UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DISEÑO E IMPLEMENTACIÓN DE SISTEMA DE CONTROL PARA ROBOT OPEN SOURCE TIPO SCARA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

FELIPE GERMÁN GUAJARDO MENA

PROFESOR GUÍA:
ANDRÉS CABA RUTTE

COMISIÓN EVALUADORA:
PATRICIO MENDOZA ARAYA
FRANCISCO CASADO CASTRO

Este trabajo ha sido financiado por Beauchef Proyecta, parte de Ingenieria y Ciencias 2030

SANTIAGO DE CHILE
2020

RESUMEN DE LA MEMORIA PARA OPTAR AL
TÍTULO DE: Ingeniero Civil Eléctrico
POR: Felipe Germán Guajardo Mena
FECHA: 15/06/2020
PROFESOR GUIA: Andrés Caba

DISEÑO E IMPLEMENTACIÓN DE SISTEMA DE CONTROL PARA ROBOT OPEN SOURCE TIPO SCARA

En el presente trabajo se muestra el diseño de un sistema eléctrico y de control para un robot *Open Source Hardware* tipo SCARA. El trabajo es parte de un proyecto conjunto entre ingeniería mecánica e ingeniería eléctrica con el fin de llevar a cabo la construcción y la operación del robot diseñado. Todo el material generado en este proyecto está libre en internet para quien desee replicarlo.

El sistema eléctrico diseñado se basa en motores de corriente continua sin escobillas (BLDC) utilizados en drones, los que tienen una alta densidad de potencia y torque máximo constante en casi todo su rango de operación. Estos motores son operados por una placa de circuito *Odrive*, la que permite mover los motores de forma controlada con alta precisión mediante un lazo de control interno. La combinación de estos motores con la placa es una solución costo-eficiente. El computador que organiza todo es una *Raspberry Pi*, la que corre un programa con una clase desarrollada a la medida para el funcionamiento del robot.

Para generar simulaciones del desempeño del brazo robótico se utiliza ROS, la plataforma mas utilizada para hacer desarrollos de robótica. Se utiliza el diseño en CAD creado por el memorista de ingeniería mecánica para generar un URDF, un formato de archivo compatible con ROS. Utilizando este se logran hacer simulaciones. Esta metodología facilita el posterior desarrollo del robot. Mediante optimización por enjambre de partículas se encuentran ganancias de los controladores que reducen el error cuadrático medio para curvas suaves generadas con un algoritmo en el estado del arte.

A Claudia, mi hija,
para quien quiero hacer un mundo mejor
y quien ha sido mi motor para seguir adelante.

Agradecimientos

Quiero agradecer a mis amigos, en especial a Andrés e Ismael, con quienes comparto muchas caídas y levantadas, mas aun sueños, que se embarcaron en este desafío sin pensarlo dos veces. A Esteban, Vicente y Roberto inseparables amigos de tantas trasnochadas y duros semestres. A Olgu, mi hogar y compañera de aventuras. A tantos otros amigues de los que he aprendido durante esta etapa de mi vida.

Doy gracias a Beauchef Proyecta, oficina que nos brindo el apoyo para realizar este proyecto. A su coordinador Francisco Casado y a mi profesor guía, Andrés Caba por darme la oportunidad de hacer un trabajo tangible y divertido, por siempre hacerle frente al paradigma de como se hacen las cosas en la Universidad.

Por último, a mi familia, a mi hermana Mariela, la ingeniera mas responsable y profesional que conozco y a mi hermano Joaquín que la vida le enseña de la misma forma que a mi. Eternas gracias a mis padres, María y Germán quienes me enseñaron a amar y no dudaron en sacrificar lo que fuera necesario para que pudiera ser lo que soy, Feliz.

Tabla de contenido

Capítulo 1

Introducción

"Si las maquinas producen todo lo que necesitamos, el resultado dependerá de cómo se distribuyan las riquezas. Todos podrían disfrutar de una vida de lujoso tiempo libre si la riqueza producida por las maquinas fuese compartida, o por el contrario, la mayoría de las personas podrían llegar a ser miserablemente pobres si los dueños de las maquinas conspiraran con éxito en contra de la redistribución de las riquezas. Hasta el momento, con la tecnología avanzando hacia la creciente desigualdad, la tendencia parece ser la segunda opción." Stephen Hawking

1.1. Motivación

En el tiempo en el que se ha desarrollado este trabajo de memoria, en el país ha habido un estallido social causado por la profunda desigualdad en la que vivimos en Chile, el país mas desigual de la OCDE [1]. Hace eco profundo en este contexto la cita del inicio de este capitulo con Chile como ejemplo, donde el desarrollo económico ha agudizado la desigualdad del país. Estando en las puertas de la cuarta revolución industrial es necesario cambiar este paradigma del progreso, la tecnología debe ser abierta y accesible a las personas de forma que el capital monetario no sea una barrera para su implementación.

Actualmente en el mundo se está cambiando el paradigma sobre el avance tecnológico, algunas iniciativas que van en esta linea son por ejemplo el movimiento de *free-software* (software-libre), una filosofía en la que el usuario tiene acceso al código fuente del programa y este mejora en la medida en que los usuarios hacen mejoras a este, de forma que se fomenta la solidaridad y cooperación entre los usuarios. De forma homóloga existe un movimiento llamado *Open source Hardware (OSH)* cuya ideología es que los diseños de las cosas deben ser libres para la modificación y mejoras de forma comunitaria. Algunos ejemplos notables de estos movimientos son *comma.ai*, una mezcla de hardware y software que permite implementar un piloto automático en un auto; *Open source ecology* quienes están trabajando en 50 maquinas industriales fundamentales para la sociedad de forma de lograr una nueva economía súper eficiente basada en el ingenio y la cooperación; o *Arduino* que es un entorno de desarrollo para una serie de placas con

microcontroladores que son fácilmente programables desde el computador y que cuenta con una gran comunidad y variados proyectos basados en estas. Existen actualmente un sin fin de proyectos abiertos impulsados por comunidades en distintas áreas y con esta metodología de trabajo se han logrado crear tanto dispositivos como programas que son capaces de competir y hasta sobrepasar a las alternativas cerradas disponibles en el mercado.

Por otra parte, Chile tiene una economía que se basa principalmente en la extracción de materias primas, principalmente minerales y vegetales. Esto hace que la economía local se vea fuertemente influenciada por las fluctuaciones de los mercados internacionales y además, en el caso del cobre no es una actividad sostenible en el tiempo. Es necesario dar valor agregado a las materias primas del país, por una parte para optimizar las utilidades de estas y también para hacer mas robusta la economía de este. La última tendencia y el principal componente de la cuarta revolución industrial es la **Manufactura avanzada**, la cual se basa en dotar de inteligencia computacional a los procesos de fabricación, tiene entre sus principales herramientas el internet de las cosas y la automatización de procesos. En Chile las industrias manufactureras que han implementado manufactura avanzada logran 7,5 veces más facturación que las manufactureras tradicionales y una utilidad 23 veces mayor como se muestra en [2]. Es en esta linea que la corporación al fomento (CORFO) ha invertido en los últimos años en el *Programa de Manufactura Avanzada* (PEMA), el cual consiste en apoyo mediante capacitaciones y fondos concursables.

Así, con el mundo entrando a una cuarta revolución industrial y con un fuerte movimiento de desarrollo tecnológico colaborativo y comunitario, además con un país que necesita tanto modernizar y optimizar sus actividades productivas como combatir la gran desigualdad que sufre y de la que tristemente es líder a nivel mundial, se propone crear un brazo robótico versátil y abierto que permita bajar el umbral de acceso a la tecnología y acceder a la automatización de procesos a pequeños y medianos negocios. Entre los brazos robóticos destaca el *Selective Compliant Assembly Robotic Arm* (SCARA), el cual es ideal para mover cargas relativamente livianas a una gran velocidad con precisión. Cuenta con 4 grados de libertad, es ideal para trabajar en superficies con poca variación de altura y ocupa una superficie pequeña en comparación a su área de trabajo. Para que este brazo sea abierto y accesible a todas las personas, tanto sus diseños, esquemáticos, programas, diagramas de funcionamiento serán subidos una vez completados. Siguiendo la misma filosofía de desarrollar proyectos de forma comunitaria, este trabajo es una memoria multidisciplinaria entre ingeniería eléctrica y mecánica que cuenta con el apoyo y auspicio monetario de Beauchef Proyecta.

1.2. Objetivos

El objetivo general de esta memoria es diseñar un sistema de control para un robot multiherramienta OSH tipo SCARA e implementarlo en caso de ser factible. Los objetivos específicos son:

- Modelar el funcionamiento de un brazo robótico con cuatro grados de libertad.

- Determinar estrategias de control y elegir la mas pertinente en base a simulaciones.
- Generar lista de componentes que conformaran el sistema eléctrico y de control del robot.
- Desarrollar un programa que permita la operación interconectada y controlada de los componentes del robot.
- Implementar el sistema de control en un brazo robótico construido por un memorista de mecánica.
- Evaluar el desempeño del sistema desarrollado.

