restringe las posiciones válidas. Por ejemplo, no es posible alcanzar puntos más allá del largo del brazo de Baxter, ni curvar los brazos más allá de los ángulos límite de las articulaciones, ni llegar a puntos que causen colisiones con sigo mismo. Algunas posiciones son alcanzables pero sólo en una determinada orientación, por ejemplo, un punto ubicado muy a la derecha del robot puede alcanzarse con el brazo izquierdo sólo orientado hacia la derecha. Estas restricciones juegan un papel importante al decidir las posturas que se le den al robot para manipular el cubo.

Existe un framework llamado ROS (Robot Operating System)[27] que contiene un conjunto de bibliotecas que simplifican la escritura de software para robots. Para programar al robot Baxter, se requiere como dependencia a ROS. De ésta manera, Baxter cuenta un servicio de cinemáticas inversas, pero también es posible especificar los ángulos de cada articulación directamente.

## Baxter armando el cubo de Rubik

Algunos trabajos previos que intentan armar el cubo de Rubik con el Robot Baxter son los de ActiveRobots y proyectos de estudiantes universitarios. El primero [1], utiliza un elemento de hardware adicional para escanear el cubo, el cual no se encuentra disponible en la Universidad de Concepción. El trabajo de [8] sólo es capaz de rotar las caras izquierda y derecha del cubo. El trabajo de [3] no emplea visión computacional para la detección de estados.

Existen varios trabajos de robots armando el cubo de Rubik, en particular, destaca el trabajo de [25], capaz de armar un cubo de Rubik en 0.38 segundos, superando el récord mundial humano. Sin embargo, este requiere un hardware altamente especializado, mientras que los demás trabajos como el de [12] o [48] utilizan otros Robots diferentes de Baxter, los cuales no se encuentran disponibles en la Universidad de Concepción.

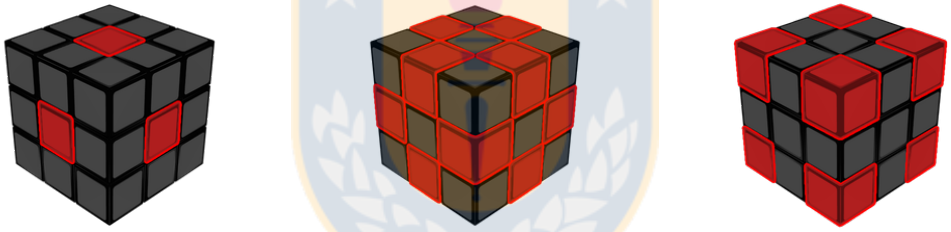
# 3 Sistema/Modelo/Método

## 3.1. Modelo

El cubo de Rubik $3 \times 3 \times 3$ está compuesto de 3 tipos de piezas distintas:

- **Centros**: piezas con un sólo color, que sólo rotan sobre su eje, 6 en total.
- **Aristas**: piezas con 2 colores, 12 en total.
- **Esquinas**: piezas con 3 colores, 8 en total.

La cantidad total de facelets es entonces $1 \times 6 + 2 \times 12 + 3 \times 8 = 54$.



(a) Centros (b) Aristas (c) Esquinas

Figura 3.1: Los 3 tipos de piezas del cubo de Rubik.

### 3.1.1. Convenciones

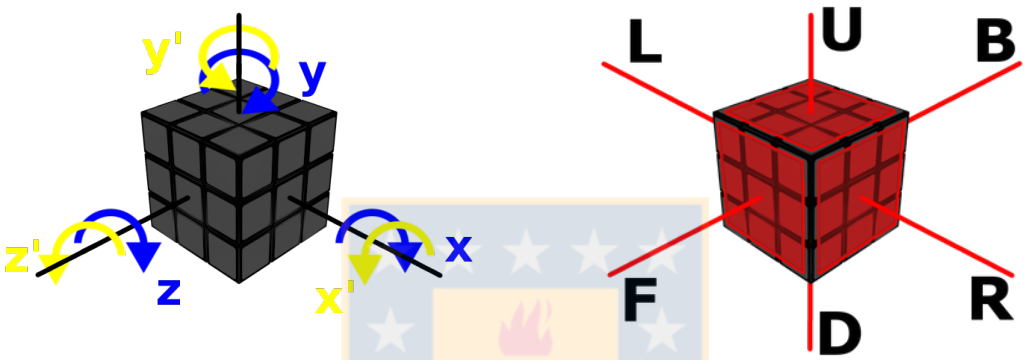
Existe una notación estándar para describir los ejes, las caras y los facelets del cubo [42]. Los 3 ejes de coordenadas del sistema de referencia utilizado se definen de la siguiente manera:

- $x$: eje que atraviesa las caras derecha e izquierda del cubo, con respecto al observador.

- $y$: eje que atraviesa las caras superior e inferior del cubo.
- $z$: eje que atraviesa las caras frontal y trasera del cubo.

Así, cada una de las piezas esquina del cubo tiene 1 color por eje, mientras que las aristas tienen un color en sólo 2 de los 3 ejes y los centros en solamente 1.

Las caras del cubo se identifican mediante la letra inicial de la palabra en inglés que identifica su posición con respecto al observador. Estas letras son **U** (up), **R** (right), **F** (front), **D** (down), **L** (left) y **B** (bottom).




(a) Ejes (b) Caras

Figura 3.2: Notación de ejes y caras del cubo.

De manera similar, los movimientos posibles de cada cara del cubo también quedan descritos por la letra correspondiente de la cara rotada. En este punto, se decidió adoptar una notación ligeramente diferente de la estándar. En la notación propuesta, $XN$ indica que la cara $X$ rota en $90 \times N$ grados, donde los grados se miden en sentido horario con respecto a un vector normal a la superficie de la cara a rotar. Entonces, por ejemplo "R1" significa rotar la cara derecha en 90 (o equivalentemente, $-270$) grados, "R2" es rotar la cara derecha en 180 (o $-180$) grados y "R3" rotar la cara derecha en 270 (o $-90$) grados. La razón del por qué usar esta notación es tan sólo un detalle de implementación, pues de esta manera el string que representa cada rotación tiene un largo fijo.

Estos ángulos bastan para describir todos los movimientos posibles, ya que 360 equivale a una operación nula y cualquier otro ángulo dejaría el cubo en un estado inválido, impidiendo los movimientos de algunas de las otras caras.

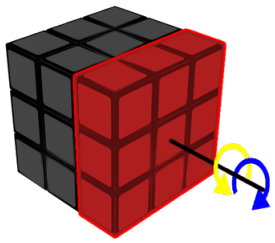


Figura 3.3: Flecha azul: movimiento *R1* (90 grados sentido horario). Flecha amarilla: movimiento *R3* (90 grados sentido antihorario).

Las piezas se identifican con las letras de las caras a las que pertenecen. Por lo tanto los centros son *U*, *R*, *F*, *D*, *L* y *B*, las aristas son *UF*, *UR*, *UL*, *UB*, *DF*, *DR*, *DL*, *DB*, *FR*, *FL*, *BR* y *BL* y las esquinas son *URF*, *URB*, *ULF*, *ULB*, *DRF*, *DRB*, *DLF*, y *DLB*.

El estado final del cubo se representó como un arreglo de 54 elementos, donde cada elemento representa un facelet. El orden de los facelets se muestra en la figura 3.4. La asignación de colores a cada cara es totalmente arbitraria y no tiene importancia para el sistema desarrollado. Lo importante es que, una vez se decida una asignación, se mantenga hasta el final.

## 3.2. Sistema

El sistema desarrollado consiste de varios módulos que pueden agruparse según su función en las siguientes categorías:

**Manipulación:** Módulos que se encargan de mover las articulaciones y controlar las pinzas del robot.

**Visión:** Módulos encargados de la detección del estado del cubo.

**Resolución:** Módulo encargado de buscar una secuencia de rotaciones de caras que lleven al cubo a su estado resuelto.

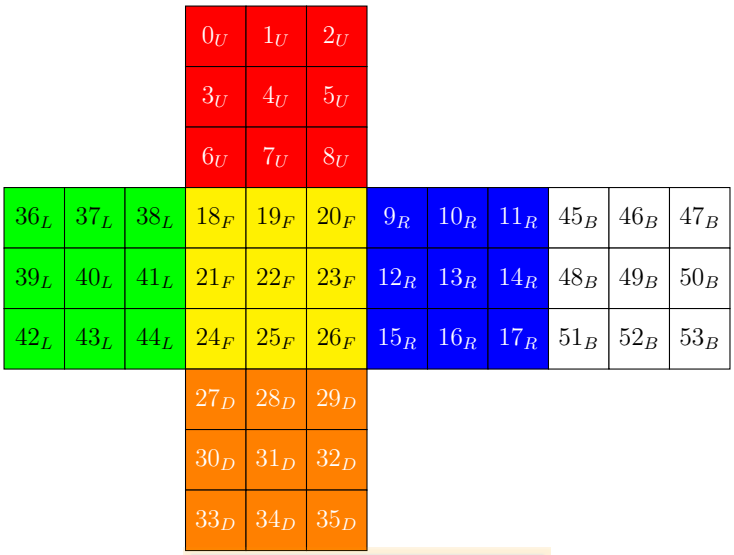



Figura 3.4: Orden de los facelets. El valor representa su posición el arreglo y el subíndice la cara a la que pertenece.

### 3.2.1. Manipulación

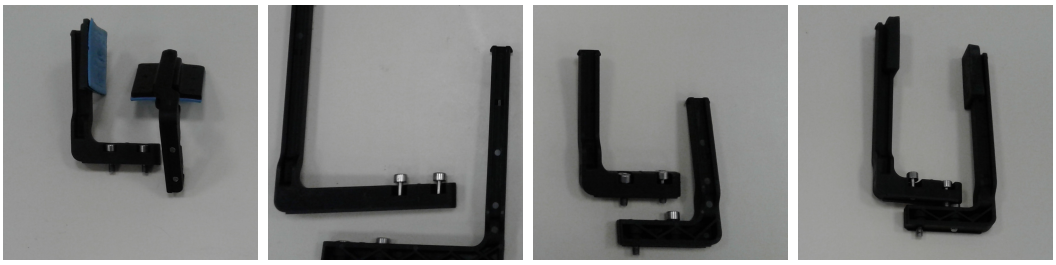
Consiste de los módulos que se encargan de la parte hardware del sistema, es decir los movimientos de las articulaciones del robot y la apertura, cierre y calibración de los grippers. Para mover los brazos del robot se usa una combinación de cinemáticas inversas y manipulación directa de ángulos de sus articulaciones.

Las tareas requeridas incluyen:

1. Recoger el cubo.
2. Acercar el cubo a una cámara para detectar su estado.
3. Realizar rotaciones de las caras.
4. Mover el cubo de un brazo a otro cuando sea necesario.

El robot Baxter posee varios tipos de grippers, mostrados en la figura 3.5. Por el tamaño del cubo, se utilizó el gripper largo y angosto, para tener un agarre lo suficientemente profundo y firme.

