MatulaRubik

Nombre y Apellido Radice Adrián – Sanabria Facundo
Asignatura Sistemas Operativos Avanzado – Docentes: Lic. De Luca Graciela – Ing. García Gerardo – Ing. Valente Waldo – Ing. Carnuccio Esteban.
Universidad Nacional de La Matanza – Departamento de Ingeniería e Investigaciones Tecnológicas

radiceadriann@gmail.com - facundosanabria@gmail.com

Kerwords IOT – Rubik – Arduino – Ethernet – Android – JSON – Post – Steeper – Driver – Puente H – DRV8825 – L293 – NDK – OpenCV.

***Abstract*- El objetivo del presente trabajo es desarrollar un sistema de IOT, donde se demuestra el conocimiento adquirido en la cátedra de sistemas operativos avanzados. El propósito del sistema es Resolver el conocido juego de lógica Cubo Rubik 3x3x3, automáticamente.**
Podrán tener acceso al código y detalles de construcción en el repositorio [github.com/BitsellerSOA/MatulaRubik].

I. Introducción

El presente trabajo es una implementación de IOT, el mismo consiste en un SE que tendrá el fin de resolver el cubo Rubik de 3x3x3. El cubo Rubik es un juego que combina lógica con movimientos mecánicos. Existen diversos algoritmos para resolverlo. El elegido en el presente trabajo es el algoritmo CFOP (Cross – F2L – OLL – PLL) o también conocido por el nombre de su autor Fridrich. El algoritmo será aplicado por un equipo celular con Sistema Operativo Android, el cual entregará la secuencia de movimientos a aplicar por la parte mecánica del sistema controlada por un Arduino Mega 2560.

II. Objetivo

Resolver el cubo Rubik 3x3x3, mediante un sistema automático implementando IOT.

III. Limite

Desde que el cubo Rubik de 3x3x3 es analizado por la App Móvil y el SE devuelve el cubo resuelto como fue indicado por la APP.

IV. Alcance

- La App Android *MatulaRubik* analizara el cubo, capturando la posición de los colores con el uso de la librería OpenCV 3.4.1, y entregara en un repositorio web la solución a ser tratada por el SE.
- La App Android *MatulaRubik* registra un historial de logros (Tiempos previos logrados) y configuración del Cubo.
- El sketch de Arduino retirara la solución entregada por *MatulaRubik* del repositorio WEB.
- El sketch de Arduino interpretará la solución del cubo y la traducirá en los movimientos mecánicos.
- El sistema embebido ante desconexión, o reseteo volverá al estado de reposo, no necesariamente informando a la app.

V. Software utilizado

- Android Studio. Para el desarrollo de la app Android.
- Arduino IDE 1.8.5. Para el desarrollo del sketch Arduino.
- Enterprise Architect. Para diagramas y documentación.
- Git. Para el control de versiones del código.
- Fritzing. Para Los diagramas de circuitos.

VI. Hardware utilizado

ELECTRONICA	CANTIDAD
Arduino Mega 2560	1
Capacitor 1000uf	1
Capacitor 100uf	3
DRV 8825	3
L293D	1
Motor DC	1
Shield W5100	1
Stepper	3
Fin de carrera	1
Resistencia	
OTROS	**CANTIDAD**
Cable	
Perfboard	1
Zocalos	1

VII. Materiales utilizados

MATERIAL	CANTIDAD	DIMENSION
Acrílico	-	
Aluminio	-	
Varilla Enroscada	4	
Chapa Galvanizada	-	
Estaño	-	
Madera	-	
Pegamento	-	
Tornillo (Pinzas)	3	
Tornillo (Sujeta retroceso)	4	
Tornillo (Sujeta Steeper)	12	
Tuerca	16	

VIII. Herramientas Utilizadas

- Soldador.
- Taladro.
- Mecha Madera y Metal
- Destornillador.
- Multímetro.
- Alicate.

IX. Detalles componentes

Arduino Mega 2560

Objetivo en el proyecto: Se encargará de efectuar los movimientos mecánicos que implican resolver el Puzzle.

Características:

CARACTERISTICAS ELECTRICAS	
Microcontrolador	ATmega 2560
Tensión de operación	5V
Tensión de alimentación	6-20V recomendado 7-12V
Pines digitales	54 (15 proveen salida PWM)
Pines Analógicos	16
Corriente por Pin I/O	20mA
Corriente Pin salida 3.3V	50mA
Memoria Flash	256KB – 8KB Bootloader
Clock	16 MHz
DIMENSIONES	
Largo	101.52 mm
Ancho	53.3 mm

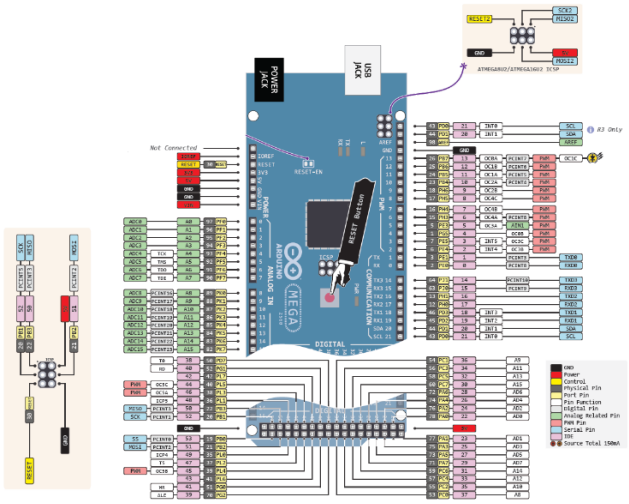


Fig. 1. Mapa de pines Arduino.

SHIELD W5100

Objetivo en el proyecto: permite la comunicación entre el Arduino y el celular. No se usa del shield las prestaciones para SD.

Pin Arduino: 50 – 51 – 52 – 53 para la comunicación SPI.

Características

- Tensión de alimentación: utiliza 5V suministrados por el Arduino.
- Buffer interno de 16k.
- Permite comunicaciones 10/100Mb.
- Indicadores Led
  - TX
  - RX
  - LINK
  - 100M
  - ON

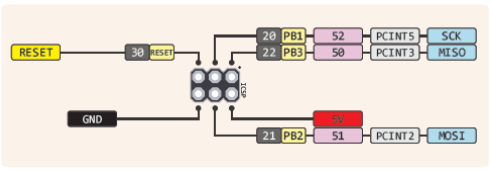


Fig. 2. Diagrama que indica que los pines del 50-53 están imposibilitados para su uso.

FIN DE CARRERA

Objetivo en el proyecto: indicar cuando el motor DC alcanzo la posición deseada.

Configuración: NC con resistencia Pull Down.

Señal: Digital.