1.3. Alcances

Este trabajo es un proyecto multidisciplinario cuyo principal objetivo es la construcción de un brazo robótico en un trabajo conjunto entre ingeniería mecánica y eléctrica. Son parte de este trabajo todas las tareas necesarias para la construcción del sistema eléctrico y de control de dicho brazo robótico y además las simulaciones de distintas estrategias de control. Sin embargo, la implementación de estas últimas en el robot real dependerá de las limitaciones de hardware, en particular de la velocidad de comunicación entre los dispositivos.

Dado que la construcción de este robot depende de la compra de componentes y que gran parte de estos no se encuentran disponibles en el mercado nacional, es necesario importarlos, en particular algunos que son claves para el funcionamiento del robot. Por esta razón, los tiempos de construcción y por ende de generación de resultados dependerá fuertemente de los procesos de envío e internación. Además, la finalización de la construcción del brazo robótico no es imprescindible para la finalización de este trabajo.

Por último, todos los aspectos mecánicos que involucra el diseño y la construcción de un brazo robótico no son parte de este trabajo, esto es parte del memorista de ingeniería mecánica.

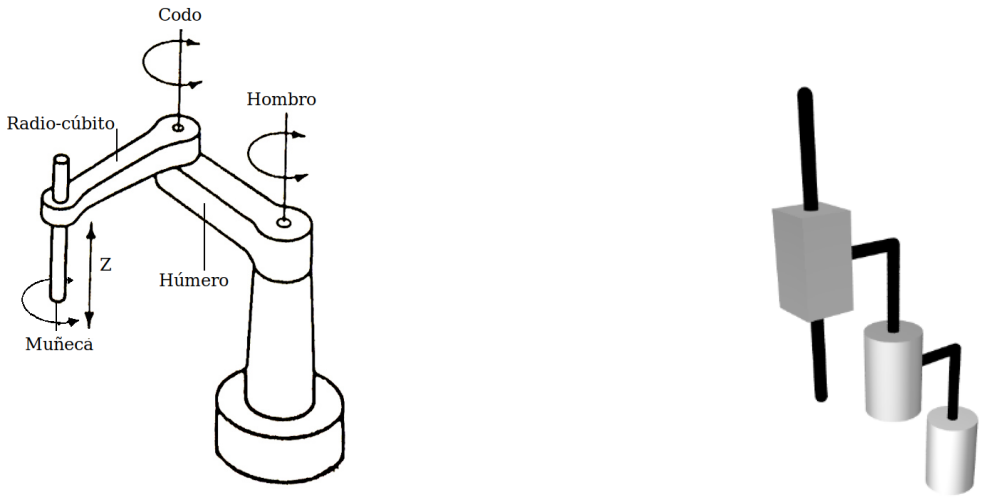
Capítulo 2

Marco teórico

En este capitulo se hace una breve descripción de los *Selective Compliant Assembly Robotic Arm* o *Selective Compliance Articulated Robot Arm* (SCARA), modelamiento y control. El modelamiento corresponde a las relaciones físicas y geométricas que describen el comportamiento del robot. Por su parte, el control, corresponde a los algoritmos y estrategias utilizadas por otros autores para controlar brazos mecánicos tipo SCARA.

2.1. Robot SCARA

Un *Selective Compliant Assembly Robotic Arm* (SCARA) es un tipo de robot el cual típicamente tiene 4 grados de libertad (hombro, codo, z y a). En la Figura 2.1a se muestra un diagrama donde las lineas con flechas al final representan los grados de libertad, además, se han incluido nombres de las articulaciones inspirados en la similitud con el brazo humano. Por otra parte, en la Figura 2.1b, se muestra el diagrama cinemático que nos indica que el hombro y el codo son grados de libertad rotacionales y Z es prismático. Por último, el cuarto grado de libertad no se incluye en el diagrama cinemático pero es también rotacional y se encuentra en la punta al final de la cadena cinemática, y se denomina A. En robótica el termino *compliant* esta relacionado al no impedimento del movimiento, a la suavidad y flexibilidad. Por esta razón el termino SCARA hace alusión a que es un brazo robótico en el que algunas de sus articulaciones no oponen resistencia al movimiento. En la práctica el termino se ocupa indistintamente si es que tiene esta característica o no, mas bien se refiere a la disposición de los eslabones.



(a) (b)

Figura 2.1: Diagramas de un SCARA. En (a) se muestra un dibujo con los nombres de las partes del robot y el tipo de movimiento. En (b) se muestra el diagrama cinemático. (a) es modificada de [3] y (b) es copiada de [4]

Este tipo de robots se caracteriza por su rapidez y precisión [5] y además por necesitar de un espacio de montura pequeño en comparación a su área de trabajo. Destacan a la hora de realizar tareas de *pick and place* (recoger y posicionar) o de ensamblaje [6]. Su área de trabajo es similar a un patrón cardioide con profundidad como se muestra en la Figura 2.2

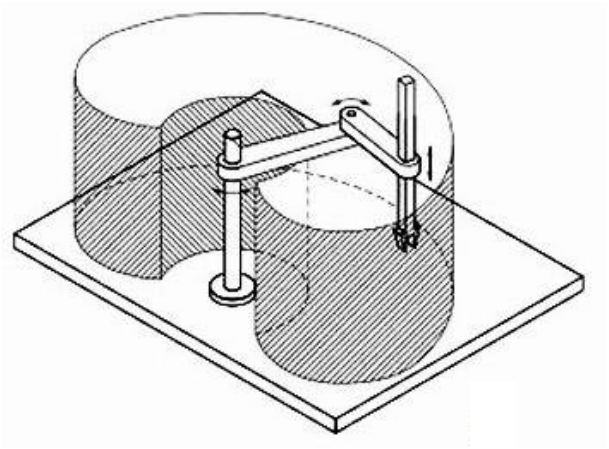


Figura 2.2: Envolvente de trabajo de un robot Scara. Copiada de [7]

El primer SCARA fue diseñado en 1981 en la universidad de Yamanashi y desde entonces han salido una gran cantidad de versiones comerciales y de investigación. En la Tabla 2.1 se muestran las especificaciones de algunos de los SCARA mas populares disponibles en la industria. En la primera columna aparece el FUM SCARA, un caso de estudio de uno de estos robots de bajo costo desarrollado por la Universidad de Ferdowsi en Mashad Iran

Tabla 2.1: Comparación entre las características de diferentes SCARA comerciales. XY corresponde al plano de trabajo que forman el movimiento del hombro y Codo, Z corresponde al eje Z, A corresponde al cuarto grado de libertad rotacional que se encuentra en la punta. Traducida y aumentada desde [8]

Especificación	Unidades	FUM SCARA	Epson LS6	Yamaha YK700XG	Adept Cobra s800	Fanuc SR-3iA
Alcance	mm	700	600	700	800	400
Velocidad XY	mm/s	8500	6800	8400	-	2466
Velocidad Z	mm/s	1100	1100	2300	1100	1800
Velocidad A	grados/s	1100	2000	1020	1200	3000
Repetibilidad XY	mm	±0.013	±0.02	±0.02	±0.017	±0.01
Repetibilidad Z	mm	±0.023	±0.01	±0.01	±0.03	
Repetibilidad eje A	grados	±0.016	±0.01	±0.05	±0.019	
Carga de trabajo	Kg	6	6	20	5.5	3

2.1.1. Modelo Cinemático

El modelo cinemático corresponde a las ecuaciones que relacionan el estado del robot con el punto que ocupa su efector en el espacio. Al analizar la forma en la que están construidos este tipo de robots en la Figura 2.1 se aprecia que la posición del efector final en el plano XY depende de los ángulos del hombro y el codo, sin embargo, la posición en el eje Z depende solamente de la altura de la muñeca, por esta razón es conveniente analizar el modelo cinemático en 2 partes, el plano XY (hombro-codo) y el eje Z.

Al mirar al robot desde arriba como en la Figura 2.3 se puede ver que una determinada configuración de los ángulo θ_1 y $\theta 2$ tiene una correspondencia a un punto (x, y) esto se conoce como la cinemática directa y esta relación se muestra en las ecuaciones (2.1) y (2.2). En cambio, dado un punto (x, y) se pueden conocer los ángulos θ_1 y $\theta 2$ que le corresponden, esto se conoce como la cinemática inversa y esta relación se muestra en las ecuaciones (2.3) y (2.4) y estas relaciones se obtienen en [9]. Cabe destacar que en las ecuaciones (2.3) y (2.4) existen 2 combinaciones de ángulos que satisfacen la igualdad, se debe elegir uno de estos pares según lo mas conveniente para la aplicación especifica que se esté llevando a cabo.