Los agarres son siempre de la forma como se muestra en la figura 3.6b. Esto impide la rotación de tres caras: las 2 que tocan los grippers y la cara que está



(a) Gripper 1. (b) Gripper 2. (c) Gripper 3. (d) Gripper 4.

Figura 3.5: Grippers disponibles. Los de la figura (a) sólo permiten rotar una cara sin cambiar de mano el cubo. Los de la figura (b) son más anchos que el largo de las caras del cubo. Los de la figura (c) son muy cortos para tener un agarre estable. Los de la figura (d) fueron los gripper utilizados.

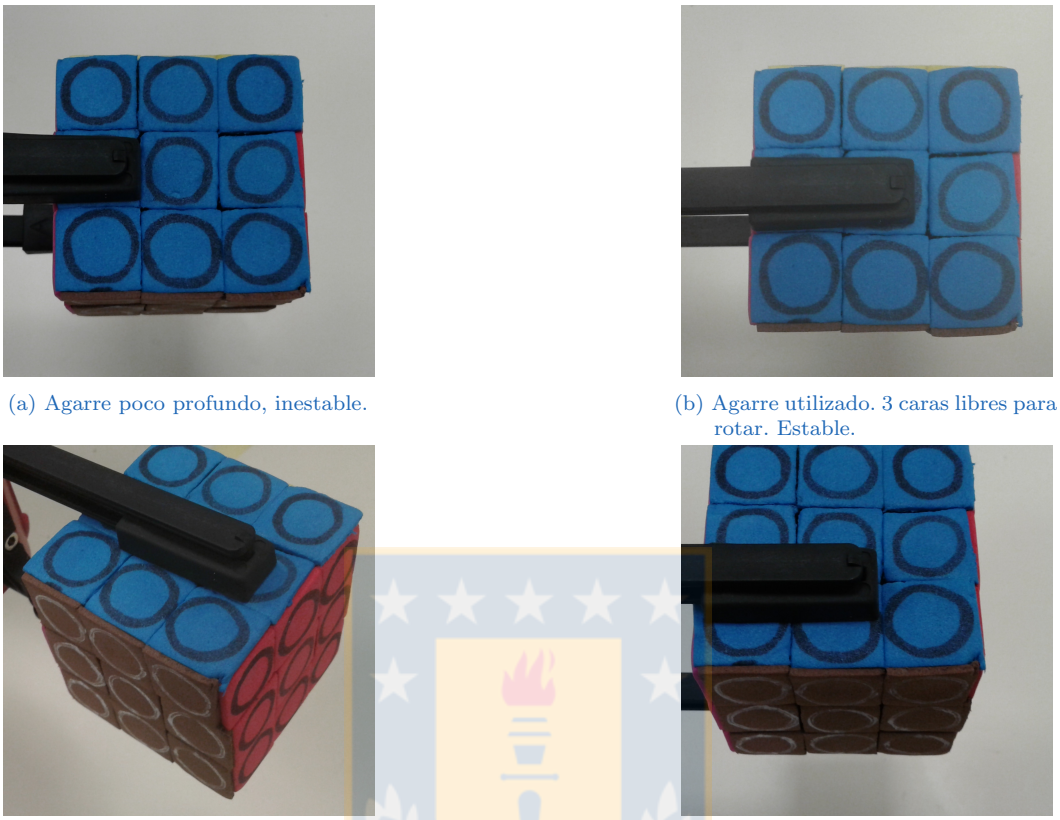
más cercana al brazo del robot. La manera en que se realizan las rotaciones es que un brazo sostiene el cubo en una orientación cómoda para que el otro brazo se acerque y agarre una cara no bloqueada de manera perpendicular, y la gire en 90, 180 o 270 grados según corresponda. Esto permite rotar libremente 3 caras. Para rotar las otras 3, **se cambia el cubo al otro brazo**, para que éste lo sostenga en una forma que libere las caras previamente bloqueadas.

### Recogida

La ejecución comienza con el cubo en la mesa, en una posición marcada cuyas coordenadas son conocidas. Los ejes se definieron con respecto al observador mirando de frente al robot, y no con respecto al punto de vista del robot. Utilizando el servicio de cinemáticas inversas de Baxter, se lleva el brazo izquierdo a una posición cercana al cubo, con las pinzas apuntando hacia abajo, para luego recogerlo. Este agarre bloquea las caras $L$, $R$ y $U$.

### Acercamiento a cámara

El servicio de cinemáticas inversas, si bien garantiza la posición y orientación final solicitada para un gripper, no especifica nada sobre la trayectoria a realizar. Esto puede causar varios problemas, entre ellos los más graves son: la imposibilidad de llevar a cabo un movimiento debido a la complejidad de



(a) Agarre poco profundo, inestable.

(b) Agarre utilizado. 3 caras libres para rotar. Estable.

(c) Agarre profundo, cara lejana (roja) bloqueada.

(d) Agarre impreciso, cara lateral (marrón) bloqueada. Inestable.

Figura 3.6: Diferentes agarres.

la trayectoria, a pesar de que sea una configuración válida; y la ejecución de ciertas trayectorias que causan que el robot se golpee a sí mismo. Debido a la naturaleza misma de IK, no es posible predecir cuándo suceden estos errores. Así que en este paso se mueve el brazo utilizando directamente los ángulos de cada una de sus articulaciones, para guiarlo a una configuración favorable, seguido inmediatamente por cinemáticas inversas para obtener la posición y orientación final deseada. Las 6 configuraciones necesarias para apuntar cada cara a la cámara se muestran en la figuras 3.8 y 3.9.

Figura 3.7: Posición y orientación al recoger el cubo. Los grippers tocan las caras $L$ y $R$.

(a) Capturando la cara $F$ (b) Capturando la cara $D$ (c) Capturando la cara $B$

Figura 3.8: Configuraciones del brazo izquierdo para acercamiento a la cámara.

## Rotaciones de caras

Esta tarea se resuelve de una manera muy similar a la anterior, en el sentido de que usa la misma idea de ángulos directos seguido por IK. Existe una diferencia con respecto a las coordenadas utilizadas: acá se escogió una posición en la que los grippers del robot pudiesen llegar en 3 orientaciones distintas por cada brazo. Estas configuraciones se muestran en las figuras 3.10 y 3.11.

Otra diferencia es que en este paso se agregan posiciones intermedias al acercar

(a) Capturando la cara $L$ (b) Capturando la cara $U$ (c) Capturando la cara $R$

Figura 3.9: Configuraciones del brazo derecho para acercamiento a la cámara.

(a) Rotación cara $F$ (b) Rotación cara $D$ (c) Rotación cara $B$

Figura 3.10: Configuraciones del brazo izquierdo para rotaciones.

(a) Rotación cara $L$ (b) Rotación cara $U$ (c) Rotación cara $R$

Figura 3.11: Configuraciones del brazo derecho para rotaciones.

el brazo libre a la posición del cubo, para que el servicio de cinemáticas inversas prefiera movimientos paralelos y no cause colisiones no deseadas.

Para efectuar una rotación de alguna cara, la secuencia de pasos es:

1. Brazo que sostiene el cubo se mueve a una posición definida en función de la cara que será rotada (especificada mediante ángulos de articulaciones directamente).
2. Brazo libre se mueve a una posición cercana a la cara a rotar del cubo, pero no lo suficiente para agarrarlo (especificada mediante IK).
3. Brazo libre se mueve a una posición aún mas cercana a la cara a rotar del cubo, lo suficiente para agarralo (IK, movimiento paralelo).
4. Gripper de brazo libre se cierra.
5. Brazo libre rota articulación de su muñeca en el ángulo deseado (ángulos directos).
6. Gripper de brazo libre se abre.
7. Brazo libre se aleja del brazo que sostiene el cubo para permitir libertad en los movimientos siguientes (IK, seguido de ángulos directos).

#### Cambio de mano

Este paso utiliza ángulos directos y cinemáticas inversas. Las posiciones para realizar este paso son las mismas de los brazos ocupados para los movimientos $U$ y $R$ respectivamente (ambos brazos horizontales, apuntandose a si mismos), con la diferencia de que el brazo izquierdo tiene un desplazamiento de 90 grados. Así, cuando se cambia el brazo de la mano izquierda a la derecha, se liberan las caras $L$, $R$ y $U$, pero se bloquean las caras $F$, $B$ y $L$, y viceversa.

Al tener 3 caras libres por brazo, no suele ser necesario realizar muchos cambios de mano para llevar a cabo una secuencia de giros. Sin embargo, en el peor caso siempre se tendrá tantos cambios de mano como rotaciones se vaya a realizar.

### 3.2.2. Visión

Estos módulos se encargan de obtener el estado del cubo. Es la parte más compleja del sistema, en términos de la cantidad de submódulos involucrados.

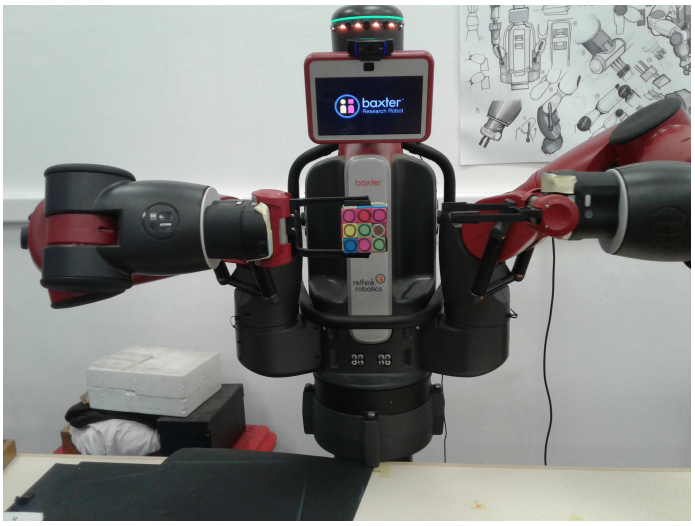


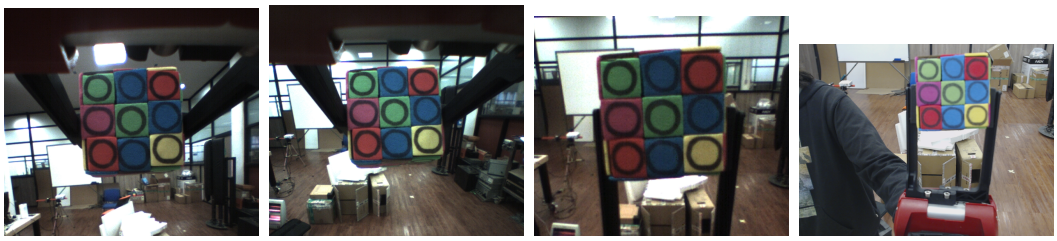

Figura 3.12: Cambio de mano.

### Preparaciones previas en el robot

Si bien Baxter posee cámaras en sus 2 manos y en su cabeza, la calidad de la imagen que se obtiene usándolas es insatisfactoria. Se decidió entonces utilizar una cámara externa, en concreto, una Logitech C920[46], la cual se ubicó encima de la cámara de la cabeza del robot. Ésta fue conectada directamente al computador desde donde se ejecuta el programa principal, pues a pesar de que el robot tiene interfaces USB (que permite el uso de joysticks entre otros dispositivos) no se encontró documentación ni códigos de ejemplo respecto del uso de cámaras externas.