Estados: High cerrado, Low Abierto. Por configuración NC.

Pin Arduino: Pin 22 configurado como entrada.

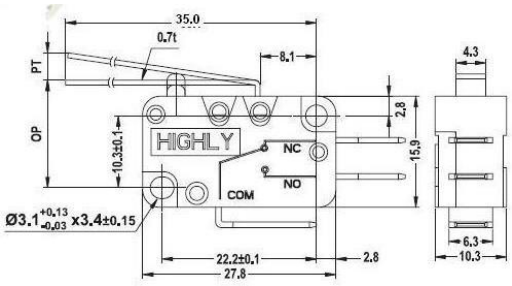


Fig. 3. Diagrama físico del sensor fin de carrera.

Iteración con Arduino: Para simplificar su descripción utilizaremos el autómata descripto en la Fig 1.

- El estado 1 implica el Motor DC en reposo.
- El estado 2 indica el motor DC excitado.
- La transición q1 se da cuando el Arduino interactúa con el puente H y este polariza el motor DC para que empiece a moverse.
- La transición q3 implica la lectura Low en el pin digital en el que está conectado el fin de carrera.
- La transición q2 implica la lectura HIGH en el pin digital en el que está conectado el fin de carrera.

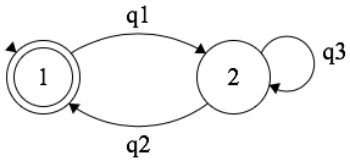


Fig. 4. Máquina de estados iteración Arduino fin de carrera.

Iteración Mecánica: Dado que existen retardos para pasar del estado 2 al 1, la chapa del fin de carrera respecto a la posición deseada se separa una distancia D Fig 1. Los retardos incluyen desde que se recibe el nivel alto en el Arduino, este envía la orden de detención, el DC se detiene y se compensa el empuje.

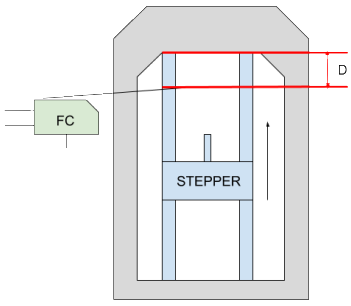


Fig. 5. Representación física de la iteración. No está a escala.

RESISTENCIA

Objetivo en el proyecto: Completar circuito para el Fin de carrera. La misma permite que no se produzca un corto al cerrarse el circuito.

Valor: 10kΩ

Estados: High cerrado, Low Abierto. Por configuración NC.

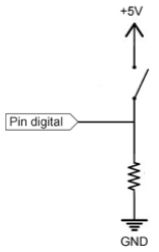


Fig. 6. Diagrama lógico de configuración pull down.

CAPACITOR

Objetivo en el proyecto: Proteger a los DRV8825 que son muy sensibles a picos de tensión. Estas variaciones de tensión principalmente aparecen cuando los motores actúan.

Tipo: Electrolítico.

Valores: el fabricante del drv8825 recomienda capacitores con un valor mínimo de 47uf, dado a nuestra disponibilidad usamos de 100uf.

PERFBOARD

Objetivo en el proyecto: permite el montaje de los circuitos comentados en FIG 15. La perfboard es una placa de circuito perforada cuyos huecos están circundados por material conductor, usualmente cobre, pero que no están interconectados entre sí. Este tipo de placas requieren que cada componente esté soldado a la placa y además las interconexiones entre ellos sea realizada a través de cables o caminos de soldadura.

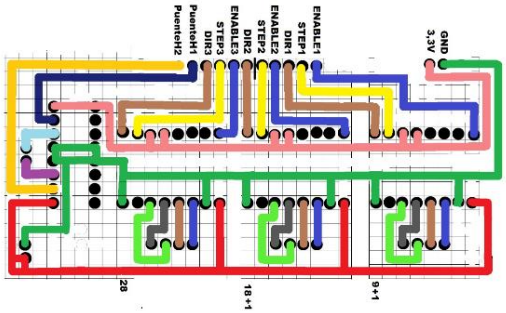


Fig. 7. Caminos de soldadura que fueron realizados en la perfboard.

CABLE SIMPLE

Objetivo en el proyecto: conductores eléctricos. Se utilizan para interconectar los motores al driver, y en la alimentación del sistema embebido.

CABLE ETHERNET

Objetivo en el proyecto: Permite la conexión del SE a una Red.

Características: UTP CAT 5e Norma T568, Cruzado.

MOTOR DC

Objetivo en el proyecto: el retroceso y avance del brazo que mueve la cara lateral.

Características: Tención de alimentación 9V.

Sentido de giro: Controlado por el puente HL293D.

PUENTE H L293D[1]

Objetivo en el proyecto: Control de giro del Motor DC que desplaza el brazo lateral.

Características:

CARACTERISTICAS ELECTRICAS	
Tensión lógica	5V
Tensión de salida (Usada)	9V
Corriente de salida	600 mA
Admite PWM	SI

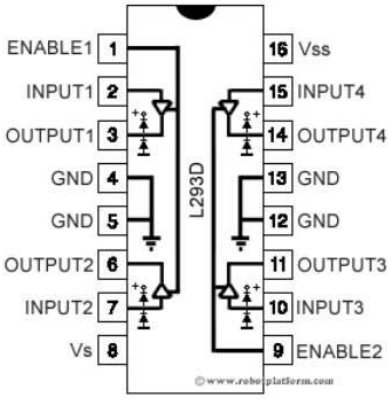


Fig. 8. Pines Integrado L293D

Pin Arduino: Pin 38 y 39 configurado como salida.

Iteración con Arduino:

ENABLE	**INPUT 1**	**INPUT 2**	**Motor**
LOW	HIGH/LOW	HIGH/LOW	PARADO
HIGH	HIHG	LOW	MOV 1
HIGH	LOW	HIGH	MOV 2

Tab. 1. En el proyecto solo se trabaja con las dos últimas filas. Mov 1 y Mov 2 representaran en el proyecto un movimiento de izq a derecha o de derecha a izquierda.

MOTOR STEPPER BIPOLAR

Objetivo en el proyecto: se encarga de rotar las caras del cubo con pasos de $\pi/2$ y múltiplos del mismo.

[1] Para el proyecto no es necesario un puente H doble, se utilizó el L293D porque ya se lo poseía.

Características:

CARACTERISTICAS ELECTRICAS	
Step Angle	1.8 deg (200 pas por vuelta)
Tensión	4,83 V
Corriente	0.84 A
Torque	2.8 Kg/cm

Movimiento: Controlado por el DRV8825.