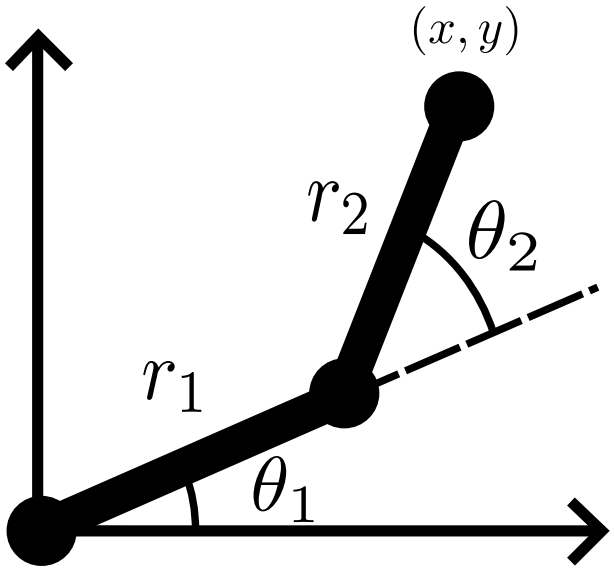


Figura 2.3: Definición de ángulos y su relación con un punto x,y

$$x = r_1 cos(\theta_1) + r_2 cos(\theta_1 + \theta_2) \tag{2.1}$$

$$y = r_1 sin(\theta_1) + r_2 sin(\theta_1 + \theta_2) \tag{2.2}$$

$$cos(\theta_2) = \frac{(x^2 + y^2) - (r_1^2 + r_2^2)}{2 r_1 r_2} \tag{2.3}$$

$$tan(\theta_1) = \frac{-(r_2 sin\theta_2)x + (r_1 + r_2 cos\theta_2)y}{(r_2 sin\theta_2)y + (r_1 + r_2 cos\theta_2)x} \tag{2.4}$$

Cabe mencionar que esta elección de sistemas de referencia para analizar la posición del robot en el plano XY es conveniente, desde el punto de vista de la aplicación, puesto que en un robot real y son estos ángulos los que son leídos por los sensores de posición angular. Mas adelante, al analizar el modelo dinámico del robot sería conveniente medir los ángulos en sistemas de referencia que sean paralelos entre si, ya que de esta forma el movimiento del hombro no generará un cambio en el ángulo del codo al mantener este paralelo. En la Figura 2.4 se muestra el nuevo sistema de coordenadas que es amigable con las ecuaciones que se verán mas adelante. De la Figura se extraen las ecuaciones (2.5) y (2.6), esto pues $\theta_1 = q_1$, la linea punteada es una extensión de la recta que forma este ángulo y los ejes x de los sistemas de coordenadas son paralelos ya que el ángulo se repite. Luego $q2$ es la suma del ángulo medido θ_2 con θ_1.

$$q_1 = \theta_1 \tag{2.5}$$

$$q_2 = \theta_2 + \theta_1 \tag{2.6}$$

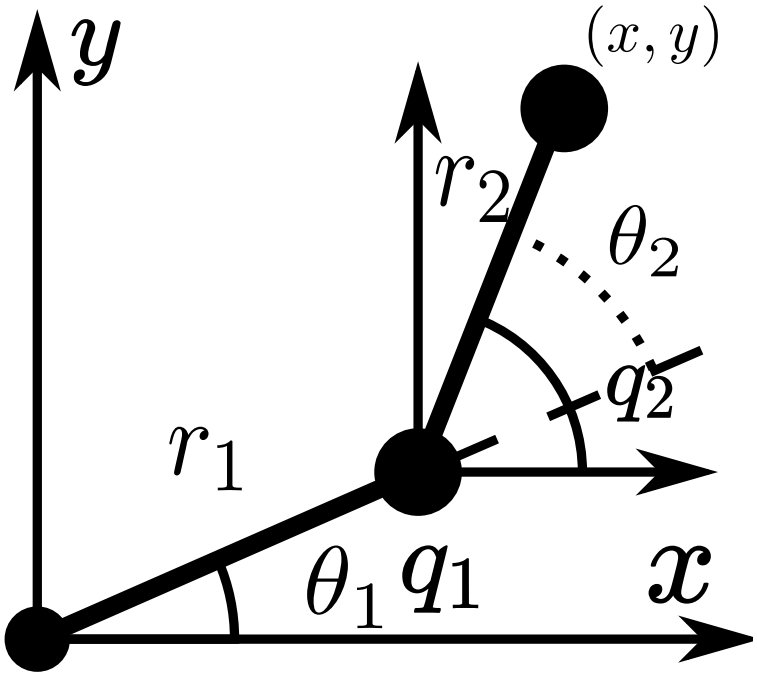


Figura 2.4: Sistema de coordenadas paralelo.

Además, al definir los ángulos de esta forma y calcular (x, y) en función de (q_1, q_2) (cinemática directa) se obtienen las ecuaciones (2.7), (2.8). Estas ecuaciones también se pueden extraer directamente de reemplazar (2.5) y (2.6) en (2.1) y (2.2).

$$x = r_1 cos(q_1) + r_2 cos(q_2) \tag{2.7}$$

$$y = r_1 sin(q_1) + r_2 sin(q_2) \tag{2.8}$$

Al considerar el efecto de una caja reductora con relación $1 : \eta$ con $\eta > 1$ desde un punto de vista cinemático y cuando la entrada de la reductora gira un ángulo α entonces la salida de la reductora gira α/η. Finalmente, al incluir este efecto en las articulaciones del codo y el hombro entonces, siendo α_1 y α_2 los ángulos de los respectivos motores, es claro que $\theta_1 = \alpha_1/\eta$ y $\theta_2 = \alpha_2/\eta$. Bajo este razonamiento y aplicándolo en las ecuaciones del sistema de referencia conveniente (2.5) y (2.6) se obtienen las ecuaciones (2.9) y (2.10).

$$q_1 = \alpha_1/\eta \tag{2.9}$$

$$q_2 = (\alpha_2 + \alpha_1)/\eta \tag{2.10}$$

Respecto a la altura en Z, se considera que este movimiento se consigue gracias a un husillo de bolas de paso p, lo que significa que cada 2π radianes que gira el tornillo, la tuerca avanza p. En la Figura 2.5 se

muestra un husillo de bolas y su funcionamiento básico. En la ecuación (2.11) se muestra la relación entre el ángulo del motor 3 α_3 y el avance en z.

Figura 2.5: Husillo de bolas que transforma movimiento rotacional en movimiento angular

$$z = \frac{p\alpha_3}{2\pi} \tag{2.11}$$

El cuarto grado de libertad corresponde a un giro angular de la punta del brazo robótico, por el momento asumiremos que esta relación es directa y sin reductora, por lo que el giro del motor es igual al ángulo de la punta $\alpha_4 = \theta_4$.

Se sigue el mismo procedimiento para utilizar sistemas de referencias convenientemente paralelos en estos dos últimos ángulos. Se suma al ángulo anterior del sistema de referencia no paralelo a la lectura del ángulo. Sin embargo, es claro que el movimiento de α_3 no tiene efecto en el sistema de referencia convenientemente paralelo para α_4 pero la posición del hombro si influye en el. Por esta razón, para obtener el nuevo sistema de referencia paralelo para α_3 y α_4 basta con sumar el ángulo del codo en el sistema de referencia paralelo, el cual se mostró anteriormente en la ecuación (2.10). Estas relaciones se resumen en las ecuaciones (2.12) y (2.13).

$$q_3 = \alpha_3 + q_2 = \alpha_3 + (\alpha_2 + \alpha_1)/\eta \tag{2.12}$$

$$q_3 = \alpha_4 + q_2 = \alpha_4 + (\alpha_2 + \alpha_1)/\eta \tag{2.13}$$

Finalmente, la relación entre el vector de ángulos de giro de los motores que componen el robot y el sistema de coordenadas convenientemente paralelo se resumen en la ecuación matricial

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \underbrace{\begin{bmatrix} 1/\eta & 0 & 0 & 0 \\ 1/\eta & 1/\eta & 0 & 0 \\ 1/\eta & 1/\eta & 1 & 0 \\ 1/\eta & 1/\eta & 0 & 1 \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix}}_{\alpha} \tag{2.14}$$

2.2. Modelo dinámico

En la presente sección se muestran 2 formas de obtener un modelo dinámico de un SCARA, por modelo fenomenológico y mediante simulaciones basadas en el diseño en CAD utilizando *Robotic Operative Sistem* (ROS).