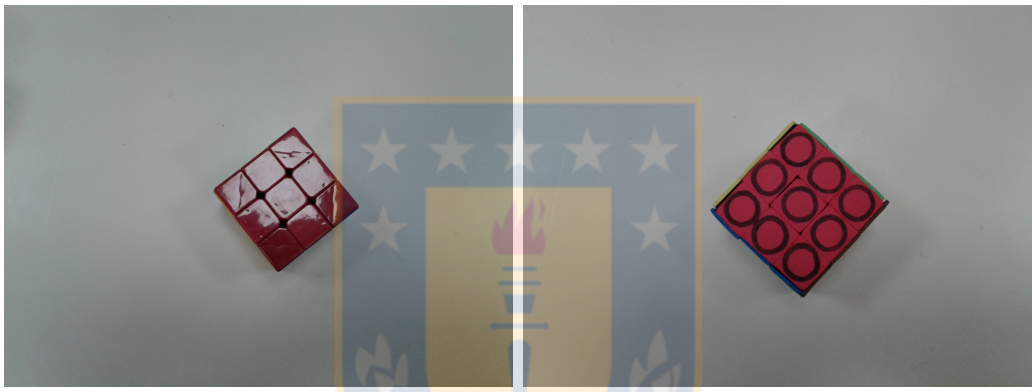
### Preparaciones previas cubo

El cubo fue cubierto con etilvinilacetato, mejor conocido como goma EVA. Esto cumple varios objetivos. En primer lugar, es un material que refleja la luz más difusamente que el plástico de los stickers del cubo, por lo que los colores capturados por la cámara tienden a ser más uniformes y se evitan las reflexiones especulares en la imagen. La diferencia es notable y se puede apreciar en la figura 3.14.



(a) Cámara brazo izquierdo. (b) Cámara brazo derecho. (c) Cámara de la cabeza. (d) Cámara externa.

Figura 3.13: Calidad de imagen de las distintas cámaras disponibles. Las imágenes tomadas por las cámaras del robot presentan ruido en forma de un patrón mallado, mientras que la externa no tiene este problema.



(a) Luz sobre el plástico. Algunos facelets podrian ser erróneamente detectados como blanco. (b) Luz sobre la goma EVA.

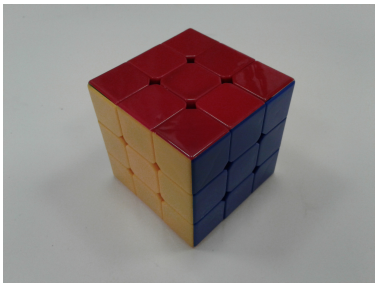
Figura 3.14: Difusión de luz en ambos materiales.

En segundo lugar, la goma EVA aumenta el roce entre los grippers y el cubo, ya que el plástico original tendía a deslizarse con mayor facilidad. Por último pero no menos importante, provee algo de protección cuando el cubo es maltratado por los movimientos poco delicados del robot.

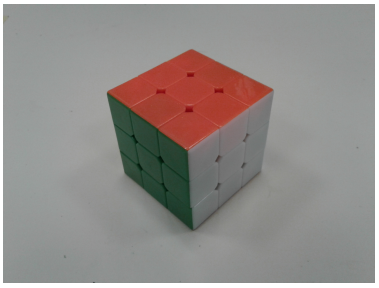
Los colores de goma EVA utilizados fueron rojo, azul, verde, amarillo, rosado y marrón, correspondientes a los mismo colores originales con excepción de los 2 últimos (naranja y blanco, originalmente), pues no se disponía de naranja y el blanco se ensucia bastante luego de muchas manipulaciones.

Sobre cada facelet del cubo se dibujó un círculo con un marcador permanente,

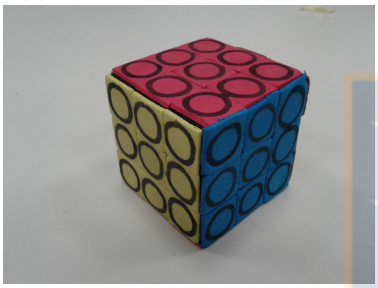
negro, para denotar el borde de las regiones de interés (los facelets) con un color que contrastase los colores claros de la goma EVA, con excepción del marrón, cuyos círculos fueron dibujados con color blanco para mayor contraste.



(a) Cubo original, caras $U$ $R$ y $F$.



(b) Cubo original, caras $D$ $L$ y $B$.



(c) Cubo cubierto, caras $U$ $R$ y $F$.

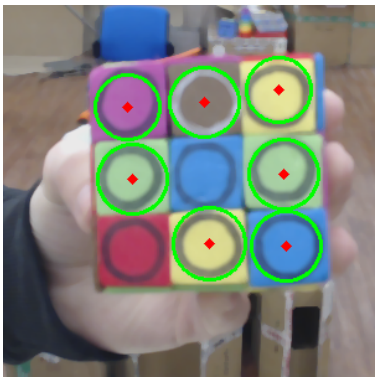
(d) Cubo cubierto, caras $D$ $L$ y $B$.

Figura 3.15: Colores del cubo de Rubik usado.

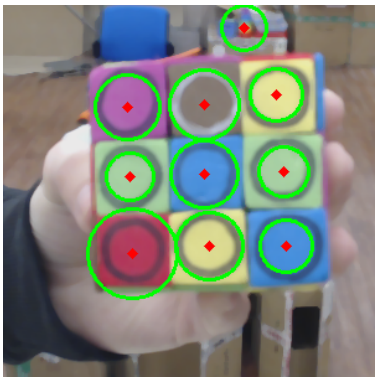
## Círculos de Hough

Una vez que el manipulador acerca el cubo a la cámara, se capturan fotos de cada cara. Sobre cada una de estas 6 imágenes, se aplica el algoritmo de Hough para detección de círculos, provisto por la biblioteca de OpenCV *HoughCircles* que implementa dicho algoritmo. El método itera hasta que encuentra exactamente 9 círculos en cada imagen, entregando sus centros y radios respectivos.

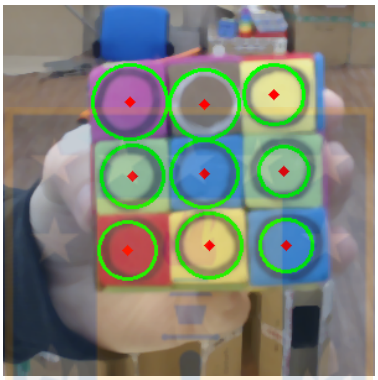
La transformada de Hough[41] es una técnica para extracción de características en imágenes, inicialmente concebida para detectar líneas, pero posteriormente se generalizó para detectar círculos o elipses. Una ventaja de utilizar esta transformación es que es muy robusta, pudiendo detectar círculos imperfectos, incluso cuando están parcialmente ocultos por un objeto entre la cámara y el



(a) Detección incorrecta, faltan círculos.



(b) Detección incorrecta, sobran círculos.



(c) Detección correcta.

Figura 3.16: Identificación de círculos.

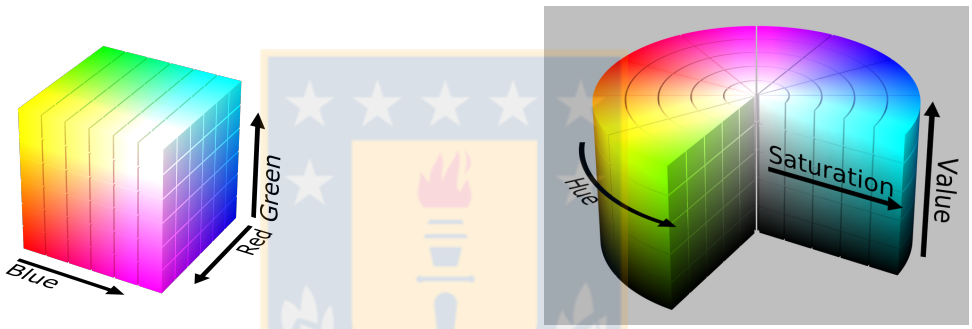
círculo. Su cómputo y en particular su implementación en OpenCV es bastante rápida, sin embargo no es perfecta. Dependiendo del ambiente de la imagen y los parámetros utilizados se pueden obtener falsos positivos, o falsos negativos, como se ve en la figura 3.16.

Mediante ensayo y error se obtuvieron parámetros lo suficientemente buenos para la transformada de Hough. Estos fueron radio mínimo y máximo de 10 y 35 píxeles para los círculos, con una distancia mínima entre ellos de 30 píxeles. Otros dos parámetros correspondientes a umbrales que controlan la cantidad potencial de círculos (`param1` y `param2`) se dejaron en 80 y 30 respectivamente.

### Colores representantes

Una vez se tienen los 54 círculos, se procede a extraer un color representativo de la región encerrada por cada uno. Se decidió usar la mediana RGB, pues en el caso de una detección imperfecta de un círculo (como el facelet rosado de la figura 3.16c) no se quiere que el borde del círculo afecte al color representante. Para mayor comodidad en la iteración, se utilizó como región un cuadrado inscrito en un círculo en lugar del círculo completo. Este paso tiene como resultado un vector 54 colores RGB.

### Clasificación




(a) Espacio de colores RGB.

(b) Espacio de colores HSV.

Figura 3.17: En RGB, los colores rojo, verde y azul corresponden a 3 ejes cartesianos. En HSV, el espacio tiene coordenadas cilíndricas, donde los colores "rojos" tienen radios similares entre sí, al igual que los "verdes", "azules" y demás colores.

Para identificar cuales facelets son del mismo color, independientemente de cual sea éste, se utilizó un modelo no supervisado para clasificarlos. El algoritmo utilizado fue el de mezcla de gaussianas (Gaussian Mixture Model, o *GMM*)[33], con 6 componentes. Las características usadas fueron las componentes matiz y saturación de la conversión de espacio RGB (Red, Green, Blue)[44] al espacio HSV (Hue, Saturation, Value)[23] de cada uno de los colores representantes de cada facelet.

Luego de realizar muchos experimentos se observó que la distribución RGB de los colores que vienen de la misma cara se tienden a alinear en rectas (ejemplo de distribución en figura .1, en anexo). Después de intentar infructuosamente con

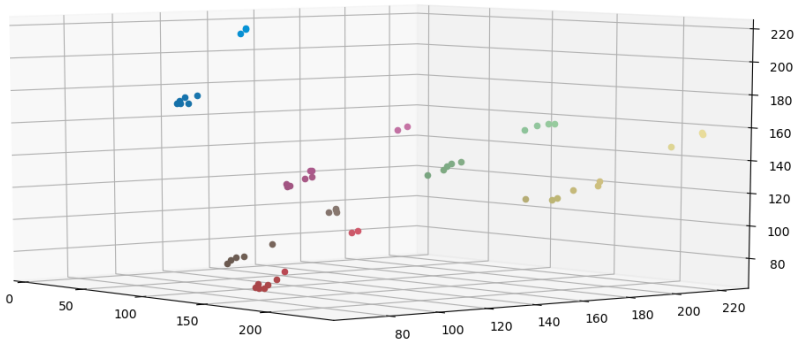



Figura 3.18: Colores representantes en espacio RGB. Los facelets del mismo color se tienden a alinear en rectas que crecen en los 3 ejes.

varios modelos de agrupamiento (K-Means, KMeans despúes de PCA, KMeans en espacio HSV) se decidió optar por un modelo de mezcla de gaussianas en HSV, pues los colores, con excepción del marrón tienen componentes de matiz muy similares, variando principalmente en su saturación o brillo.