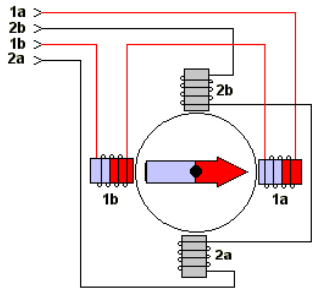


Fig. 9. Esquema de un motor bipolar. Se denominan bipolares porque de acuerdo al sentido de la corriente se forma un campo magnético S/N o N/S.

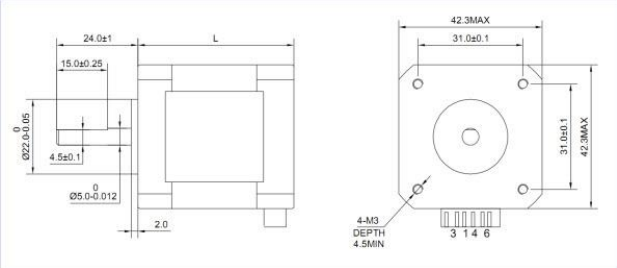
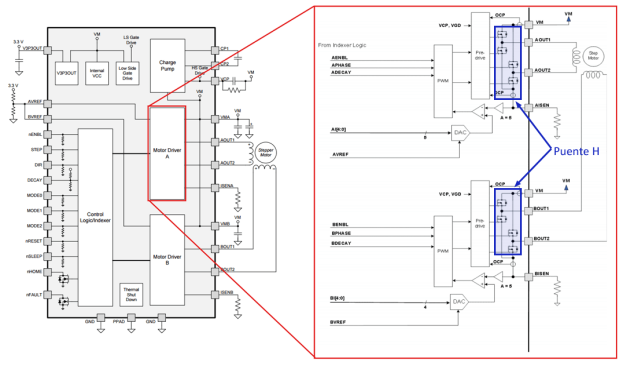


Fig. 10. Dimensiones físicas del motor utilizado.

DRV8825

Objetivo en el proyecto: Control de los motores paso a paso del circuito. En sencillas palabras actúan como una válvula que separa la corriente que alimenta al Arduino de la que es necesario hacer circular por el motor. Este driver particularmente esta compuesto por dos puente H que utiliza para cambiar la dirección de la corriente en las bobinas del motor bipolar que controla.



.

Fig. 11. Composición del DRV8825.

Características:

CARACTERISTICAS ELECTRICAS	
Intensidad máxima	2.5A a 24V
Tensión máxima	8.2V a 45V
Tensión de pines lógicos de entrada	2.25V a 5.5V

Pin Arduino: Pines 24, 28, 33 para enable, 25,30 y 34 para step y 26, 31 y 36 para dirección.

Iteración con Arduino:

El Arduino envía el pulso (STEP). Cada pulso le dice al motor que tiene que avanzar un paso. Otro pin nos dice la dirección de avance (DIR). Si queremos ir en el sentido de las agujas del reloj, o al contrario. De esta forma, cada vez que el driver recibe un pulso en el pin STEP, el circuito comprueba el voltaje del pin DIR, y alimenta las bobinas del motor en el orden adecuado.

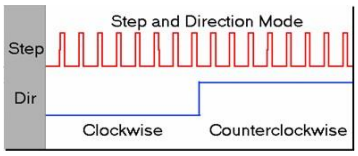


Fig. 12. Señal enviadas al Driver.

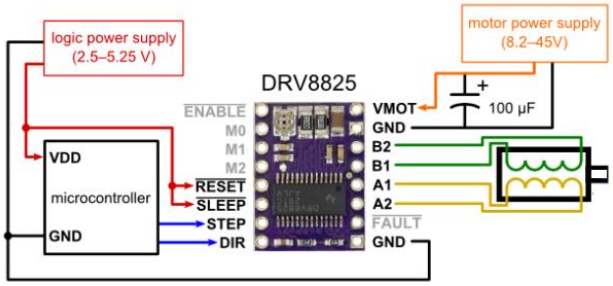


Fig. 13. Diagrama de bloques del DRV8825.

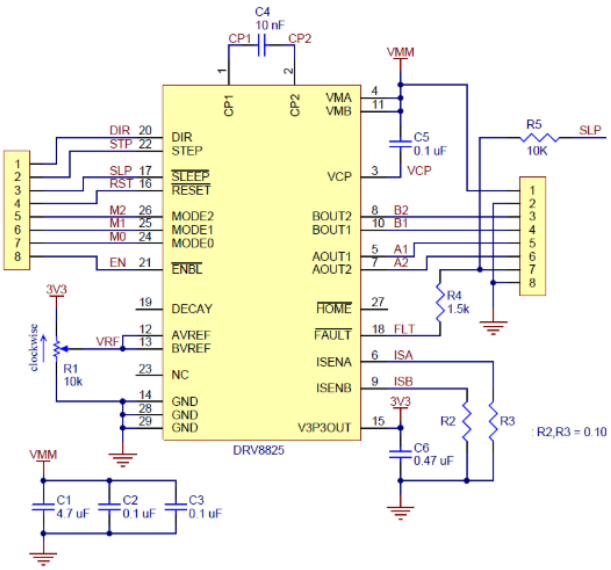


Fig. 14. Diagrama lógico del DRV8825

X. CIRCUITOS

CIRCUITO PARA CONTROL DE MOTOR STEPPER.

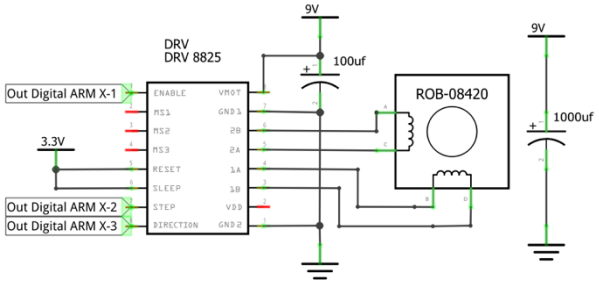


Fig. 15. Diagrama lógico para control de motores Stepper. Este circuito se implementa 3 veces para controlar cada uno de los motores. El circuito completo podrá ser consultado al final de este documento.

Etiqueta	Pin Arduino
MOTOR U	
X-1	24
X-2	25
X-3	26
MOTOR D	
X-1	28
X-2	30
X-3	31
MOTOR L	
X-1	33
X-2	35
X-3	36

BREVE EXPLICACIÓN DEL CIRCUITO:

El objetivo de este circuito es el control de un motor paso a paso, por lo tanto, tendremos 3 en total. Los motores pasos a paso tienen como objetivo principal efectuar los movimientos sobre las caras del cubo. Los movimientos a efectuar son giros de 90 grados en sentido horario/antihorario o múltiplos del mismo. El DRV será el intermediario entre el paso a paso y el Arduino. El capacitor de 100uf observador en la figura fue recomendado por el fabricante como fue especificado en el apartado Detalles de componentes Capacitor.