2.2.1. Modelo dinámico fenomenológico

Las ecuaciones anteriores muestran de que forma se relacionan los ángulos con la posición sin tener en cuenta el como llevar a cabo este movimiento (cinemática), por lo que es necesario describir la física que gobierna el comportamiento de este sistema (dinámica). En [10], [11], [12], [13], [14] y [15] se utiliza un procedimiento similar para modelar el comportamiento dinámico del robot, el cual se basa en el Lagrangiano. El Lagrangiano es la suma de la energía potencial y cinética del sistema, se muestra en la ecuación (2.15) donde T es la energía cinética y V el potencial del sistema. Por su parte la energía cinética puede venir de dos fuentes, el momentum lineal y el momentum angular, según se muestra en la ecuación (2.16), donde v es la velocidad lineal del centro de masa m, $\dot{\theta}$ es la velocidad angular respecto al eje de rotación, e I es el momento de inercia en el eje de rotación.

$$L = T - V \tag{2.15}$$

$$T = \underbrace{(mv^2)/2}_{\text{momentum lineal}} + \underbrace{(I\dot{\theta}^2)/2}_{\text{momentum angular}} \tag{2.16}$$

Luego, se utiliza la ecuación de *Euler Lagrange* para obtener una EDO para cada una de las coordenadas del sistema. Esta ecuación se muestra en (2.17), donde q_i representa una coordenada del sistema para la cual se obtendrá la EDO, L es el *lagrangiano* de la ecuación (2.15), el punto en la parte superior $\dot{(\ )}$ denota la derivada temporal y Q_i representa el torque o fuerza no conservativa que realiza trabajo en el eje q_i. Este procedimiento y sus fundamentos son mostrados en [16].

$$\frac{\delta L}{\delta t}\frac{\delta L}{\delta \dot{q}_i} - \frac{\delta L}{\delta q_i} = Q_i \tag{2.17}$$

Al resolver la ecuación (2.17) respecto a cada coordenada y ordenar el resultado de una forma conveniente se puede generar la EDO matricial para la dinámica del sistema según la ecuación (2.18) [17], donde q representa el vector de coordenadas $q = [q_1...q_n]^t$, $\dot{q}$ y $\ddot{q}$ representan su primera y segunda derivada respecto al tiempo, $M(q)$ es la matriz de inercias, $C(q, \dot{q})$ es la matriz de las fuerzas de Coriolis, $N(q, \dot{q})$ son los términos asociados al potencial y a las energías no conservativas, por último $Q = [Q_1...Q_n]$ es el vector de fuerzas y torques que actúan en q_i.

$$M(q)\ddot{q} + C(q, \ddot{q})\dot{q} - N(q, \dot{q}) = Q \tag{2.18}$$

El enfoque propuesto es adecuado para modelar en general cualquier robot, independiente de que tipo de actuador tenga, ya sea rotacional o prismático. En el caso del SCARA, de las referencias mostradas al inicio de esta subsección ninguna puede ser utilizada directamente pues, o se consideran solo 2 grados de libertad y se ignora el roce [10], [12] y [15], o tienen más articulaciones [14]. El enfoque propuesto en [11] es el mas parecido a la configuración deseada, donde se consideran 3 grados de libertad, movimiento angular de hombro y movimiento prismático del eje z, además incluye roce. Sin embargo, por lo general el movimiento prismático se obtiene a partir de un tornillo de bolas y una tuerca, en el caso del SCARA esta rotación se realiza respecto a un eje paralelo a los del hombro y el codo, por lo que el momentum angular que produce el movimiento del eje Z influye en ambos y debe ser considerado. Entonces a continuación se presenta un nuevo modelo dinámico de SCARA, el cual incluye los 4 grados de libertad, el efecto del roce y también el uso de cajas reductoras.

En la Figura 2.6 se muestra el diagrama de SCARA a considerar. El símbolo (M) representa un motor, a su vez los engranajes que aparecen mas abajo de los motores ubicados más a la izquierda representan cajas reductoras. Mas abajo se ha incluido un glosario de los términos a considerar en este modelo.

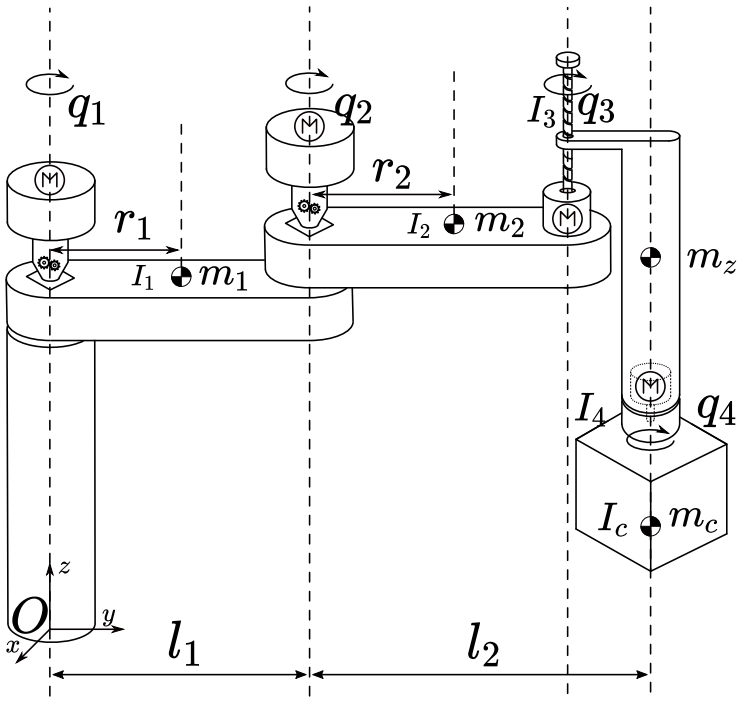


Figura 2.6: Diagrama dinámico de robot SCARA.

Luego, al aplicar la ecuación (2.17) en (2.28) se obtienen cuatro ecuaciones, las cuales se ordenan para expresarse de la forma mostrada en la ecuación (2.18). En (2.29) se muestra la matriz de inercias.

- q_1, q_2, q_3, q_4 Son los angulos de las partes del robot respecto a una linea paralela al eje X, como se mostró en la Figura 2.4. q_1 corresponde al hombro, q_2 corresponde al codo, q_3 es el angulo del motor asociado al eje z, q_4 es el ángulo del efector final.
- m_1, m_2: Masa total del humero y radio-cubito respectivamente. Incluye también la masa de las reductoras y de los motores.
- m_z: Masa que se mueve a lo largo de Z.
- m_c: Masa de la carga que transporta el robot.
- I_1, I_2, I_3, I_4. Son las inercias respecto al eje de giro. I_1 corresponde a la inercia del primer eslabón o humero respecto a su eje de giro, I_2 corresponde a la inercia del segundo eslavón o radio-cúbito respecto a su eje de giro, I_3 es la inercia del motor del eje z mas el tornillo y acople, I_4 es la inercia del motor 4 y la parte final del robot. I_c es la inercia de la carga.
- l_1, l_2: es el largo del primer y segundo eslabón respectivamente. l_2 incluye además el lago al centro de masa de m_z el cual se ha impuesto que a lo largo del eje z coincida con el centro de masa de la carga y el efector final
- r_1, r_2: es la distancia a los centros de masa del primer y segundo eslabón respectivamente

- τ_1, τ_2, τ_3, τ_4: Son los torques en el hombro, codo, husillo de z y efector final respectivamente.
- p: Es el paso del motor del husillo de bolas en z
- η: es la relación de reducción de las cajas reductoras del codo y hombro. η es mayor a uno y se asumirá igual para el codo y el hombro.

Para obtener el modelo dinámico se parte por analizar la energía cinética y potencial. Dado que la energía cinética tiene una componente asociada al momentum lineal es conveniente derivar las ecuaciones de posición de cada uno de los centros de masa. Además, las velocidades de cada eje (x, y, z) se suman al cuadrado y a dicha suma se le aplica raiz, una suma de distancias euclideanas. Las ecuaciones derivadas se muestran en (2.19) y (2.20) para el primer eslabón del robot y en (2.21) y (2.22). Además, en estas dos últimas ecuaciones, basta con reemplazar r_2 por l_2 y asi se obtienen las velocidades lineales de m_z y m_c.

$$\dot{x_1} = -r_1 sin(q_1)\dot{q_1} \tag{2.19}$$

$$\dot{y_1} = r_1 cos(q_1)\dot{q_1} \tag{2.20}$$

$$\dot{x_2} = -r_2 sin(q_2)\dot{q_2} - r_1 sin(q_1)\dot{q_1} \tag{2.21}$$

$$\dot{y_2} = r_2 cos(q_2)\dot{q_2} + r_1 cos(q_1)\dot{q_1} \tag{2.22}$$

Luego, se utiliza la ecuación (2.16) para formar el Lagrangiano. Por orden, se define K_1, K_2, K_3 y K_4 como la energía cinética asociada al movimiento de cada eslabón y se muestra en (2.23), (2.24), (2.25) y (2.26) respectivamente. Además, en la ecuación (2.27) se muestra la energía potencial de este sistema, la cual corresponde a la gravedad que afecta el eje z y la carga. El lagrangiano corresponde a la suma de todas estas expresiones y se muestra en (2.28)

$$K_1 = \frac{I_1 \dot{q}_1^2}{2} + \frac{m_1 r_1^2 \dot{q}_1^2}{2} \tag{2.23}$$

$$K_2 = \frac{I_2 \dot{q}_2^2}{2} + \frac{m_2 \left(\left(-L_1 \sin\left(q_1\right) \dot{q}_1 - r_2 \sin\left(q_2\right) \dot{q}_2 \right)^2 + \left(L_1 \cos\left(q_1\right) \dot{q}_1 + r_2 \cos\left(q_2\right) \dot{q}_2 \right)^2 \right)}{2} \tag{2.24}$$

$$K_3 = \frac{I_3 \dot{q}_3^2}{2} + \frac{m_z p^2 \dot{q}_3^2}{8\pi^2} + \frac{m_z \left(\left(-L_1 \sin\left(q_1\right) \dot{q}_1 - L_2 \sin\left(q_2\right) \dot{q}_2 \right)^2 + \left(L_1 \cos\left(q_1\right) \dot{q}_1 + L_2 \cos\left(q_2\right) \dot{q}_2 \right)^2 \right)}{2} \tag{2.25}$$