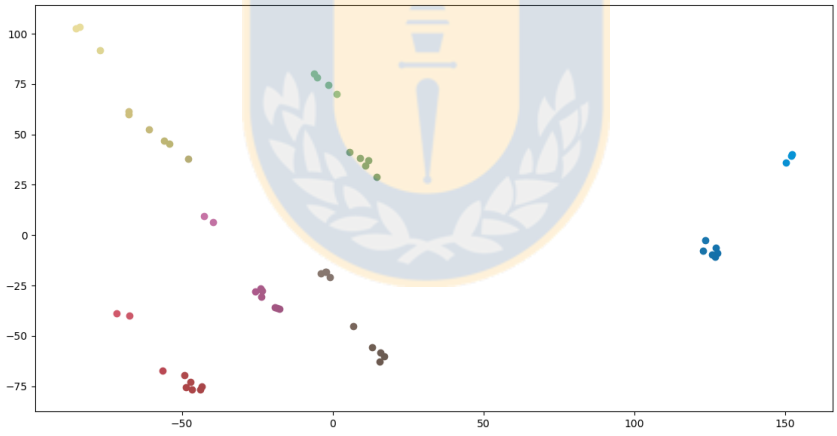



Figura 3.19: Colores representantes, luego de aplicar PCA con 2 componentes. Clases no trivialmente separables.

Aprovechando el hecho de que las piezas centro del cubo no se mueven, se puede saber con certeza de que sus facelets son todos de colores distintos, no así las aristas y esquinas del cubo que pueden moverse libremente a cualquier otra ubicación. Los colores de los centros se utilizaron entonces como las medias

iniciales de las gaussianas. Además, se asignaron las probabilidades a priori en 1/6 a cada gaussiana, pues todas las caras tienen la misma cantidad de facelets. Las covarianzas iniciales de cada gaussiana se tomaron con mayor varianza en el eje de saturación que en el de matiz. Esto es simplemente conocimiento previo de la distribución de los datos luego de muchos experimentos. El efecto de elegir las covarianzas de esta forma se muestra en las figuras 3.20 y 3.21.

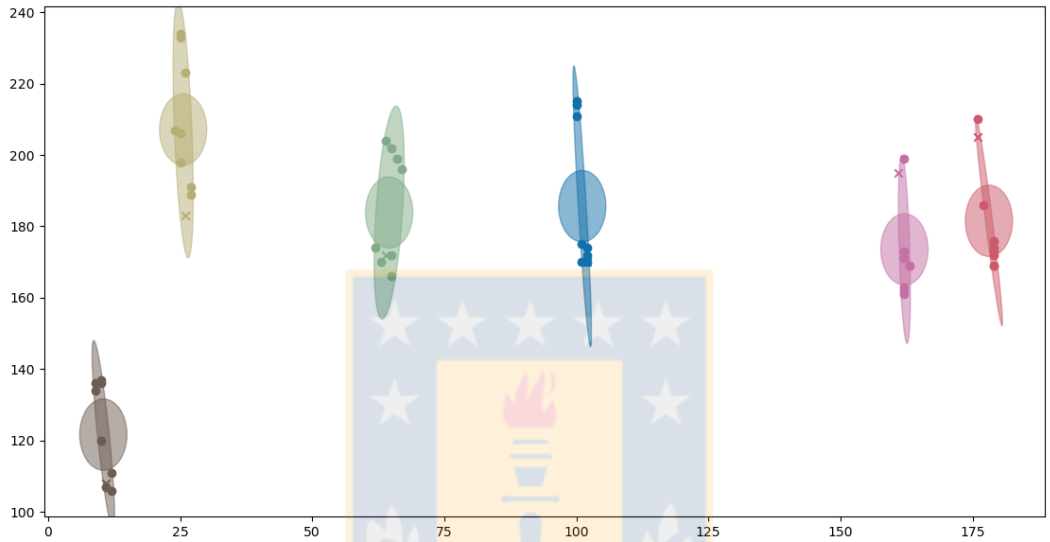



Figura 3.20: Colores representantes en espacio HSV. Eje horizontal es matiz, eje vertical es saturación. Los puntos marcados con cruces indican los representantes de los centros, y por ende las medias iniciales de las gaussianas. Las elipses grandes son las covarianzas finales, a 2 desviaciones estándar las elipses pequeñnas son las covarianzas iniciales. Esta clasificación es correcta.

El algoritmo de agrupamiento de un modelo de mezcla de gaussianas suele calcularse de manera iterativa a partir de una solución inicial, siguiendo el proceso de Expectation-Maximization. Cuando el algoritmo GMM era incializado con el resultado de procesar los puntos con K-Means (como lo hace, por ejemplo el modelo GMM de la biblioteca de Python scikit-learn), se tiende a confundir los colores rojo y rosado, ya que sus matices suelen están más cerca que la distancia en saturación más grande entre 2 rojos o 2 rosados.

Al finalizar este paso, si es que no hubo algún error de clasificación, se tiene un vector de 54 caracteres en el orden dado por la figura 3.4. Así, el caracter $X$ en

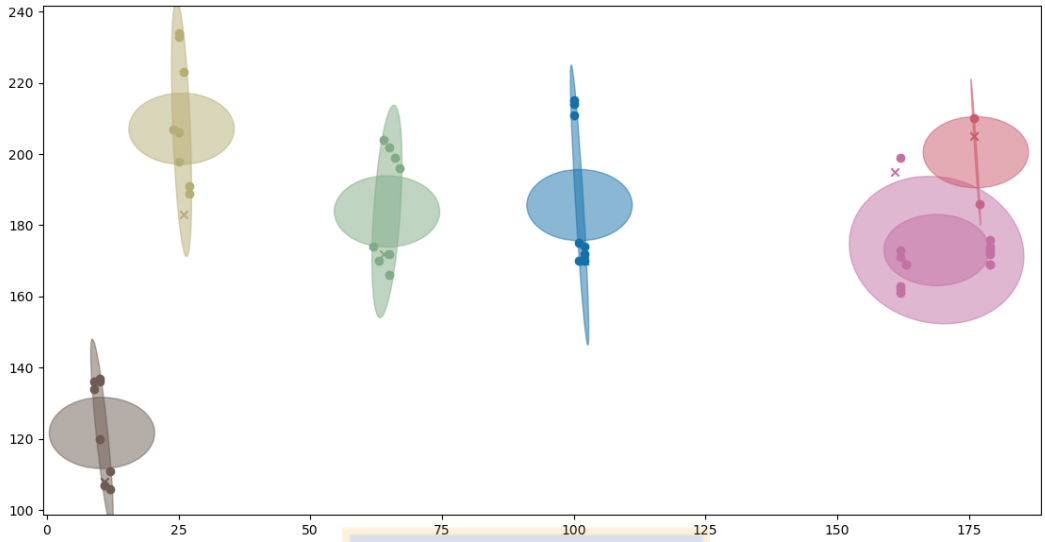



Figura 3.21: El mismo caso de la figura 3.20. Cuando las covarianzas crecen por igual en ambos ejes, ocurren más errores de clasificación, al igual que con KMeans. En particular, puntos rojos fueron incorrectamente clasificados como puntos rosados.

la posición $i$ indica que el facelet $i$ es del mismo color que el centro de la cara $X$. Por ejemplo, para los representantes de las figuras 3.20, el estado resultante es el string *LLLFUBRUBRRFRRUDBBFFUDFUDRRLFFBDDBRRBLUDLFDL-FUUUBBLDDL*.

### 3.2.3. Resolución

Para la fase de resolución se utilizó el algoritmo de 2 fases de Kociemba. Este algoritmo encuentra soluciones a cualquier configuración del cubo mediante una búsqueda exhaustiva, realizada en dos pasos.

En la primera fase, se lleva la permutación actual a algún estado en la que todas las piezas arista y esquina tengan orientación igual a cero (ver figura 3.22). La segunda fase lleva dicha configuración -sin orientaciones- al estado resuelto.

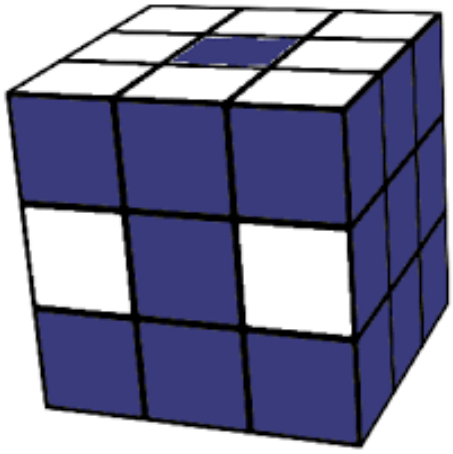


Figura 3.22: La orientación de una pieza del cubo se define con respecto a un sistema de referencia dado por los facelets marcados en blanco. Si el facelet marcado de una pieza coincide con la referencia, entonces tiene orientación 0. Para las aristas, si está volteada con respecto a la referencia, tiene orientación 1. Para las esquinas, si el facelet marcado está rotado 120 grados en sentido horario, tiene orientación 1, o $-1$ si es está en sentido antihorario. El subgrupo de permutaciones que resultan al finalizar la fase 1 de Kociemba son todas aquellas que preserven la orientación 0 para todas las piezas. Los movimientos que mantienen este invariante son $\{U1, U2, U3, D1, D2, D3, R2, L2, F2, B2\}$.

El no representar las orientaciones de las piezas reduce considerablemente el espacio de búsqueda. Más aún, son lo suficientemente pocos estados como para poder calcular las distancias exactas al estado resuelto mediante una búsqueda en anchura BFS[18]. Estos valores se utilizan para guiar la búsqueda de la fase 2, permitiendo obtener el camino óptimo del estado sin orientaciones al estado resuelto, el cual en el peor de los casos es de largo 18.

Por otro lado, la fase 1 debe llevar el cubo inicial a un estado sin orientaciones, pero no se sabe de antemano el objetivo, pues cualquiera que cumpla el requisito mencionado serviría. Precomputar las distancias exactas en este paso sería demasiado costoso. Este paso no necesariamente encuentra una solución de largo óptimo, pero en el peor de los casos el largo obtenido es de 12 rotaciones.

Así, el algoritmo de 2 fases de Kociemba siempre encuentra una solución de largo a lo más 30. Esto no es óptimo, pero es muy bajo. La mayor ventaja

de Kociemba es que es un algoritmo extremadamente rápido con la ayuda de tablas para lookup y pruning.