CIRCUITO PARA CONTROL DE MOTOR DC.

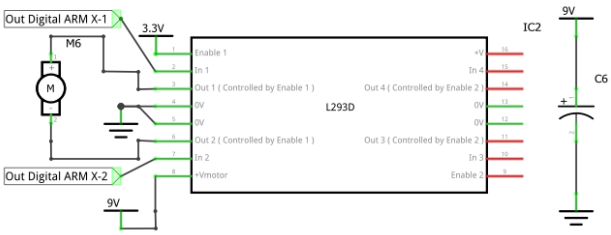


Fig. 16. Diagrama lógico para control de motor DC. El circuito completo podrá ser consultado al final de este documento.

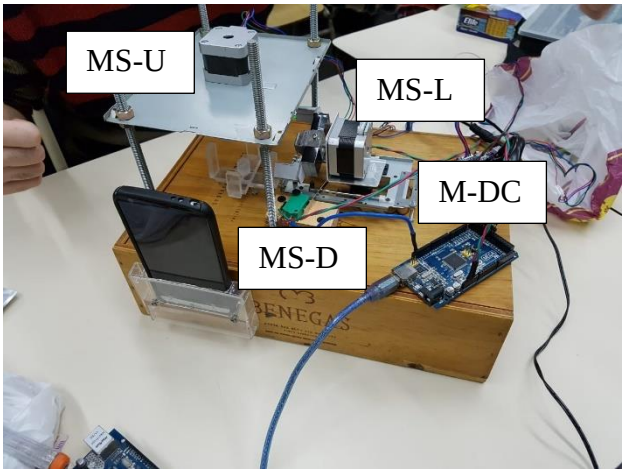
Etiqueta	Pin Arduino
MOTOR DC	
X-1	52
X-2	53

BREVE EXPLICACIÓN DEL CIRCUITO:

El objetivo de este circuito es permitir un desplazamiento horizontal de uno de los brazos. Un movimiento horizontal puede ser de izquierda a derecha o de derecha a izquierda. Para lograr dicho movimiento, si el objeto que queremos mover interactúa con un tornillo sin fin, bastaría con cambiar el sentido de giro de un motor. El motor que interactúa en esta parte del circuito es un DC convencional, por lo tanto, su sentido de giro queda determinado por cómo es polarizado. Para paralizar de dos formas distintas al motor utilizamos un puente H, en este caso uno de los dos que provee el L293D. Los 3.3v que son utilizados para habilitar el L293D serán suministrados por el Arduino. Los 9v empleados para alimentar el motor DC serán suministrados por la fuente externa que alimenta a todo el proyecto. X-1 y X-2 enviaran los pulsos necesarios para lograr la codificación digital (Tab. 1) que requiere el puente H.

XI. CONSTRUCCIÓN MECÁNICA

MAQUINA:



Motores	**Movimiento**
MS-U	U
MS-D	D
MS-L	{ F, L , R , B }
M-DC	RETROCESO M-L

{U, D, F, L, R, B} Son las caras del cubo Rubik, en introducción sobre CFOP se explica la nomenclatura.

PINZAS:

Para sujetar las pinzas a los ejes del motor paso a paso, se utilizó el concepto de las borneras.

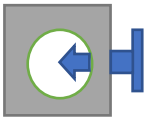


Fig. 16. En azul se simboliza al tornillo que presionara el borde plano del eje del motor paso a paso.

XII. INTRODUCCIÓN A CFOP

Existen muchos algoritmos para poder resolver el cubo Rubik, en este trabajo se eligió a CFOP o también conocido por el nombre de su autor Fridrich. El motivo fue porque era el que conocíamos. El algoritmo como indica en su sigla se compone de 4 etapas Cross, F2L, OLL, PLL.

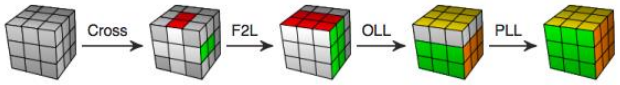


Fig. 18. Automta que representa las etapas principales del Algoritmo

El algoritmo tiene una nomenclatura especifica la cual fue adoptada para nuestro proyecto con ciertas modificaciones.

CARAS (C)	
Superior (Up)	U
Inferior (Down)	D
Trasera (Back)	B
Derecha (Right)	R
Izquierda (Left)	L
FRONTAL (Front)	F
Movimientos	
Horario 90*	(C)
Horario 180*	(C)2
Antihorario 90*	(C)’
Antihorario 180*	(C)’2

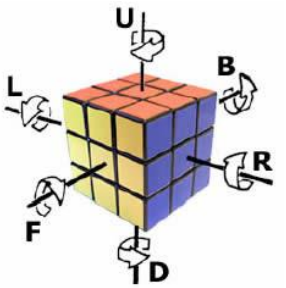


Fig. 19. Nomenclatura. Notar que en la nomenclatura se emplea un apostrofe para los movimientos antihorarios, que en el código será remplazado por una ‘A’ quedando por ejemplo FA, UA, etc. Se considera sentido horario respecto a mirar la cara de frente. Los movimientos de 180 grados fueron efectuados como dos consecutivos de 90 grados.

Para el desarrollo del paper, consideramos que empezamos armando la cara blanca. A continuación se detalla cada uno de los estados de las distintas etapas, y la secuencia de movimientos que implican cambiar de estado.

CROSS

Objetivo: Armar cruz en la cara de partida.

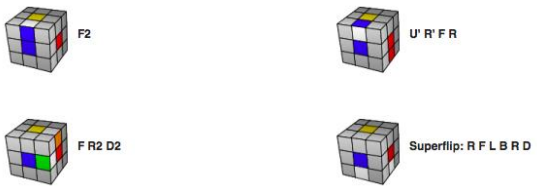


Fig. 20. Estados inicial y transición para salir del mismo.

F2L FIRST TWO LAYER

Objetivo: Ubicar esquinas de cara inferior, y bordes de cara intermedia. Se pueden dar 41 casos.

Fig. 21. Estados inicial y transición para salir del mismo.

OLL ORIENTATION OF THE LAST LAYER

Objetivo: Orientar piezas de capa superior sin importar su ubicación.