$$K_4 = \frac{I_4\dot{q}_4^2}{2} + \frac{m_c p^2 \dot{q}_3^2}{8\pi^2} + \frac{m_c\left(\left(-L_1\sin\left(q_1\right)\dot{q}_1 - L_2\sin\left(q_2\right)\dot{q}_2\right)^2 + \left(L_1\cos\left(q_1\right)\dot{q}_1 + L_2\cos\left(q_2\right)\dot{q}_2\right)^2\right)}{2} \tag{2.26}$$

$$V = \frac{gp\left(-m_c - m_z\right)q_3}{2\pi} \tag{2.27}$$

$$\begin{aligned} L = &\frac{I_1\dot{q}_1^2}{2} + \frac{I_2\dot{q}_2^2}{2} + \frac{I_3\dot{q}_3^2}{2} + \frac{I_4\dot{q}_4^2}{2} + \frac{L_1^2 m_2\dot{q}_1^2}{2} + \frac{L_1^2 m_c\dot{q}_1^2}{2} + \frac{L_1^2 m_z\dot{q}_1^2}{2} + L_1L_2m_c\cos\left(q_1 - q_2\right)\dot{q}_1\dot{q}_2 \\ &+ L_1L_2m_z\cos\left(q_1 - q_2\right)\dot{q}_1\dot{q}_2 + L_1m_2r_2\cos\left(q_1 - q_2\right)\dot{q}_1\dot{q}_2 + \frac{L_2^2 m_c\dot{q}_2^2}{2} + \frac{L_2^2 m_z\dot{q}_2^2}{2} \\ &- \frac{gm_c pq_3}{2\pi} - \frac{gm_z pq_3}{2\pi} + \frac{m_1 r_1^2\dot{q}_1^2}{2} + \frac{m_2 r_2^2\dot{q}_2^2}{2} + \frac{m_c p^2\dot{q}_3^2}{8\pi^2} + \frac{m_z p^2\dot{q}_3^2}{8\pi^2} \end{aligned} \tag{2.28}$$

Luego, al aplicar la ecuación (2.17) en (2.28) se obtienen cuatro ecuaciones, las cuales se ordenan para expresarse de la forma mostrada en la ecuación (2.18). En (2.29) se muestra la matriz de inercias, en (2.30) la matriz de las fuerzas coriolis y en (2.31) el vector de potencial.

$$M = \begin{bmatrix} I_1 + L_1^2m_2 + L_1^2m_c + L_1^2m_z + m_1r_1^2 & L_1\left(L_2m_c + L_2m_z + m_2r_2\right)\cos\left(q_1 - q_2\right) & 0 & 0 \\ L_1\left(L_2m_c + L_2m_z + m_2r_2\right)\cos\left(q_1 - q_2\right) & I_2 + L_2^2m_c + L_2^2m_z + m_2r_2^2 & 0 & 0 \\ 0 & 0 & \frac{\pi^2 I_3 + \frac{p^2\left(m_c + m_z\right)}{4}}{\pi^2} & 0 \\ 0 & 0 & 0 & I_4 \end{bmatrix} \tag{2.29}$$

$$C = \begin{bmatrix} 0 & -L_1\left(L_2m_c + L_2m_z + m_2r_2\right)\sin\left(qu_1 - qu_2\right)\dot{qu}_2 & 0 & 0 \\ L_1\left(L_2m_c + L_2m_z + m_2r_2\right)\sin\left(qu_1 - qu_2\right)\dot{qu}_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.30}$$

$$g = \begin{bmatrix} 0 \\ 0 \\ \frac{gp\left(m_c + m_z\right)}{2\pi} \\ 0 \end{bmatrix} \tag{2.31}$$

Para completar este modelo es necesario incluir el roce. En [11] se incluye el roce directamente en la ecuación dinámica según se muestra en la ecuación (2.32). En (2.33) se muestra el torque producido por el roce, F_c y F_v son los parámetros asociados a su componente culombica y viscosa respectivamente. Se consideran los mismos parámetros para cada una de las articulaciones, aun cuando los actuadores funcionan de forma distinta. Este enfoque está basado en [18], donde se presenta un procedimiento para estimar los coeficientes F_c y F_v en conjunto con la matriz de inercia. Cabe mencionar que el roce considerado en la ecuación (2.33) solo considera el efecto del roce a altas velocidades, esto es correcto para este robot.

$$M(q(t))\ddot{q(t)} + C(q(t), \dot{q(t)})\dot{q(t)} + \tau_{friction} + g(q(t)) = \tau(t) \tag{2.32}$$

$$\tau_{fric} = F_v\dot{q} + F_c sign(\dot{q}) \tag{2.33}$$

Cabe recordar que este modelo está definido con sistemas de referencias con ejes paralelos según se mostró en la sección anterior. Obviamente, este modelo no se condice con la realidad, pues el ángulo se mide en los motores (antes de las cajas reductoras) y además el origen depende de los eslabones anteriores, ambos sistemas de referencia están relacionados según en la ecuación (2.14). Cada sistema de referencia tiene un modelo asociado y existe una relación directa entre ambos modelos según se investiga en [19]. Sea q el sistema de referencia definido con ejes paralelos y α el sistema de referencia definido según la posición de los motores de tal forma que ambos se relacionan según la ecuación (2.14). Al resolver las ecuaciones de *Euler Lagrange* y encontrar 2 sistemas ordenados como en la ecuación (2.17) con matrices $M(q)$, $C(q, \dot{q})$, $g(q)$ y $\bar{M}(\alpha)$, $\bar{C}(\alpha, \dot{\alpha})$ $\bar{g}(\alpha)$, respectivamente, se puede obtener una relación entre ambos sistemas según se muestra en las ecuaciones (2.34), (2.35), (2.36). Además, la relación entre los vectores de torque de ambos sistemas se muestra en (2.37).

$$\bar{M}(\alpha) = A^t \times M(\underbrace{A\alpha}_{q}) \times A \tag{2.34}$$

$$\bar{C}(\alpha, \dot{\alpha}) = A^t \times C(\underbrace{A\alpha, A\dot{\alpha}}_{q,\dot{q}})A \tag{2.35}$$

$$\bar{g}(\alpha) = A^t g(\underbrace{A\alpha}_{q}) \tag{2.36}$$

$$\bar{\tau} = A^t \times \tau \tag{2.37}$$

Finalmente, en la ecuación (2.38) se muestra la forma final del modelo del SCARA. Nótese que el sistema obtenido utiliza el torque definido según el sistema de referencia de los motores y las coordenadas del sistema definido respecto a la paralela, para una posterior simulación se debe incluir una transformación lineal del sistema de referencia, desde el que es conveniente para simular, al que es factible medir.

$$M(q(t))\ddot{q(t)} + C(q(t), \dot{q(t)})\dot{q(t)} + \tau_{friction} + g(q(t)) = \tau(t) = A^T\tau \tag{2.38}$$

Luego para poder hacer simulaciones con este modelo es necesario escribirlo de la forma $\dot{x} = f(x)x + u$. Primero se define $x = [\dot{\alpha_1}\dot{\alpha_2}\dot{\alpha_3}\dot{\alpha_4}]^T$ y se calcula x. En el anexo B se muestra la implementación de este

modelo en Matlab, los resultados obtenidos hacen pensar que este modelo tiene algún error de arrastre por lo que no se va a utilizar. Se ha dejado acá para que en un trabajo futuro sea corregido e implementado.

$$\dot{x} = M^{-1}A^{T}\tau - M^{-1} * (C * x + g) = f(x) + u \tag{2.39}$$

2.2.2. Robotic Operative System

Robotic Operative System (ROS), es una alternativa *Open-Source* para operar todo tipo de robots. Es lo que se llama un *Middleware* una plataforma que conecta la información de cada parte del robot con el software que usa esa información. Si bien en el presente trabajo solo se usa para generar simulaciones, tiene mucha mas potencialidad que eso, en cuanto es una plataforma que permite el desarrollo y la interaccion de robots de todo tipo y ante diferentes situaciones.

La plataforma ROS se basa en los siguientes elementos:

- `Node`: un nodo es un elemento que procesa información, puede ser, el controlador de una rueda, la lectura de un sensor, un mapa del ambiente del robot, etc. Ros basa su topología en los nodos, cualquier proceso del robot o sistema de robots es un nodo.
- `Message`: es una estructura de datos utilizada para comunicar información entre nodos.
- `Topic`: un tópico es la forma en la que se organiza la trayectoria de la información. Un nodo que tenga información que puede servirle a otro la publica en un tópico. Mientras que un nodo que utiliza la información que se publica en un tópico se suscribe a esta.
- `Service`: Un servicio es la estructura que tiene ROS para pedir información a un determinado nodo. Un nodo pone a disposición un servicio, otro nodo solicita la información y espera la respuesta.