Se adaptó la implementación de [16] a las necesidades del sistema desarrollado, además de portarlo al lenguaje Python 2 para integrarlo facilmente a los demás módulos.

Este módulo recibe un string con el estado o permutación del cubo y entrega una secuencia de giros en el formato descrito en la sección 3.1.1. Por ejemplo, para el mismo string mostrado en la sección anterior (*LLLFUBRUBRRFRRUDBBF-FUDFUDRRLFFBDDBRRBLUDLFDLFUUUBBLDDL*), la secuencia de giros que lleva al cubo a su estado resuelto es *B1 L1 D1 R1*. Obsérvese que para que el robot realice dicha secuencia de acciones, si comienza con el cubo en su brazo izquierdo podrá realizar la rotación *B1* inmediatamente. Sin embargo, como el gripper izquierdo bloquea las caras $L$, $R$ y $U$, se deberá realizar un cambio de mano antes de realizar el giro *L1*. Lo mismo sucederá cuando posteriormente se quiera efectuar *D1* con el cubo en el brazo derecho y *R1* con el cubo en el brazo izquierdo.

### 3.2.4. Orden de ejecución final

El orden de ejecución final es el siguiente:

1. Robot recoge el cubo de Rubik con su brazo izquierdo. La posición donde queden ubicados los grippers corresponderán a las caras $R$ y $L$.
2. Robot mueve brazo izquierdo de manera de apuntar las caras $F$, $B$ y $D$ (en ese orden) directamente hacia la cámara ubicada en su cabeza. Cada cara es fotografiada una vez y se guarda la matriz correspondiente a la imagen RGB y los 9 círculos detectados en cada cara.
3. Robot entrega el cubo de su mano izquierda a su mano derecha.
4. Robot mueve brazo derecho de manera de apuntar las caras $R$, $L$ y $U$ (en ese orden) directamente hacia la cámara ubicada en su cabeza. Cada cara es fotografiada una vez y se guarda la matriz correspondiente a la imagen RGB y los 9 círculos detectados en cada cara.
5. Se extrae el color representativo de cada uno de los 54 círculos, dando a lugar a un arreglo de 54 colores RGB.

6. Se agrupan los colores obtenidos en el paso anterior para obtener el estado del cubo, como una permutación.
7. Se toma la permutación obtenida y se obtiene una secuencia de rotaciones para resolver el cubo, la solución.
8. Se realizan las rotaciones de la solución moviendo los brazos del robot, cambiando de mano cuando sea necesario.

# 4 Experimentos y resultados

Se realizaron 2 tipos de experimentos: de visión y de manipulación. Pruebas de resolución no se llevaron a cabo, ya que el solucionador siempre encuentra una solución, a menos que el cubo se encuentre en un estado inválido.

Se utilizó el scrambler legacy de la World Cube Association, el generador que solía utilizarse en las competencias oficiales de speedcubing para permutar cubos. Con él se generaron 20 permutaciones del cubo de Rubik. Cada una de estas permutaciones, también llamados "desarmes", consiste de 30 rotaciones, y fueron aplicadas manualmente sobre el cubo. La tabla .1 del apéndice muestra las secuencias de rotaciones utilizadas en cada experimento.

## 4.1. Visión

Para la visión, se utilizaron los 20 desarmes en su totalidad. Cada ejecución comenzó con el robot recogiendo el cubo desde una mesa.

Del total de pruebas, el 65 % no tuvo errores de clasificación. La cantidad de facelets mal clasificados en promedio fue de 1,35 facelets, con desviación estándar de 2,220. La cantidad de facelets mal clasificados en cada experimento se muestra en la figura 4.1. En la tabla .3 del apéndice, se muestra específicamente cuales fueron los estados del cubo en cada ejecución, y exactamente cúales fueron los faceles indebidamente clasificados.

Respecto a la totalidad de facelets en todos los experimentos, el 97,5 % fue etiquetado correctamente. Lamentablemente, basta con 1 sólo facelet incorrecto para que el solver sea incapaz de encontrar una solución, por lo que en el mejor de los casos el robot hubiese sido capaz de resolver el 65 % de estos desarmes.

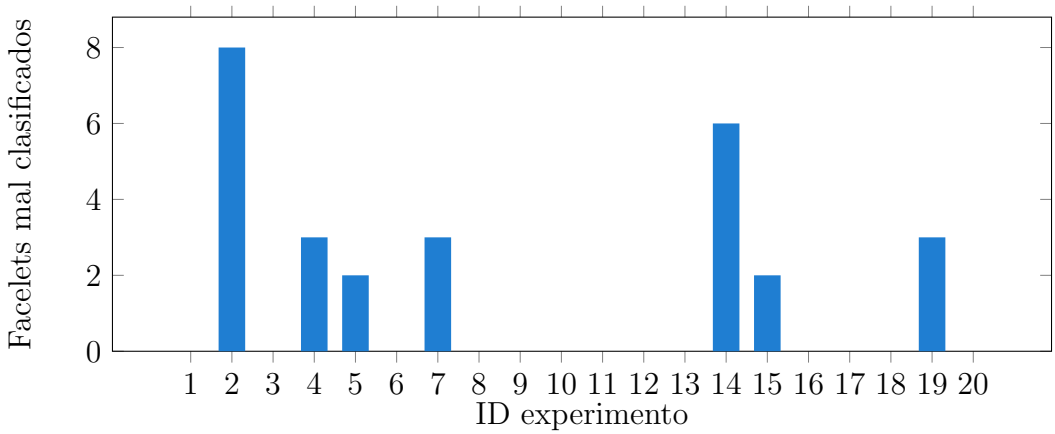



Figura 4.1: Facelets mal clasificados por cada prueba realizada.

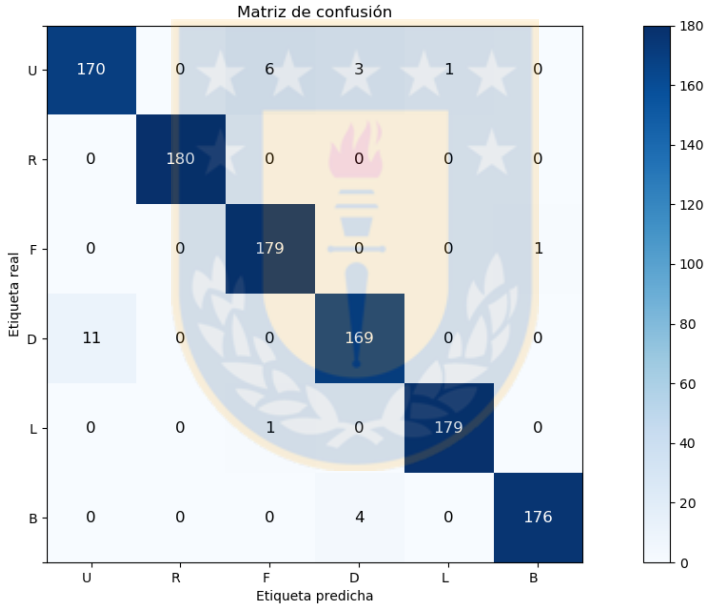



Figura 4.2: Matriz de confusión de predicción de colores, sin normalizar.

Una parte importante de los errores se dieron entre las caras U y D, que en las pruebas realizadas correspondieron a los colores rojo y rosado. Estos colores son muy similares entre sí, aunque esto depende mucho de las condiciones de iluminación del ambiente donde se encuentre el robot. El detalle de los errores

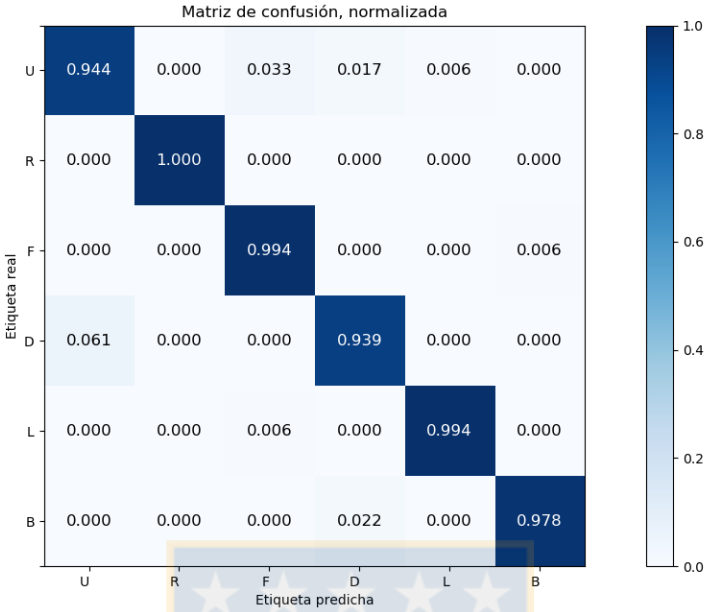



Figura 4.3: Matriz de confusión de predicción de colores, normalizada.

por cara se muestra en las figuras 4.2 y 4.3.

## 4.2. Manipulación

Para esta clase de pruebas se utilizaron los mismos desarmes que en las pruebas de visión, pero con el desarme $i$ truncado a $i$ rotaciones. De esta manera, se probó una secuencia por cada uno de los largos de secuencias posibles, desde el mínimo (1) hasta el máximo (20). Se midió en cuál giro el robot es incapaz de proseguir. Los resultados se ven en la tabla 4.1.

Si se observa la figura 4.4, se nota que a medida que aumenta la cantidad de cambios de mano, disminuye la fracción de la secuencia ejecutada exitosamente, aunque con gran variabilidad. Si notamos que a medida que crece el largo de una secuencia es más probable que hayan más cambios de mano, esto explica el por qué los desarmes más largos nunca fueron ejecutados en su totalidad. No obstante, esto no quiere decir que las secuencias cortas o con pocos cambios de mano estén libres de errores. Obsérvese por ejemplo, el caso de la secuencia 3.

| Largo | Cambios | Resultado |
|---|---|---|
| 1 | 1 | Ejecución completada exitosamente |
| 2 | 0 | Ejecución completada exitosamente |
| 3 | 1 | Ejecución falla en giro 3 |
| 4 | 2 | Ejecución completada exitosamente |
| 5 | 2 | Ejecución completada exitosamente |
| 6 | 1 | Ejecución completada exitosamente |
| 7 | 4 | Ejecución completada exitosamente |
| 8 | 2 | Ejecución completada exitosamente |
| 9 | 5 | Ejecución falla en giro 6 |
| 10 | 5 | Ejecución completada exitosamente |
| 11 | 7 | Ejecución completada exitosamente |
| 12 | 8 | Ejecución falla en giro 8 |
| 13 | 10 | Ejecución falla en giro 3 |
| 14 | 10 | Ejecución falla en giro 4 |
| 15 | 6 | Ejecución falla en giro 14 |
| 16 | 8 | Ejecución falla en giro 5 |
| 17 | 10 | Ejecución falla en giro 10 |
| 18 | 10 | Ejecución falla en giro 14 |
| 19 | 11 | Ejecución falla en giro 12 |
| 20 | 12 | Ejecución falla en giro 8 |

Cuadro 4.1: Resultados experimentos de manipulación. La columna "largo" es el largo de la secuencia y la columna "cambios" es la cantidad de cambios de mano que debe realizar el robot para ejecutarla.