R U B' I U I2' x' U' R' F R F'


R' F R F' U2 R' F R y' R2 U2 R


y L' R2 B R' B L U2' L' B M'


R' U2 x R' U R U' y R' U' R' U R' F


(R U R' U) R' F R F' U2 R' F R F'


M' U2 M U2 M' U M U2 M' U2 M


R' U2 F (R U R' U') y' R2 U2 x' R U


F (R U R' U) y' R' U2 (R' F R F')


R' U' y L' U L' y' L F L' F R


R U' y R2 D R' U2 R D' R2 d R'


F U R U' R' U R U' R' F'


L' B' L U' R' U R U' R' U R L' B L


L U' R' U L' U (R U R' U) R


(R U R' U) R U' R' U R U2 R'


L' U R U' L U R'


R' U2 (R U R' U) R


R' F' L F R F' L' F


R2 D R' U2 R D' R' U2 R'


R' F' L' F R F' L F


M' U' M U2' M' U' M


L' (R U R' U') L R' F R F'


L F R' F R F2 L'


F R' F' R U R U' R'


R' U' R y' x' R U' R' F R U R'


U' R U2' R' U' R U' R2 y' R' U' R U


F (R U R' U') (R U R' U') F'


L F' L' F U2 L2 y' L F L' F


U' R' U2 (R U R' U) R2 y (R U R' U') F'


r U2 R' U' R U' r'


R' U2 l R U' R' U l' U2 R


F' L' U' L U L' U' L U F


R' F R' F' R2 U2 x' U' R U R'


R' F R F' U2 R2 y R' F' R F'


R U R' y R' F R U' R' F' R


L' B' L U' R' U R L' B L


U2 r R2' U' R U' R' U2 R U' M


x' U' R U' R2' F x (R U R' U') R B2


L U' y' R' U2' R' U R U' R U2 R d' L'


U2 l' L2 U L' U L U2 L' U M


R2' U R' B' R U' R2' U l U l'

r' U2 (R U R' U) r

R U x' R U' R' U x U' R'

(R U R' U') x D' R' U R E'

R' F R U R' F' R y L U' L'

L F' L' U' L F L' y' R' U R

L' B' L R' U' R U L' B L

R B R' L U L' U' R B' R'

F U R U' R' F'

R' d' L d R U' R' F' R

L d R' d' L' U L F L'

F' U' L' U L F

F (R U R' U') F'

(R U R' U') R' F R F'

L U L' U L U' L' U' y2' R' F R F'

R' U' R U' R' U R U y F R' F' R

R' F (R U R' U') y L' d R

L F' L' U' L U y' R d' L'

Fig. 22. Estados inicial y transición para salir del mismo.

PLL Permutation of the Last Layer

Objetivo: Ubica en la posición correcta las piezas de la capa superior.

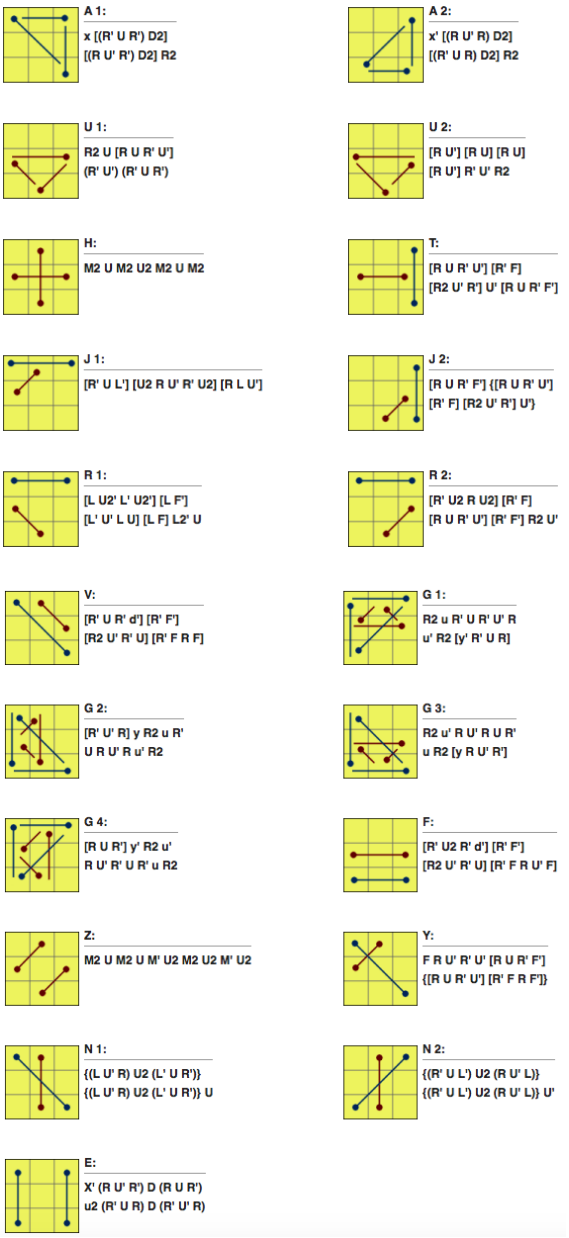


Fig. 23. Estados inicial y transición para salir del mismo.

Almacenamiento del cubo

La ubicación de las artistas y esquinas del cubo se almacenan en una matriz de 6*9 del tipo char. Con la siguiente distribución.

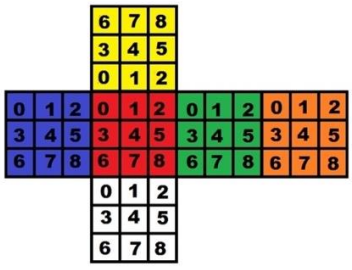


Fig. 24. Estados inicial y transición para salir del mismo.

XIII. Comunicación App y Arduino

RESUMEN MENSAJES JSON INTERCAMBIADOS

- OK
- ERROR
- DESCONEXION
- SOLUCION LISTA
- COLOCAR CUBO
- {R, L, D, U, B, F, X, Y, Z, V, W, N} Representan Movimientos.