En la actualidad ROS es el estándar en desarrollos de robótica, es una plataforma ampliamente utilizada y que es compatible con otras alternativas *Open-source*. Cuenta con una gran comunidad que mantiene actualizada la plataforma. Además, tiene una extensa *Wiki*, un sistema de trabajo informático utilizado en los sitios web que permite a los usuarios modificar o crear su contenido de forma rápida y sencilla, en la que se encuentra toda la información del código, como utilizarlo y tutoriales para todos los niveles de experticia.

Cabe mencionar que la utilización de esta plataforma tiene una dura curva de aprendizaje, una simple aplicación implica la escritura de muchos archivos, de distintas extensiones en los que un simple error de escritura hace que todo falle. Por esta razón, a la hora de hacer un robot desde 0, nace la pregunta de cuales son las ventajas que tiene esta plataforma contra realizar un código propio. Lo último puede parecer como una buena alternativa en cuanto a tiempo, sin embargo, al querer hacer una aplicación mas compleja,

por ejemplo coordinar varios robots a la vez o implementar visión computacional, el trabajo de hacer todo este código desde 0 es mas extenso que hacerlo en ROS. En otras palabras, si bien es mas difícil hacer implementaciones en ros al principio, permite realizar aplicaciones mas complejas a la larga.

Para realizar simulaciones en ROS se utiliza Gazeboel que permite simular el comportamiento de robots en la realización de tareas varias. Es muy versátil en cuanto permite simular distintos tipos de robots en distintos escenarios, tanto en interior como exterior o en condiciones especificas (submarinas, viento, 0 gravedad entre otras). Funciona por si solo mediante una GUI y también mediante comandos en la consola, además se puede utilizar en conjunto con ROS. Para poder cargar un modelo de robot en Gazebo es necesario tener un URDF o un SDF.

URDF significa (*Unifiedl Robot Description File*) es un archivo en formato `xml` cuya extensión es URDF y como su nombre lo indica, es una descripción del robot. Para el URDF un robot es un conjunto de eslabones (*link*) rígidos que se juntan en articulaciones (*joint*) como se muestra en la Figura 2.7. Este formato es adecuado para describir robots rígidos que puedan ser representados en forma de árbol [20], esto deja afuera a los robots paralelos, en los cuales dos o mas cadenas de eslabones se conectan en 2 partes, curiosamente existe una topología de SCARA similar la cual se basa en este principio, otro ejemplo son los robots tipo delta.

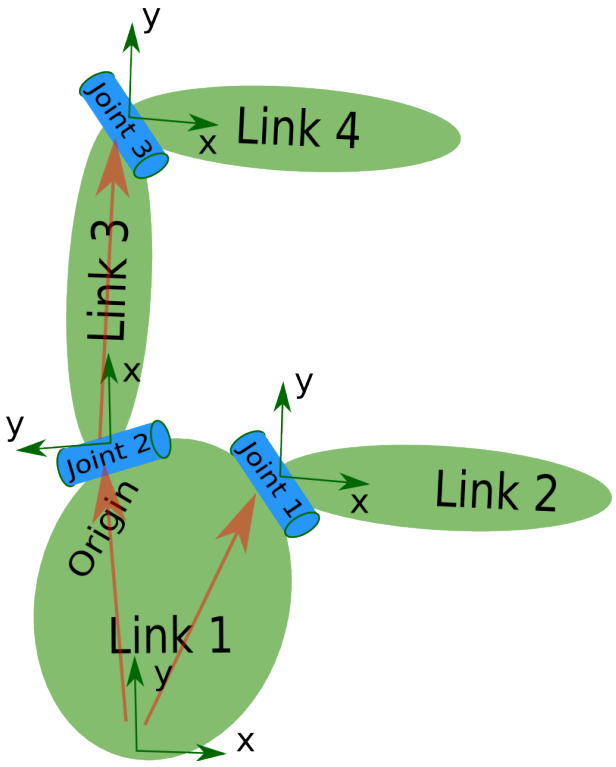


Figura 2.7: Diagrama de descripción de un robot mediante URDF. Imagen copiada de [20]

Para escribir un URDF se debe empezar por un robot, luego definir los eslabones (*links*), que se conectan mediante las articulaciones (*joint*), como se muestra en el URDF básico mostrado a continuación.

```
1 <robot name="pr2">
```

```
  <link> ... </link>
  <link> ... </link>
  <link> ... </link>

  <joint> .... </joint>
  <joint> .... </joint>
  <joint> .... </joint>
</robot>
```

Dentro de cada elemento eslabón (*link*) se describen 3 tags o propiedades, estos son <visual¿, <collition> e <inertial¿. El tag <visual> describe como se ve el robot, esto se puede hacer mediante figuras predeterminadas o cargando un archivo de modelo tridimensional, el cual puede ser en formato .STL o .DAE. El tag <collition> describe los limites físicos del eslabón, donde choca, si bien esta característica y visual guardan estrecha relación, la primera es lo que se ve y la segunda es donde se choca, tenerlas separadas permite cargar un archivo tridimensional de menor resolución en <collition> y así disminuir la carga computacional de la simulación. Por último, <inertial> describe la inercia del eslabón, para esto es necesario definir el origen del centro de masa y el tensor de inercia, el cual es una matriz simétrica respecto a su diagonal, se describe mediante 6 parametros.

Dentro del elemento articulación (*link*) se definen los eslabones conectados mediante los tags <parent> y <child>. El tag <parent> el eslabón desde cual se conecta, y <child> el eslabón conectado. También se define el tipo de articulación, esta puede ser *prismatic* (movimiento lineal), *revolute* (movimiento rotacional con limites), *continuos* (movimiento rotacional sin limites), *fixed* (fijo), *floating* (movimiento totalmente libre) y *planar* (movimiento en un plano). Además es necesario definir un eje del movimiento y parámetros de ubicación y rotación del eslabón <child>, estas relaciones se definen según las distancias entre los orígenes de ambos eslabones.

Para este trabajo se ha creado un URDF indirectamente desde el programa en el cual se encuentra el diseño mecánico, este procedimiento se describe con detalle en el anexo A

2.3. Control

El problema del control de un brazo robótico presenta otras aristas que no existen a la hora de diseñar un controlador para una planta típica. Por lo general, al diseñar y afinar un controlador se busca que tenga una buena respuesta ante una entrada tipo escalón, haciendo que el tiempo de establecimiento y el sobrepaso sean lo menores posibles y cumplan con restricciones impuestas previamente. Esta estrategia es adecuada cuando la planta a controlar tiene pocos cambios en la referencia, por ejemplo si se quisiera mantener el flujo de agua constante durante un proceso químico o controlar la temperatura en un horno de pan, este no es el caso de un brazo robótico. Durante su operación es necesario mover el brazo de un punto a otro, donde el punto de inicio y el punto de final son arbitrarios según la tarea que se este

desarrollando. Si se hiciera esto solamente cambiando la referencia en forma de escalón pasaría que si la diferencia entre ambas referencias es mayor que para la cual su controlador fue diseñado, el sobrepaso sería mas grande. Si por el contrario se diseñara el controlador para el cambio de referencia mas extremo, cuidando tener un sobrepaso igual a 0 para no sobrepasar los limites físicos del robot, entonces el tiempo de establecimiento sería excesivo para el resto de movimientos posibles, el brazo se tendría que mover mas lento que lo que puede. Para hacer frente a esta problemática se pueden tomar 2 estrategias, la primera es desarrollar un controlador que se adapte a las distintas referencias y estados del robot, para esto existe un vasto abanico de posibilidades con diferentes cualidades pero con la desventaja de que la aplicación en la realidad no es directa y depende del hardware que se seleccione. La segunda estrategia que se puede tomar, es mas sencilla de aplicar y no depende del tipo de controlador a utilizar ni tampoco es conflictiva con el desarrollo de un controlador especializado, esta estrategia corresponde a elegir una trayectoria optimizada. Esta subsección tiene como primer objetivo mostrar controladores tipo PID y procedimientos de afinación de estos; y como segundo objetivo, estudiar el problema de la generación de trayectorias óptimas.

Un controlador es un dispositivo que se encarga de mantener una variable observada de una planta o proceso $y(t)$ cercana a una referencia $u(t)$. El controlador mas utilizado e la industria se llama PID, y utiliza el error entre estas dos variables para calcular la entrada para la planta que hace converger este error a 0. Su funcionamiento se basa e calcular el error, su integral y su derivada, multiplicar estas 3 partes por su respectiva ganancia y sumarlas, esta es la salida del controlador y la entrada de la planta, como se muestra en la Figura 2.8. Este tipo de controlador ha sido ampliamente usado en la industria en procesos de todo tipo y sus primeras aplicaciones datan desde 1911 [21].

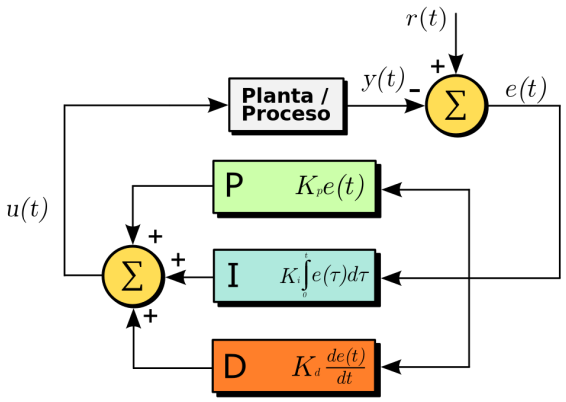


Figura 2.8: Diagrama de controlador PID. Dibujo de TravTigerEE, wikicommons.