A pesar de consistir de tan solo 3 movimientos y 1 cambio de mano, el robot fue incapaz de llevar a cabo la secuencia en su totalidad, fallando en el último giro. En el otro extremo se presentan casos como el de la secuencia 11, que con 11 movimientos y 7 cambios de mano se pudo ejecutar por completo.

Otra de las causas de error en la manipulación se deben a la precisión del robot Baxter. Sus especificaciones indican un error de ±0,5 centímetros al controlar las posiciones de las pinzas. Los facelets del cubo de Rubik utilizado tiene un ancho de 1,83 centímetros, por lo que hay poco margen para los errores de precisión de hardware. Los fallos generados al acumular errores luego de varias rotaciones dentro de una misma secuencia se pueden clasificar en los siguientes

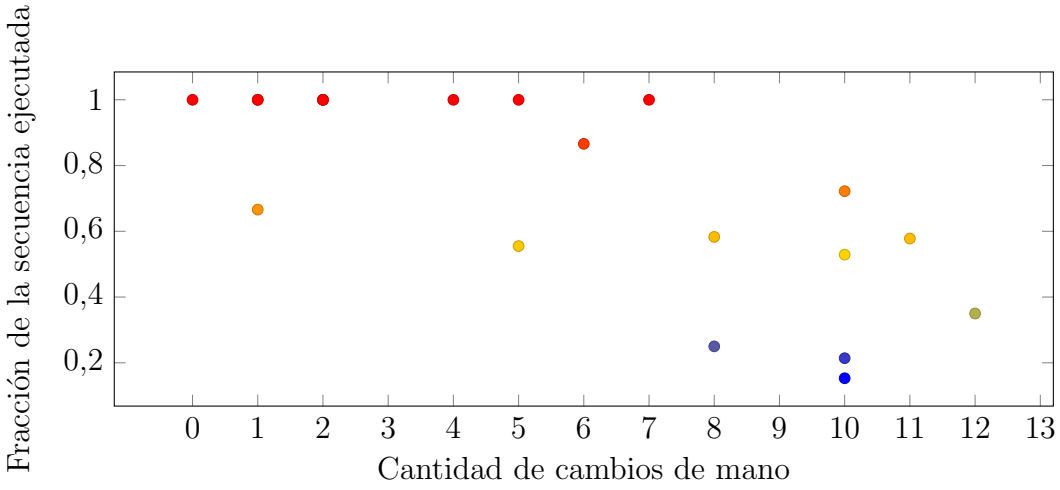



Figura 4.4: Cantidad de cambios de mano versus fracción de la secuencia ejecutada correctamente.

tipos:

1. agarre muy profundo: el robot intenta girar con un gripper una cara bloqueada por el otro brazo, produciendo que piezas aristas o esquinas salgan disparadas.
2. agarre poco profundo: el robot intenta girar con un gripper una cara pero no consigue estar lo suficientemente cerca. Esto no maltrata el cubo, pero produce que todas las rotaciones posteriores lleven el cubo a un estado totalmente distinto del deseado.
3. cambio de mano mal alineado: al cambiar el cubo de un gripper a otro, se bloquea accidentalmente alguna o varias de las caras que debiesen quedar libres. Esto puede permitir la ejecución momentáneamente, dependiendo de los giros siguientes, pero eventualmente puede causar errores del tipo 1 o 2.

Respecto de la visión, el factor más importante al detectar los colores correctamente es la iluminación. Aún con el cubo cubierto con goma EVA, el sólo hecho de apuntar el cubo en ángulos distintos hacia la cámara produce varianzas importantes entre los puntos representantes de un mismo color. Debido a los límites de las articulaciones del robot, no fue posible hacer que para las caras $U$ y $D$ (las que quedan opuestas a las cámaras de los brazos del robot)

quedasen perfectamente perpendiculares a la cámara de la cabeza robot. Para compensar este hecho, se le dio un ángulo de inclinación a las demás caras, pero que nuevamente, por limitaciones en las articulaciones del robot no fue exactamente el mismo ángulo que para las caras $U$ y $D$. Esta pequeña diferencia se puede apreciar en las figuras 3.20 y 3.21, donde se ve que cada color esta formado por 2 sub clusters. En cada cluster, el sub cluster de más arriba (es decir, el con mayor saturación), corresponde a facelets en las caras $U$ y $D$, que recibieron mayor luz al estar ligeramente más inclinados hacia la fuente de luz de la habitación. Si bien se podrían evitar las inclinaciones en ángulos distintos al agregar cambios de mano adicionales (para que las caras U y D no queden opuestas a las mano del robot), esto incurriría en más acumulación de errores de precisión, además de romper los invariantes definidos y aumentar la cantidad de posturas, complejizando bastante la manipulación y ralentizando la ejecución.

Para finalizar, no se consideran los resultados presentados (por "buenos" que hayan sido) como indiscutibles, pues dado que la magnitud del espacio de estados del cubo de Rubik es enorme ($\approx 4{,}5 \times 10^{19}$), ninguna cantidad no descomunal de pruebas será representativa de todas las configuraciones posibles.

# 5 Conclusiones

Se demostró que el Robot Baxter es capaz de resolver el cubo de Rubik en su totalidad, desde recogerlo, pasando por su detección y resolución hasta su manipulación, al menos para instancias pequeñas, con algunas preparaciones previas y con algo de suerte en la precisión del robot. Esto se consiguió aplicando diversas técnicas de varias áreas de la inteligencia artificial.

El sistema desarrollado consigue detectar correctamente sobre el 90 % de los facelets individuales del cubo, pero aún hay mucho que se puede mejorar. Hay varios aspectos que se pueden trabajar a futuro:

- Explorar la posibilidad de usar visión para recoger el cubo de la mesa, independientemente de dónde esté ubicado.
- Utilizar las cámaras de los brazos del robot para corregir de alguna manera los agarres mal alineados antes que sucedan.
- Explorar otros esquemas de posicionamiento, de manera que se minimicen o reduzcan los desplazamientos de los brazos y/o los cambios de mano.
- Aplicar técnicas más sofisticadas de visión computacional para detectar el estado del cubo, posiblemente sin ayuda de artefactos como los círculos o la goma EVA.
- Considerar un solucionador óptimo, o que, al menos genere soluciones más cortas que Kociemba en un tiempo razonable.

El proyecto realizado involucró utilizar un Robot para una tarea poco convencional. Se espera que este trabajo pueda inspirar y haga crecer el interés en la robótica en los estudiantes de Informática y carreras afines, que los haga explorar las capacidades de la robótica más allá de sus aplicaciones industriales típicas y que los incentive a resolver problemas complejos mediante la integración del mundo del software y del hardware.

# Glosario

**BFS** Breadth-First Search.

**CFOP** Cross, First 2 Layers, Orientation last Layer, Permutation last layer.

**EVA** Etilvinilacetato.

**GMM** Gaussian Mixture Model.

**HSV** Hue, Saturation, Value.

**HTM** Half-Turn Metric.

**IDA*** Iterative Deepening A*.

**IK** Inverse Kinematics.

**PCA** Principal Component Analysis.

**QTM** Quarter-Turn Metric.

**RGB** Red, Green, Blue.

**ROS** Robot Operating System.

**USB** Universal Serial Bus.

**WCA** World Cube Association.

# Bibliografía


[1] ActiveRobots. *Baxter Research Robot Solves Rubik's Cube*. 2014. URL: https://www.youtube.com/watch?v=vF9usYszChU.

[2] Forest Agostinelli y col. «Solving the Rubik's cube with deep reinforcement learning and search». En: *Nature Machine Intelligence* 1 (jul. de 2019). DOI: 10.1038/s42256-019-0070-z.

[3] Victoria Albanese. *Baxter "Solves"the Rubik's Cube*. 2018. URL: https://www.youtube.com/watch?v=RdZpreCTMT8.

[4] A. Aristidou y col. *Inverse Kinematics Techniques in Computer Graphics: A Survey*. 2018.

[5] SAGE Automation. *Introducing Baxter the world's first interactive manufacturing robot*. 2015. URL: https://www.sageautomation.com/news/pages/introducing-baxter-the-worlds-first-interactive-manufacturing-robot.

[6] Andrew Chen y Kevin Wang. «Robust Computer Vision Chess Analysis and Interaction with a Humanoid Robot». En: *Computers* 8 (feb. de 2019), pág. 14. DOI: 10.3390/computers8010014.

[7] Joseph C. Culberson y Jonathan Schaeffer. «Pattern Databases». En: *Computational Intelligence* 14.3 (1998), págs. 318-334. DOI: 10.1111/0824-7935.00065.

[8] Chris Dawes. *Baxter Rubiks Algorithm Demonstration*. 2015. URL: https://www.youtube.com/watch?v=9NDJwh9Zbxg.

[9] Desconocido. *The Mathematics of the Rubik's Cube*. 2009. URL: https://web.mit.edu/sp.268/www/rubik.pdf.

[10] Naomi T Fitter y col. *Exercising with Baxter: Design and Evaluation of Assistive Social-Physical Human-Robot Interaction*. 2018.

[11] C. Fitzgerald. *Developing baxter*. Abr. de 2013. DOI: 10.1109/TePRA.2013.6556344.

[12] Computing Forever. *Robot solves Rubik's Cube*. 2011. URL: https://www.youtube.com/watch?v=lD_LUOKKQfU.

[13] William Fotheringham. *Fotheringham's Extraordinary Sporting Pastimes*. Pavilion Books, 2006. ISBN: 9781861059536.

[14] Jessica Fridrich. *My System for Solving Rubik's Cube*. 2011. URL: http://www.ws.binghamton.edu/fridrich/system.html.

[15] Hans Kloosterman. *Rubik's cube in 42 moves*. 1990. URL: http://www.math.rwth-aachen.de/~Martin.Schoenert/Cube-Lovers/michael_reid__an_upper_bound_on_god%27s_number.html.

[16] Herbert Kociemba. *Cube Explorer (Software)*. URL: http://kociemba.org/cube.htm.

[17] Herbert Kociemba. *The Two-Phase-Algorithm*. 2015. URL: http://kociemba.org/twophase.htm.

[18] Richard E Korf. *Artificial intelligence search algorithms*. 1996.

[19] Richard E. Korf. «Finding Optimal Solutions to Rubik's Cube Using Pattern Databases». En: AAAI'97/IAAI'97 (1997), págs. 700-705.

[20] Richard E. Korf, Michael Reid y Stefan Edelkamp. «Time complexity of iterative-deepening-A*». En: *Artificial Intelligence* 129.1 (2001), págs. 199-218.

[21] Dan Kunkle y Gene Cooperman. «Twenty-six Moves Suffice for Rubik's Cube». En: ISSAC '07 (2007), págs. 235-242. DOI: 10.1145/1277548.1277581.