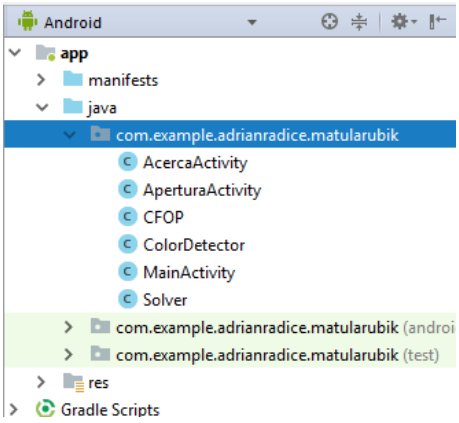
PSEUDOCODIGO

1 La App verifica si existe el Arduino.
1.1 La App envía un mensaje a la Dirección IP del Arduino.
1.2. La app Espera respuesta del Arduino por un intervalo de tiempo determinado.
1.3. El Arduino ante la solicitud del celular puede:
1.3.a No responder.
1.3.a.1 Se vence el plazo de tiempo en la app y emite un mensaje que dice Arduino no responde.
1.3.b Estoy Ocupado
1.3.b.1 La app informa el estado del Arduino.
1.3.c Estoy disponible
2 Si La app determina la disponibilidad del Arduino, verifica tener el cubo frente a su cámara.
2.1 Si la respuesta anterior es negativa solicita que se coloque un cubo.
2.2 Si la respuesta anterior es positiva escanea la primera cara.
2.3 La app Envía un post a Arduino indicando que mueva el motor de abajo y superior en sentido horario.
2.4 Arduino ejecuta el movimiento y informa a la app.
2.5 La app escanea la cara.
SE REPITE 3 VECES LA SECUENCIA DEL 2.4 AL 2.5
3 La app notifica que se mueva el motor lateral mediante un post para escanear la cara inf.
3.1 Arduino recibe la solicitud ejecuta la combinación de movimiento motor lateral y luego inferior y superior para que la cara inferior quede frente la cámara
3.1 La app escanea la cara
La app notifica que se mueva el motor inf y sup para que la cara sus quede frente a la cámara
4.1 Arduino recibe la solicitud ejecuta el movimiento
4.2 La app escanea la cara
La app Aplica el algoritmo para resolver, mientras tanto Arduino espera el post de la app indicando la disponibilidad de la solución para poder descargarla y aplicarla.
La app espera que Arduino indique que termino de aplicar la secuencia de movimientos y notifica a la aplicación que finalizo.
APARTIR DE PASO 5 NO ES NECESARIO EL CELULAR EN LA BASE DE LA MAQUINA

XIV. Diagramas App

- Activity
  - Acerca => Información de la aplicación
  - Apertura => Splash de bienvenida
  - Main => principal
- Clases
  - CFOP => Algoritmo de resolución
  - ColorDectector => Reúne métodos para detectar lo colores con la cámara.

- Solver => Reúne los métodos para resolver el cubo



XV. Requisitos App

Requisito	
OS	> = **Android 6.0**
Cámara	**SI**
Uso de Red	**SI**
Memoria	**<COMPLETAR>**

XVI. Introducción a OPENCV y NDK

OPENCV es una librería de visión artificial de uso libre, desarrollada en C/C++ lo cual para poder ser utilizada por Android debemos recurrir a la cross compilación ofrecida por NDK. De los módulos ofrecidos por OPENCV emplearemos el CORE (Incluye las estructuras de datos básicas y las funciones básicas de procesamiento de imágenes) y el OBJDETECT.

Para la detección de colores utilizaremos el concepto de máscaras y el espacio de colores HSV. Cada mascará definirá un rango de valores máximo y mínimo. Tendremos una máscara por color excepto por el rojo que dado a la ubicación del mismo en el campo de HSV requiere dos, porque tiene dos rangos Fig.5.

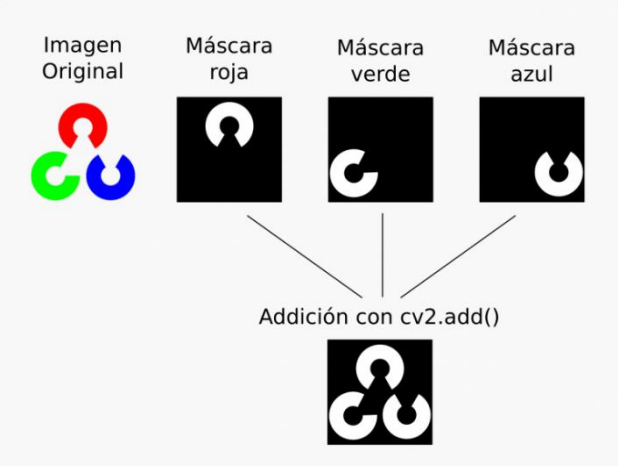


Fig. 26. Funcion add OpenCv.

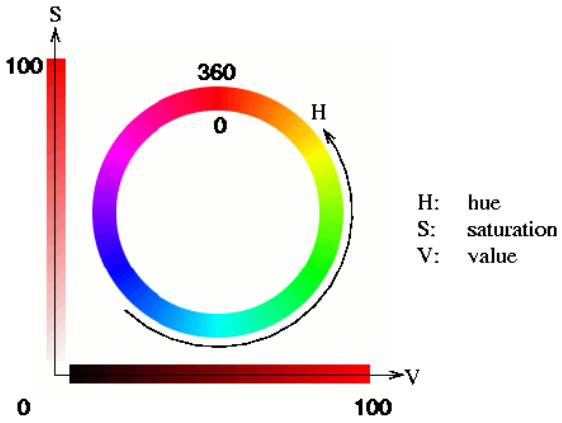


Fig. 27. Se puede apreciar en la imagen que el Rojo define dos rangos por la componente H, uno del 0 hasta estar próximo al amarillo, y otro desde las fronteras con el violeta hasta el 0.

XVII. Implementando HPC

Consultar https://github.com/BitsellerSOA/MatulaRubik/tree/master/TP%20HPC/HPC.pdf

XVIII. Pruebas

Pruebas algoritmo de resolución

Dado que son extremadamente grandes (eq.1) todas las configuraciones que se pueden obtener al mezclar el cubo Rubik, sería imposible querer probar que el algoritmo puede resolverlas a todas. Por lo tanto, consideraremos que resolver el cubo seria la integración de muchos métodos, que podrán ser probados unitariamente y luego mediante una prueba de integración podremos afirmar que si logra resolver 10 escenarios distinto el algoritmo será aceptable.

Las pruebas unitarias que deberá enfrentar el algoritmo podrán ser agrupadas en 3 etapas a ejecutar en orden:

- Realiza los movimientos simples correctamente.
- Resuelve correctamente un paso del algoritmo CFOP.
- Identifica correctamente que secuencia de movimientos se debe aplicar.

La primera etapa es fácil de probar, el objetivo es que haga los desplazamientos correctamente en la matriz. Por lo tanto se tendrá una prueba para cada posible movimiento del cubo { U, U', D, D', L, L', R, R', B, B', F, F'}. Se partirá siempre del cubo ordenado (matriz Fig. 5) y se comparará con el resultado esperado.

La segunda etapa tiene como objetivo probar a cada una de las secuencias de CFOP. Se entrega la matriz en el estado esperado y luego se compara con la matriz obtenida, para identificar si la secuencia es correcta.

La tercera etapa, probara el método que determina en qué estado se encuentra el cubo, para ello se entrega el cubo desordenado y se compara con el estado esperado.

Superada las tres etapas anteriores consideramos que el algoritmo será óptimo para pasar a la prueba de integración. Donde se entregará 10 configuraciones distintas del cubo con distintos grados de complejidad, y se probará que la matriz quede en el estado de cubo resuelto.