La implementación de este tipo de controlador ha sido amplia en los robots SCARA, en este trabajo analizaremos 3 aplicaciones con diferentes enfoques pero mismo controlador. Primero, en [10] se utiliza el un modelo fenomenológico basado en las ecuaciones de *Euler-Lagrange*, además se incluye el comportamiento de los motores DC y su relación de torque vs corriente en el modelo. En esta ocasión el control se realiza mediante un controlador proporcional derivativo PD para el hombro y el codo. No se da detalle sobre como se obtienen las ganancias de los controladores. Tampoco se concluye con alguna medida sobre el

sobrepaso o tiempo de establecimiento, sin embargo según los gráficos de sus resultados su modelo se ajusta con buena precisión a los resultados experimentales, los cuales se muestran en la Figura 2.9 . El aporte de este trabajo es la obtención del modelo fenomenológico y posterior contraste de este con resultados experimentales.

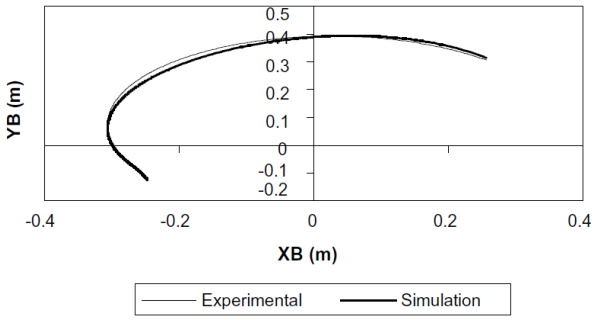


Figura 2.9: Trayectoria seguida por brazo SCARA modelo *Serpent-1* utilizando strategia de control PD, la duración de la prueba fue 3 segundos. Figura copiada desde [10].

En [22] se ocupa una estrategia similar, en esta ocasión el tipo de controlador es PID, también se sintoniza por separado para el hombro y el codo y el método de sintonización es heurístico en base a simulaciones. Para realizar estas se utilizan herramientas de *MATLAB* en base a un diseño creado en *SolidWorks*. Además en este trabajo se incluye una trayectoria especifica, la cual corresponde a un polinomio e grado 3 que conecta los puntos inicial y final. En la Figura 2.10 se muestran los resultados de una simulación con el controlador encontrado y con una trayectoria optimizada.

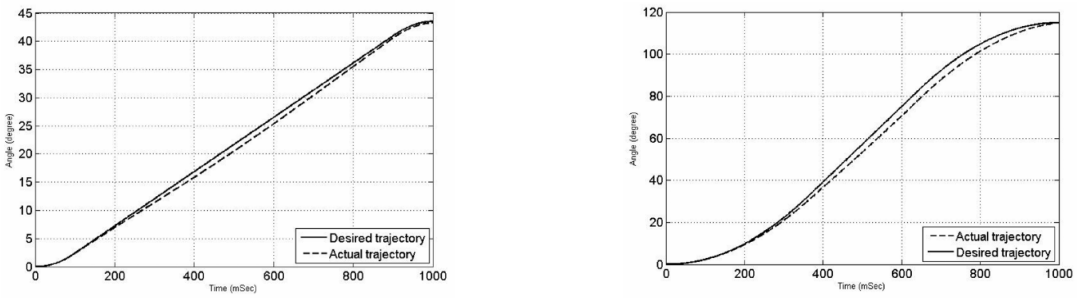


Figura 2.10: Simulación de brazo SCARA con modelo basado en CAD utilizando estrategia de control PID para una trayectoria cúbica, la duración de la prueba fue de 1 segundo.(a) y (b) corresponden al ángulo respecto al tiempo para el hombro y el codo respectivamente. Figura copiada desde [22].

En [23] nuevamente se genera un modelo dinámico en base a las ecuaciones de *Eueler-Lagrange* y posteriormente es controlado en simulaciones con un controlador PD. Tampoco se incluye un procedimiento de afinación de este controlador. Posteriormente se utiliza este controlador para entrenar una red neuronal de 3 capas que recibe el error entre el set point y el estado del sistema (para cada uno de los grados de libertad). La red neuronal se entrena utilizando backpropagation. Se prueban distintas funciones de transferencias y cantidad de neuronas en la capa oculta hasta encontrar una configuración que minimice el MSE. La red neuronal presenta mejores resultados

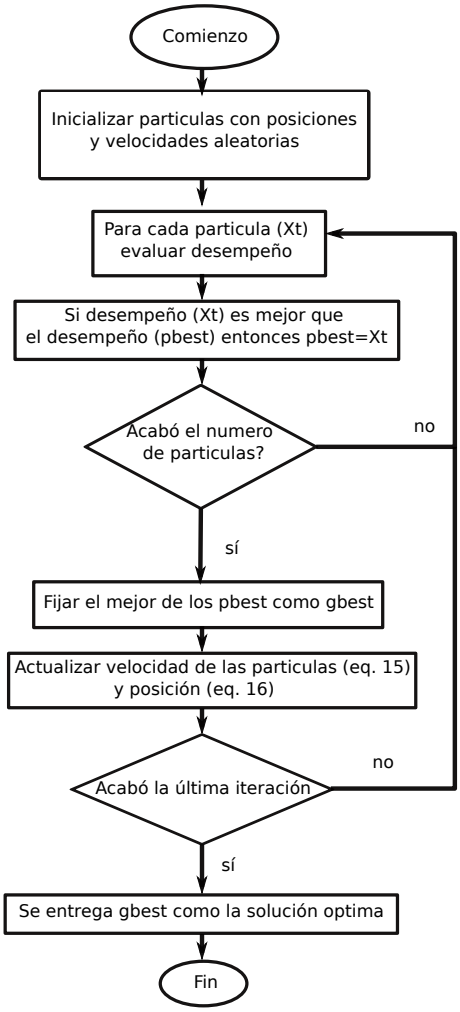


Figura 2.11: Algoritmo PSO. Traducida desde [24], las ecuaciones a las que se hace referencia se encuentran en dicho trabajo.

De estas 3 aplicaciones en diferentes SCARA se puede concluir que la estrategia de utilizar un controlador PID es adecuada. Sin embargo no se ha incluido un proceso de entrenamiento o afinación de estos controladores, la elección de las ganancias se ha hecho de forma heurística. En [24] se propone una metodología para la afinación del controlador PID. Se utiliza como ejemplo un robot sencillo de 2 grados de libertad y se determina su modelo fenomenológico en base a las ecuaciones de *Euler-Lagrange*, luego se le agrega un controlador PID a ambas articulaciones. Con esta estructura se utiliza *Particle Swarm Optimization* para determinar las ganancias que minimizan el error cuadrático medio para una entrada tipo escalón. PSO es un algoritmo de optimización iterativo que se basa en una "población de partículas"las que corresponden a un set de parámetros candidatos, los que se evalúan en la función a minimizar. En la siguiente iteración cada partícula mueve sus parámetros acercándose a la partícula que tuvo el mejor desempeño y también considerando las decisiones que ha tomado en el pasado y han mejorado el propio. Se puede encontrar mas detalle de este algoritmo en [25]. En la Figura 2.11 se muestra un diagrama del algoritmo utilizado en el trabajo citado.

Hasta este punto se cuenta con un controlador que funciona en este tipo de robot y una estrategia para afinarlo. Para mejorar mas aun el desempeño del sistema a implementar se propone seguir la segunda estrategia mencionada anteriormente, definir una trayectoria óptima. Utilizar una trayectoria con sobraceleración limitada es un método ampliamente utilizado para suavizar el comportamiento dinámico y reducir vibraciones y desgaste en sistemas robóticos [26]. La sobreaceleración o *jerk* en inglés corresponde a la derivada o tasa de cambio de la aceleración. Existen distintos algoritmos para generar este tipo de trayectorias, en [27] se hace una revisión del estado del arte en este problema a la fecha y se propone una nueva metodología para generarlas en base a la función sigmoide, la cual se muestra en la ecuación (2.40). Esta función es una especie de escalón de 0 a 1 pero con una subida paulatina e infinitamente derivable en todos sus puntos, estas son características deseables al hacer trayectorias pues aseguran transiciones suaves a lo largo de la trayectoria. Esta metodología para generar trayectorias es independiente de la configuración cinemática del robot y puede aplicarse para cada articulación por separado o en un movimiento en conjunto, en el plano XY por ejemplo, se incluye en el trabajo 2 estrategias para coordinar 2 o mas grados de libertad moviéndose simultáneamente.