[22] Daniel Kunkle y Gene Cooperman. «Twenty-six Moves Suffice for Rubik's Cube». En: ISSAC '07 (2007), págs. 235-242. DOI: 10.1145/1277548.1277581.

[23] Dongqing Li, ed. *HSV Color Space*. Boston, MA: Springer US, 2008, págs. 793-793. ISBN: 978-0-387-48998-8. DOI: 10.1007/978-0-387-48998-8_656.

[24] Shenglan Liu y col. *Color Recognition for Rubik's Cube Robot*. Ene. de 2019.

[25] MIT News. *Solving a Rubik's Cube in record time*. 2018. URL: http://news.mit.edu/2018/featured-video-solving-rubiks-cube-record-time-0316.

[26] Lars Petrus. *Solving Rubik's Cube for speed*. 2006.

[27] Morgan Quigley y col. «ROS: an open-source Robot Operating System». En: *ICRA Workshop on Open Source Software* 3 (ene. de 2009).

[28] Silviu Radu. *Rubik can be solved in 27f*. 2006. URL: http://forum.cubeman.org/?q=node/view/53.

[29] Silviu Radu. *Solving Rubik's Cube in 28 Face Turns*. 2005. URL: http://forum.cubeman.org/?q=node/view/37.

[30] Michael Reid. *New Upper Bound*. 1992. URL: http://www.math.rwth-aachen.de/~Martin.Schoenert/Cube-Lovers/michael_reid__new_upper_bound.html.

[31] Michael Reid. *New Upper Bounds*. 1995. URL: http://www.math.rwth-aachen.de/~Martin.Schoenert/Cube-Lovers/michael_reid__new_upper_bounds.html.

[32] Michael Reid. *Superflip required 20 face turns*. 1995.

[33] Douglas A. Reynolds. *Gaussian Mixture Models*. 2009.

[34] Iván Osvaldo Ridao Freitas y Santiago Agustín Vidal. *Algoritmos de resolución para el cubo de Rubik*. 2005.

[35] Rethink Robotics™. *Baxter Research Robot Hardware Specifications*. 2015. URL: http://sdk.rethinkrobotics.com/wiki/Hardware_Specifications.

[36] Tomas Rokicki. «Towards God's Number for Rubik's Cube in the Quarter-Turn Metric». En: *The College Mathematics Journal* 45.4 (2014), págs. 242-242. ISSN: 07468342, 19311346.

[37] Tomas Rokicki. *Twenty-Three Moves Suffice*. 2008. URL: http://forum.cubeman.org/?q=node/view/117.

[38] Tomas Rokicki. «Twenty-Two Moves Suffice for Rubik's Cube®». En: *The Mathematical Intelligencer* 32.1 (mar. de 2010), págs. 33-40. ISSN: 1866-7414. DOI: 10.1007/s00283-009-9105-3.

[39] Tomas Rokicki y col. «The diameter of the rubik's cube group is twenty». En: *SIAM Review* 56.4 (2014), págs. 645-670.

[40] Stuart Russell y Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN: 0136042597, 9780136042594.

[41] Allam Shehata y col. *A Survey on Hough Transform, Theory, Techniques and Applications*. Feb. de 2015.

[42] David Singmaster. *Notes on Rubik's Magic Cube*. 1981. URL: https://maths-people.anu.edu.au/~burkej/cube/singmaster.pdf.

[43] Milan Sonka, Vaclav Hlavac y Roger Boyle. *Image processing, analysis and and machine vision (3. ed.)*. Ene. de 2008. ISBN: 978-0-495-24438-7.

[44] Sabine Süsstrunk, Robert E. Buckley y Steve Swen. *Standard RGB Color Spaces*. 1999.

[45] Morwen Thistlethwaite. *52-move algorithm*. 1981. URL: https://www.jaapsch.net/puzzles/thistle.htm.

[46] C920 HD Pro Webcam. URL: https://www.logitech.com/es-mx/product/hd-pro-webcam-c920.

[47] Dik Winter. *New upper bound on God's algorithm for Rubik's cube*. 1992. URL: https://sciences.ucf.edu/math/~reid/Rubik/Cubelovers/.

[48] Cezary Zielinski y col. *MRROC++ Based Controller of a Dual Arm Robot System Manipulating a Rubik's Cube*. Ago. de 2019.

# Anexos

## Secuencias de desarme utilizadas

| ID | Desarme |
|---|---|
| 1 | R2 D1 L3 D3 R1 B3 D2 F2 L1 R2 U3 R2 F2 D2 F1 U3 F2 D3 U3 L1 B2 L2 R2 U3 R1 U2 F2 D3 R2 U3 |
| 2 | D1 F2 D3 B2 L1 B2 F2 D3 B2 L3 F2 U1 R2 B2 U3 B3 D1 R2 F3 D3 L2 R2 F2 L3 D2 U1 R2 F3 L3 B1 |
| 3 | L2 U2 F1 R2 F3 R2 F3 L1 B2 R2 B1 F2 D2 U2 L2 R2 D2 B2 U3 F1 U3 L3 R2 D1 U1 B1 F3 L2 R3 F2 |
| 4 | B2 F3 R1 D2 U1 L3 R2 B1 U1 F2 U2 B3 L3 B1 F1 D3 U2 R3 B3 R3 F1 R2 D1 L3 F1 L2 R3 B1 F1 U1 |
| 5 | D1 R1 U2 L3 B2 R3 B3 R2 B3 L1 U1 B3 F2 L1 R1 U3 B1 U1 R2 D1 U1 B2 D2 U3 B2 U3 L2 R3 D3 F2 |
| 6 | U2 L3 U3 L3 R2 U3 B2 R2 B1 U1 L3 R3 D1 R3 D2 F2 R2 B1 F3 L1 B2 D1 U1 L3 U1 R2 F3 D2 F1 R1 |
| 7 | B1 F2 R3 F1 L1 B3 F2 D2 L3 R2 F3 L3 R1 D2 U1 F1 R2 F2 U3 R1 F1 D2 B2 R1 B1 R1 D3 U1 F3 R1 |
| 8 | D1 U1 L1 U2 L1 R1 B1 F1 R1 D3 U3 L3 R3 D1 U2 B3 R1 U1 R1 D1 B3 F1 U2 B2 F2 D1 U1 L2 D3 F2 |
| 9 | U3 R1 D3 B2 F2 U2 B3 L2 R2 F3 D2 U3 F3 L2 R1 D2 B3 D2 L2 U2 L1 U1 B1 F1 D2 L3 B2 U2 L3 U1 |
| 10 | L3 R1 B3 F3 U3 F2 L3 R1 D3 F3 U3 F1 D3 U3 B2 R3 B3 F3 R3 B2 F1 D2 B3 D1 B1 R1 F1 R2 U1 L2 |
| 11 | L3 B2 R1 B3 D1 F2 L3 B1 R1 B2 D3 B1 U1 F3 R2 D3 L2 R2 U1 B2 U2 R1 D3 U3 F1 R2 B3 F1 U2 F3 |
| 12 | D3 U1 L1 R1 B1 F1 R1 F2 U2 F3 L3 R2 D2 U2 F3 U3 F3 L3 B3 F1 U3 L3 B2 F1 L3 D1 L3 R2 D2 R3 |
| 13 | R3 D2 L1 R1 F1 L2 B1 F2 L2 D1 L2 R2 B1 F2 U2 B3 L1 U3 F1 D3 F2 L1 D1 F1 D1 F3 R1 U1 L3 F3 |
| 14 | B3 L2 R2 B2 F2 L1 F3 U1 F3 L3 U3 B2 F3 U2 F3 L1 D2 L3 R2 D2 F3 L2 D2 F1 D2 B3 R2 F2 D2 U1 |
| 15 | F1 U2 R3 U3 B1 F2 R2 D1 U1 L3 R3 B3 F1 D3 B3 F1 L3 B1 R1 U2 F3 U1 R3 D1 F3 D3 U2 L2 D2 B1 |
| 16 | F2 D2 L3 D3 L1 R1 D3 F3 D2 B3 U2 L1 D3 U1 L1 D3 B1 U1 R1 D3 R3 U1 R2 D3 L1 B1 R1 F3 L2 D1 |
| 17 | B3 F2 U1 F2 L1 R1 D1 U3 F1 U2 F2 L1 R1 D2 B1 D1 F2 R2 U3 R1 D2 U2 F3 L1 R3 F3 L3 R2 F2 L2 |
| 18 | B1 F3 D3 L3 B1 F2 R1 D3 L2 R1 F1 D2 F1 R3 F2 U3 B1 F2 D1 L3 U3 B2 F2 D3 L2 D3 U3 B1 U2 L1 |
| 19 | L3 U1 F1 D3 U3 R3 B2 U1 R2 F2 R3 B1 R1 U1 L2 B3 F3 U1 L2 R1 U3 B2 L3 U1 F1 R3 B1 L1 F2 L1 |
| 20 | D3 L3 R3 U2 R1 F2 L2 D3 F1 D3 R1 D2 U2 R2 F2 L2 R2 F3 R2 D3 B1 U1 B2 L1 R3 F3 D2 F1 R1 F2 |

Cuadro .1: Desarmes utilizados para la experimentos de visión.

# Desarmes truncados

| ID | Desarme |
|---|---|
| 1 | R2 |
| 2 | D1 F2 |
| 3 | L2 U2 F1 |
| 4 | B2 F3 R1 D2 |
| 5 | D1 R1 U2 L3 B2 |
| 6 | U2 L3 U3 L3 R2 U3 |
| 7 | B1 F2 R3 F1 L1 B3 F2 |
| 8 | D1 U1 L1 U2 L1 R1 B1 F1 |
| 9 | U3 R1 D3 B2 F2 U2 B3 L2 R2 |
| 10 | L3 R1 B3 F3 U3 F2 L3 R1 D3 F3 |
| 11 | L3 B2 R1 B3 D1 F2 L3 B1 R1 B2 D3 |
| 12 | D3 U1 L1 R1 B1 F1 R1 F2 U2 F3 L3 R2 |
| 13 | R3 D2 L1 R1 F1 L2 B1 F2 L2 D1 L2 R2 B1 |
| 14 | B3 L2 R2 B2 F2 L1 F3 U1 F3 L3 U3 B2 F3 U2 |
| 15 | F1 U2 R3 U3 B1 F2 R2 D1 U1 L3 R3 B3 F1 D3 B3 |
| 16 | F2 D2 L3 D3 L1 R1 D3 F3 D2 B3 U2 L1 D3 U1 L1 D3 |
| 17 | B3 F2 U1 F2 L1 R1 D1 U3 F1 U2 F2 L1 R1 D2 B1 D1 F2 |
| 18 | B1 F3 D3 L3 B1 F2 R1 D3 L2 R1 F1 D2 F1 R3 F2 U3 B1 F2 |
| 19 | L3 U1 F1 D3 U3 R3 B2 U1 R2 F2 R3 B1 R1 U1 L2 B3 F3 U1 L2 |
| 20 | D3 L3 R3 U2 R1 F2 L2 D3 F1 D3 R1 D2 U2 R2 F2 L2 R2 F3 R2 D3 |

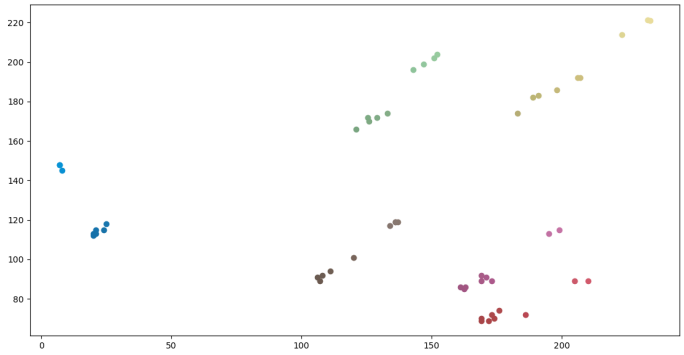
Cuadro .2: Desarmes utilizados para la experimentos de manipulación.