Pruebas identificación de colores

Dada las características de esta operación, esta prueba no puede ser automatizada, por lo tanto, se realizará por inspección visual. Se entregan por lo menos 10 distintas configuraciones random del cubo y se observa que los colores sean correctamente identificados por la aplicación. Los colores mostrados por la app deberán coincidir con los reales.

Pruebas Inspección de colores

Dada las características de esta operación, esta prueba no puede ser automatizada, por lo tanto, se realizará por inspección visual. El objetivo de esta prueba es la correcta sincronización entre la app y el Arduino. Se probará que el intercambio de confirmaciones entre ambos, en donde ni la app deberá inspeccionar colores hasta que Arduino se lo confirme, así como el Arduino no efectué movimientos hasta que la app se lo indique.

Pruebas desconexión durante inspección

Dada las características de esta operación, esta prueba no puede ser automatizada, por lo tanto, se realizará por inspección visual. El objetivo de esta prueba es la correcta sincronización entre la app y el Arduino. Ante la pérdida de comunicación entre la app y el Arduino, la app deberá informar por pantalla, y el Arduino quedará en estado de reposo. La pérdida de comunicación se evaluará de la siguiente forma.

- Desconexión del Arduino durante la inspección.
- Reseteo del Arduino durante la inspección.
- Cierre de la app durante la inspección.
- Desconexión del celular de la red.

Pruebas movimiento Arduino

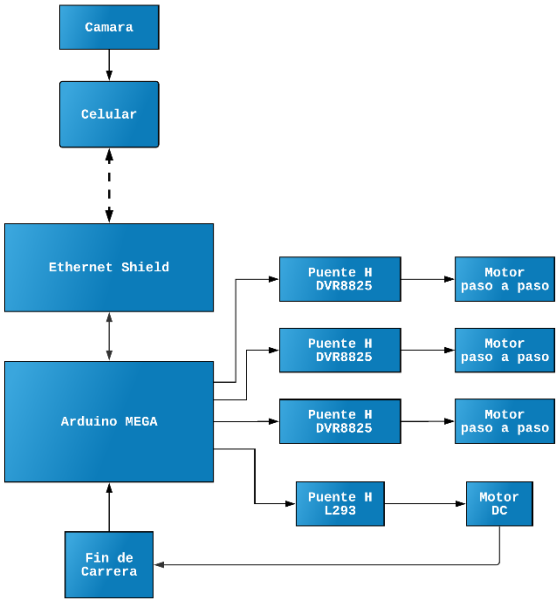
Desconocemos técnicas de pruebas automatizadas para hardware, por lo tanto, usaremos la inspección visual. Se entregará al SE el cubo y se probará cada una de las funciones del sketch que representa un movimiento, si lo aplica correctamente. Las funciones a probar son:

- u()
- d()
- f()
- r()
- l()
- b()
- uA()
- dA()
- fA()
- rA()
- lA()
- bA()
- mover(char c)

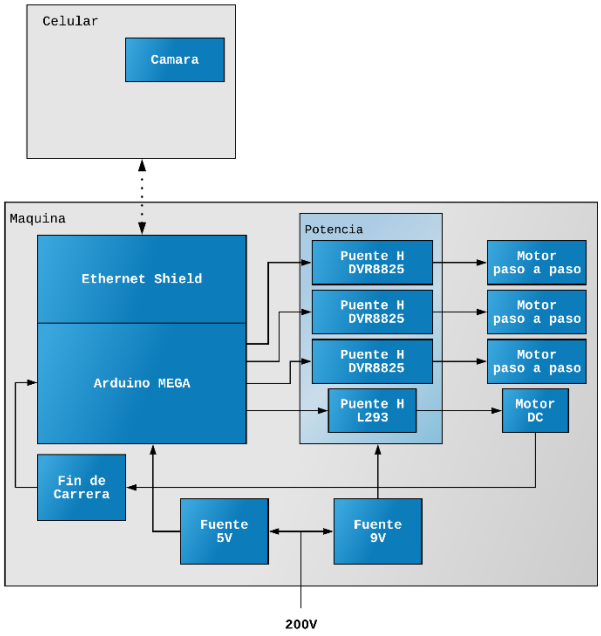
Pruebas JSON Arduino

Se probará que se realice correctamente el parcing, es decir se envía un paquete desde la app, se la recibe con el Arduino y mediante la impresión por puerto serial se observa si es capaz de obtener el movimiento solicitado, o la orden de continuar o cualquiera de los mensajes que previamente definimos.

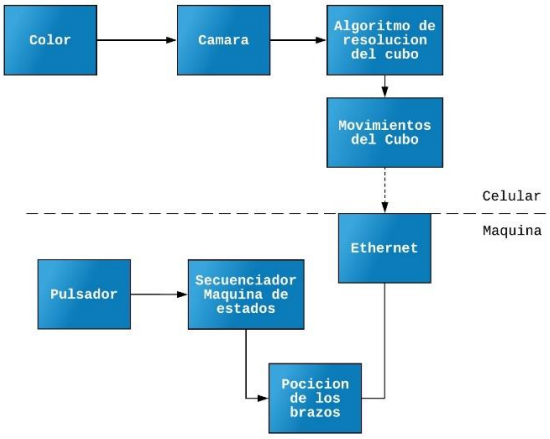
XIX. Diagrama funcional



XX. Diagrama físico



XXI. DIAGRAMA LÓGICO



XXII. ANEXO CÁLCULOS

$$Cant\ config = \frac{8! * 3^8 * 12! * 2^{12}}{12} \qquad \text{(eq.1)}$$

XXIII. ANEXO MATULAREMOTE

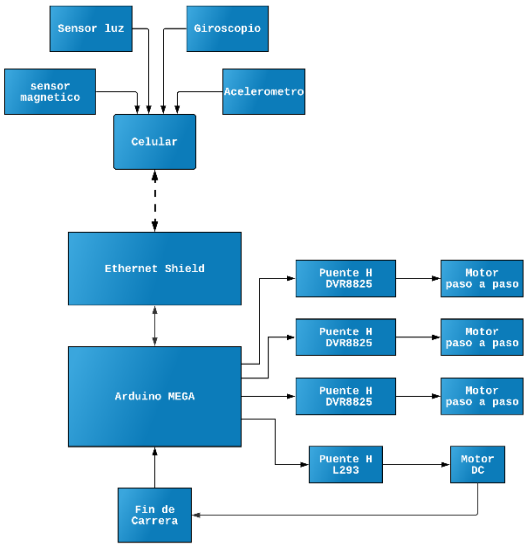
I. OBJETIVO

Manipular un cubo Rubik de 3x3x3, mediante un sistema embebido controlado remotamente desde una aplicación móvil, implementando IOT.