# Detalle experimentos visión

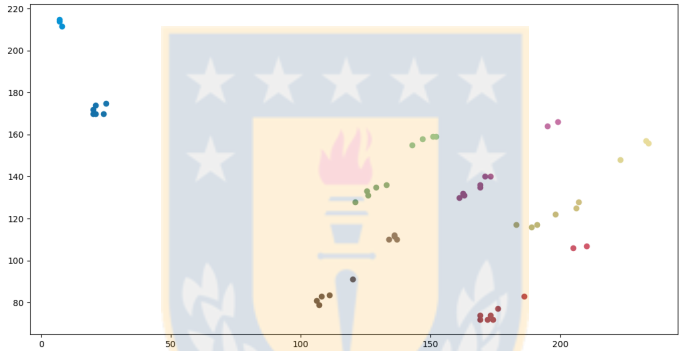
| ID | Errores | Detalle |
|---|---|---|
| 1 | 0 | FBBFUUFBDFRDLRUUDUDRRLFDUDFBFRLDRRFBLRLULFDBRLDUBBLLUB |
| | | FBBFUUFBDFRDLRUUDUDRRLFDUDFBFRLDRRFBLRLULFDBRLDUBBLLUB |
| 2 | 8 | BFFRURRULF**U**URR**D**F**U**UBF**UU**FBLURDB**D**LDLDRBRF**U**BLFRUBLLUBBLL**F**F |
| | | BFFRURRULF**D**URR**U**F**D**UBF**DD**FBLURDB**U**LDLDRBRF**D**BLFRUBLLUBBLL**U**F |
| 3 | 0 | FDBLUBLLBDDLBRFFUUDFRDFUBULRLDRDRRRLRBBDLFUFUULDUBRFBF |
| | | FDBLUBLLBDDLBRFFUUDFRDFUBULRLDRDRRRLRBBDLFUFUULDUBRFBF |
| 4 | 3 | BF**U**U**U**LFBRFBBDRUFDULUUUFRLUDFRRB**U**LUFRLFUBLFBDDRRDLBRBLL |
| | | BF**D**U**D**LFBRFBBDRUFDULUUUFRLUDFRRB**D**LUFRLFUBLFBDDRRDLBRBLL |
| 5 | 2 | RRFDUFLUURRDFRDDBBBRFDFLFUFLFLUDLURDBF**F**BLLRL**F**RBDBBULDB |
| | | RRFDUFLUURRDFRDDBBBRFDFLFUFLFLUDLURDBF**U**BLLRL**U**RBDBBULDB |
| 6 | 0 | FBBRURBLDBBRBRRFURUDLLFDUUURFLLDBRUBDFLDLFFUFDLLDBFURD |
| | | FBBRURBLDBBRBRRFURUDLLFDUUURFLLDBRUBDFLDLFFUFDLLDBFURD |
| 7 | 3 | DUFUUBFDDF**FF**DRF**F**RLUFRLFBUBRLLBLDBDRBLFRDLUBFBLRFRBLDDR |
| | | DUFUUBFDDF**UU**DRF**U**RLUFRLFBUBRLLBLDBDRBLFRDLUBFBLRFRBLDDR |
| 8 | 0 | URDDULBBLUBBFRLFURLRFFFRFUDDLRFDRUDBLFDBLUFLLRDBDBUUBR |
| | | URDDULBBLUBBFRLFURLRFFFRFUDDLRFDRUDBLFDBLUFLLRDBDBUUBR |
| 9 | 0 | UFLRULUBBRBFFRFBDRLRURFURUDBBLRDLLDFRFBLLDDUDULFDBUDBF |
| | | UFLRULUBBRBFFRFBDRLRURFURUDBBLRDLLDFRFBLLDDUDULFDBUDBF |
| 10 | 0 | LFURUDUUDBBBFRFFDDBBLDFLFDRRRULDLDUFUBRULFBBDLUFRBRLLR |
| | | LFURUDUUDBBBFRFFDDBBLDFLFDRRRULDLDUFUBRULFBBDLUFRBRLLR |
| 11 | 0 | LLFUURDLLDFUDRUUFUFFBUFFFDRUBBRDUBDBFRRBLBRBRLDDLBLLRD |
| | | LLFUURDLLDFUDRUUFUFFBUFFFDRUBBRDUBDBFRRBLBRBRLDDLBLLRD |
| 12 | 0 | RFUBULLUBUDLDRBFDFDRLRFFDUDRLRLDRUBLFRBLLFRBBFUUDBFDUB |
| | | RFUBULLUBUDLDRBFDFDRLRFFDUDRLRLDRUBLFRBLLFRBBFUUDBFDUB |
| 13 | 0 | RBFRUDBDBLBUURFRFUDLDFFBLDUFRFUDLUURDBRRLDBFDLLFRBUBLL |
| | | RBFRUDBDBLBUURFRFUDLDFFBLDUFRFUDLUURDBRRLDBFDLLFRBUBLL |
| 14 | 6 | DFLRUDRUUBFURRBFFDFLR**U**FB**UD**LRUDBD**LBF**LFUURLLULBBRRDBDBFL |
| | | DFLRUDRUUBFURRBFFDFLR**D**FB**DB**LRUDBD**UFL**LFUURLLULBBRRDBDBFL |
| 15 | 2 | RLFFUUULLBRDBRRDDRFB**D**BFRR**D**LULBUDFFDUBRLLLULFBRFDDBDFBD |
| | | RLFFUUULLBRDBRRDDRFB**U**BFRR**U**LULBUDFFDUBRLLLULFBRFDDBDFBD |
| 16 | 0 | UBFBURBLUBDDBRDLFDUDRLFLUFBFRDFDUDUBRDLULUFLLRRFFBBRRL |
| | | UBFBURBLUBDDBRDLFDUDRLFLUFBFRDFDUDUBRDLULUFLLRRFFBBRRL |
| 17 | 0 | FDBUUULFRBLLBRFRDDFDDRFUUDBRBULDRBLRLRDRLFDFFULUUBBFBL |
| | | FDBUUULFRBLLBRFRDDFDDRFUUDBRBULDRBLRLRDRLFDFFULUUBBFBL |
| 18 | 0 | BULDUBFBLBRFFRUDLRUDUUFDDRFRDLLDFDBBLLRFLRRUBUBDFBRULF |
| | | BULDUBFBLBRFFRUDLRUDUUFDDRFRDLLDFDBBLLRFLRRUBUBDFBRULF |
| 19 | 3 | DULLUULUDRRURRRULUD**DDD**FDFDLUFFDDBDFRFUBFLDFLRBFRBBLBRL |
| | | DULLUULUDRRURRRULUD**BBB**FDFDLUFFDDBDFRFUBFLDFLRBFRBBLBRL |
| 20 | 0 | BRDDUUBBLBFFFRLRLFRLUBFLDDFBFDBDDUBRDRUFLULDLLURUBRURF |
| | | BRDDUUBBLBFFFRLRLFRLUBFLDDFBFDBDDUBRDRUFLULDLLURUBRURF |

Cuadro .3: Errores visión. Esta tabla indica los facelet específicos donde el algoritmo de visión desarrollado se equivocó. En cada fila, el string superior es el resultado obtenido y el inferior el esperado. Las diferencias se muestran subrayadas y en negrita.

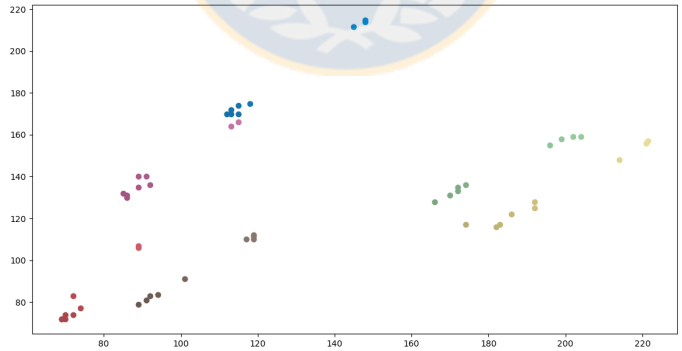
# Proyecciones de colores representantes en RGB




(a) Proyección en ejes rojo (horizontal) y verde (vertical).




(b) Proyección en ejes rojo (horizontal) y azul (vertical).




(c) Proyección en ejes verde (horizontal) y azul (vertical).

Figura .1: Proyecciones. Se aprecia la distribución en rectas de los facelets del mismo color.

# Brazos del Robot

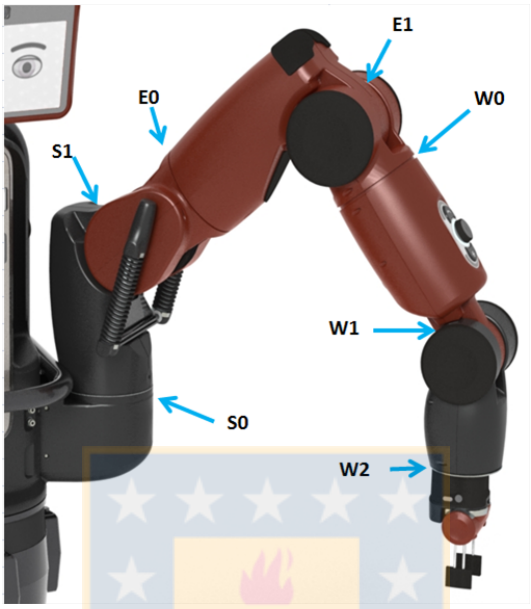



Figura .2: Articulaciones del robot Baxter.

Figura .3: Muñeca del robot.

# Recursos adicionales

1. Vídeo de ejemplo de Robot Baxter resolviendo el cubo. Este corresponde a la secuencia de largo 8 en los experimentos de manipulación. https://www.youtube.com/watch?v=AlMi1rzPEQs
2. Repositorio de código del sistema desarrollado: https://github.com/silvercobraa/memoria-de-titulo
3. WCA Legacy Scrambler: https://www.worldcubeassociation.org/regulations/history/files/scrambles/scramble_cube.htm