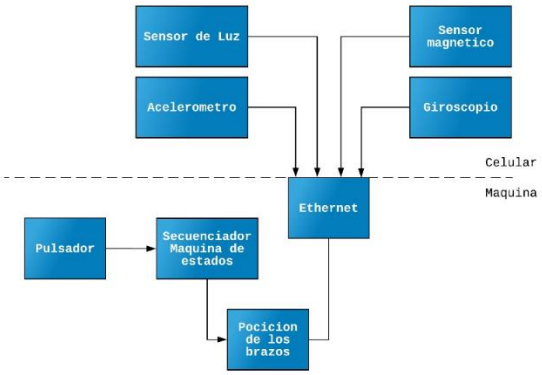
II. LIMITE

Desde que el cubo Rubik de 3x3x3 se ubica en el sistema embebido y la aplicación Android se comunica con el SE y el mismo traduce los mensajes en movimientos mecánicos.

III. DIAGRAMA FUNCIONAL

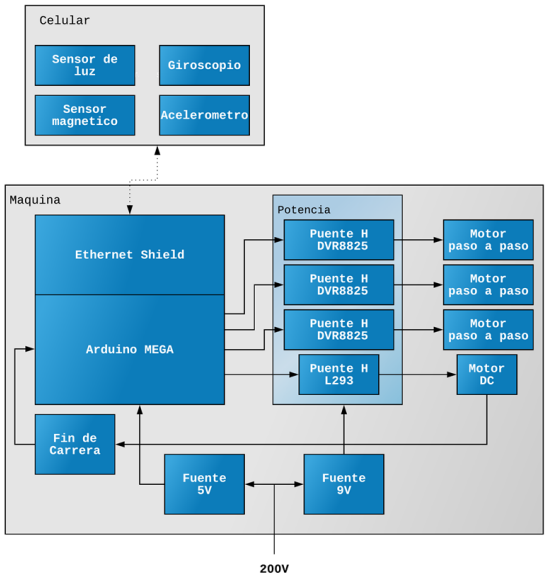


IV. DIAGRAMA LÓGICO



V.

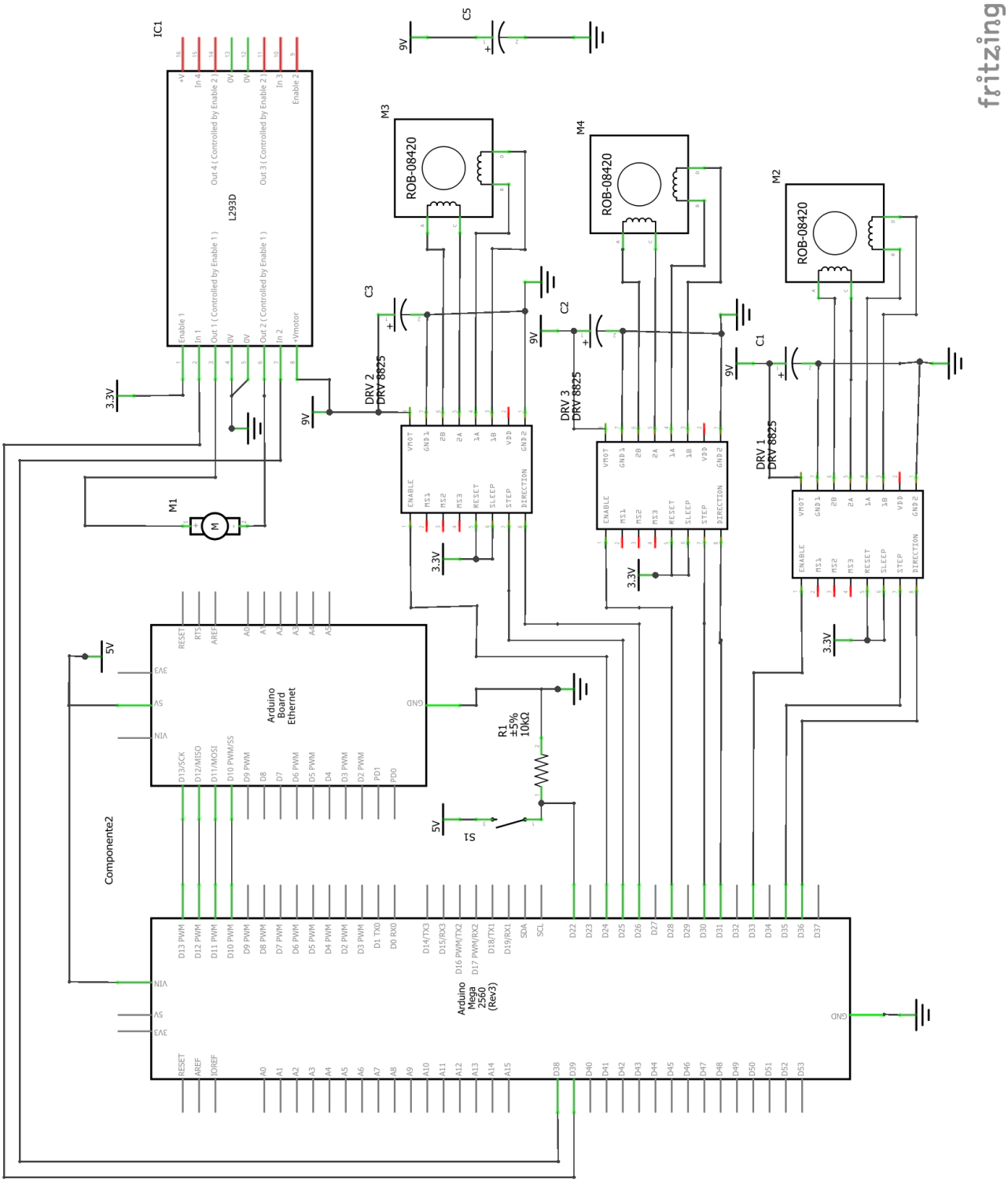
VI. DIAGRAMA FÍSICO



VII. SOFTWARE

- Arduino
  - Control general y mensajería
    - Principal.ino
  - Traducción en movimientos
    - Cubo.h
    - Cubo.cpp
- Android
  - Máquina de estado
    - Estados => CLASS
  - Configuraciones
    - SettingActivity. => CLASS
  - Ventana Principal
    - MainActivity => CLASS
  - Splash Bienvenida
    - AperturaActivy => CLASS
  - Info de la app
    - AcercaActivity => CLASS

CIRCUITO GENERAL



AUTORES

Radice AdrianArgentino, estudiante de ingeniería en informática en la Universidad Nacional de La Matanza.
Profesional dedicado a sistemas informaticos.
Usr GitHub: adrianRadice

Sanabria FacundoArgentino, estudiante de ingeniería en informática en la Universidad Nacional de La Matanza.
Profesional dedicado a sistemas informaticos.
Usr GitHub: FockaSanabria