$$\frac{1}{1+e^{-x}} \tag{2.40}$$

El procedimiento para generar una trayectoria se basa en armar un perfil de sobreaceleración, este puede tener 3 valores, 0 y la sobreaceleración máxima en valor positivo y negativo. El perfil es armado de forma que la transición entre dichos valores sea de forma suave al estar conectado mediante funciones sigmoides. Sean V_{max}, A_{max} y J_{max}, la velocidad, aceleración y sobreaceleración máxima respectivamente y D la distancia objetivo a recorrer, la diferencia entre el punto inicial y final, la cual puede ser positiva o negativa. Además imponiendo que todas las derivadas de la distancia sean 0 en las posiciones inicial y final. Para recorrer D en el menor tiempo posible, el perfil generado busca alcanzar lo mas pronto posible los limites del robot J_{max}, A_{max} y V_{max} sucesivamente y mantener dichas condiciones de saturación el mayor tiempo posible, obviamente sin sobrepasar D ni tampoco infringir las condiciones de derivadas nulas al inicio y al final. La ecuación (2.41) muestra el perfil de sobreaceleración que tiene dicho comportamiento donde $t_i(i = 0, 1, ..., 15)$ corresponden a los limites de los 15 segmentos que componen el perfil, $\tau_i = \frac{t-t_{i-1}}{t_i-t_{i-1}}$ corresponde al tiempo normalizado en el intervalo $[t_{i-1}, t_i]$. Posteriormente se integra este perfil para obtener la aceleración, la velocidad y la distancia sucesivamente, en la Figura 2.12 se muestra un ejemplo de dicho proceso, se aprecia que todas las curvas curven con las restricciones de valores máximos. Respecto al parámetro a este afecta fuertemente el comportamiento de la función sigmoide, en este trabajo se incluye un estudio completo sobre su valor y efectos en la generación de trayectorias, cuya conclusión es su valor óptimo es igual a $a = \sqrt{3}/2$ y es el que se usa en toda la publicación. Además en el estudio de este parámetro aparece una nueva restricción S_{max}, el cual es el m áximo valor de la derivada de la sobreaceleración o *Snap*. Este parámetro tiene un efecto directo en el tiempo que la trayectoria tarda en llegar a su máximo valor y por ende la duración de la trayectoria, por esta razón es utilizado como parámetro de diseño para seleccionar el comportamiento deseado en la trayectoria, un valor de S_{max} alto

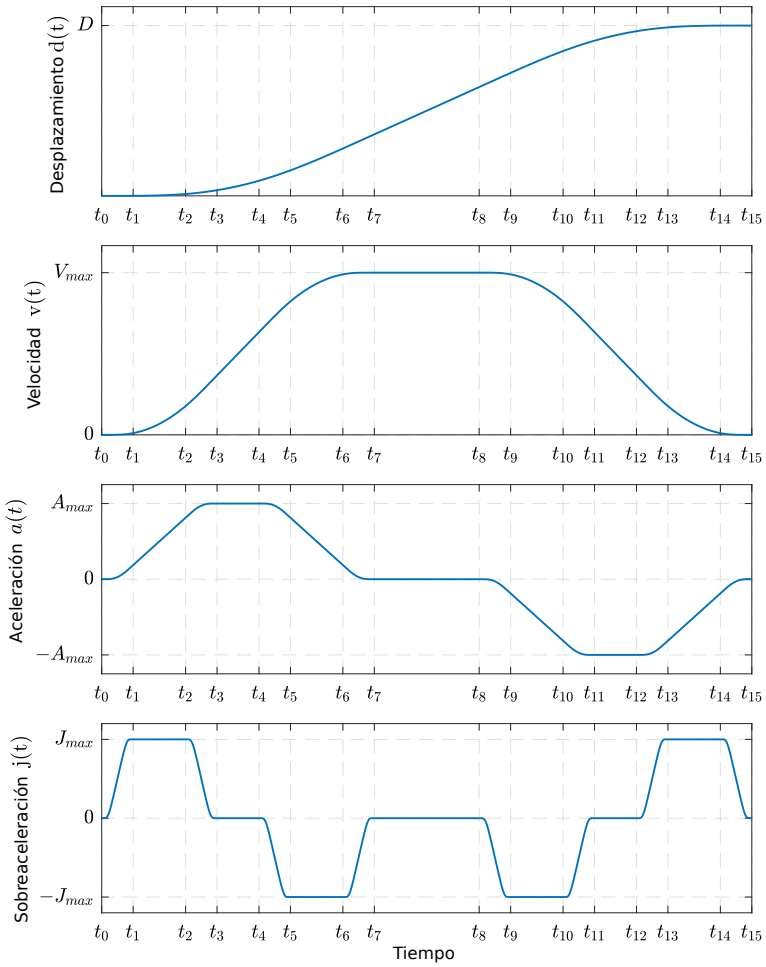


Figura 2.12: Ejemplo de generación de trayectoria optima basado en sobreaceleración limitada y transiciones suaves echas con función sigmoide. Traducida desde [27].

genera una trayectoria corta y rapida, mientras un valor bajo genera una trayectoria mas suave.

$$j(t) = sign(D) \begin{cases} J_{max} \frac{1}{1+e^{-a(1/(1-\tau_i)-1/\tau_i)}} & t_0 \leq t \leq t_1, t_{12} \leq t \leq t_{13} \\ J_{max} & t_1 \leq t \leq t_2, t_{13} \leq t \leq t_{14} \\ J_{max} \frac{1}{1+e^{a(1/(1-\tau_i)-1/\tau_i)}} & t_2 \leq t \leq t_3, t_{14} \leq t \leq t_{15} \\ 0 & t_3 \leq t \leq t_4, t_7 \leq t \leq t_8, t_{11} \leq t \leq t_{12} \\ -J_{max} \frac{1}{1+e^{-a(1/(1-\tau_i)-1/\tau_i)}} & t_4 \leq t \leq t_5, t_8 \leq t \leq t_9 \\ -J_{max} & t_5 \leq t \leq t_6, t_9 \leq t \leq t_{10} \\ -J_{max} \frac{1}{1+e^{a(1/(1-\tau_i)-1/\tau_i)}} & t_6 \leq t \leq t_7, t_{10} \leq t \leq t_{11} \end{cases} \tag{2.41}$$

Al analizar la Figura 2.12 se ve que hay tramos de sobreaceleración variable T_s ($[t_0, t_1], [t_2, t_3], [t_4, t_5], [t_6, t_7], [t_8, t_9], [t_{10}, t_{11}], [t_{12}, t_{13}]$; constante T_j ($[t_1, t_2], [t_5, t_6], [t_9, t_10]y[t_{13}, t_{14}]$), aceleración constante T_a ($[t_3, t_4]y[t_{11}, t_{12}]$) y velocidad constante T_v ($[t_7, t_8]$). La duración total de la trayectoria es $T = 8T_s + 4T_j + 2T_a + T_v$. Puede pasar que por ejemplo el actuador a utilizar cuente con una reducción de velocidad muy grande, lo que haría que tuviera gran aceleración pero muy poca velocidad máxima, o que la distancia a recorrer sea muy corta y no sea

posible alcanzar la velocidad máxima. En estos casos sucede que no se satura una o mas de las restricciones J_{max}, A_{max} y V_{max}, según el caso puede ser que una o mas secciones del perfil completo mostrado en la Figura 2.12 no se incluya, además para que la distancia final sea la deseada en estos casos se debe limitar el valor J_{max} a un valor mas bajo. Para hacer frente a estos múltiples casos y para mantener el tiempo de computo bajo, se define un árbol de decisión para generar el perfil de sobreaceleración, el cual se muestra en la Figura 2.13. Este procedimiento se basa en evaluar ordenadamente la inclusión de las partes del perfil de trayectoria completa e incluir un segmento en caso de que el tiempo total que demora la trayectoria sea el menor, esto se hace con expresiones cerradas para la duración de las partes de la trayectoria, no hay etapas de optimización. Dado que existen 4 partes de la curva T_s, T_j, T_a y T_v y que de estas la única que siempre esta presente es la de sobreaceleración variable, se generan 8 combinaciones posibles de perfiles de sobreaceleración, los cuales se describen a continuación, ademas en la Figura 2.13 en los ejemplos de las curvas se muestran sombradas en celestes las zonas incluidas.

- tipo I : Distancia muy corta, solamente se incluye T_s
- tipo II : Se alcanza la velocidad máxima, se incluyen T_s y T_v
- tipo III : Se alcanza la aceleración máxima, se incluyen T_s y T_a
- tipo IV : Se alcanza la velocidad máxima y la aceleración máxima, se incluyen T_s y T_a y T_a
- tipo V :Se alcanza la sobreaceleración máxima, se incluye T_s y T_j
- tipo VI : Se alcanza la sobreaceleración máxima y la velocidad máxima, se incluyen T_s, T_j y T_v
- tipo VII : Se alcanza la sobreaceleración máxima y la aceleración máxima, se incluyen T_s, T_j y T_a
- tipo VIII : Se alcanzan todas las restricciones máximas, se incluyen T_s, T_j, T_a y T_v

Finalmente, se cuenta con un mapa completo de como implementar el control de posición en el SCARA, la cual es implementar un controlador PID para cada uno de sus grados de libertad, definir una trayectoria optima, moviendolos desde un extremo a otro y determinar la ganancia optima utilizando optimización por PSO.

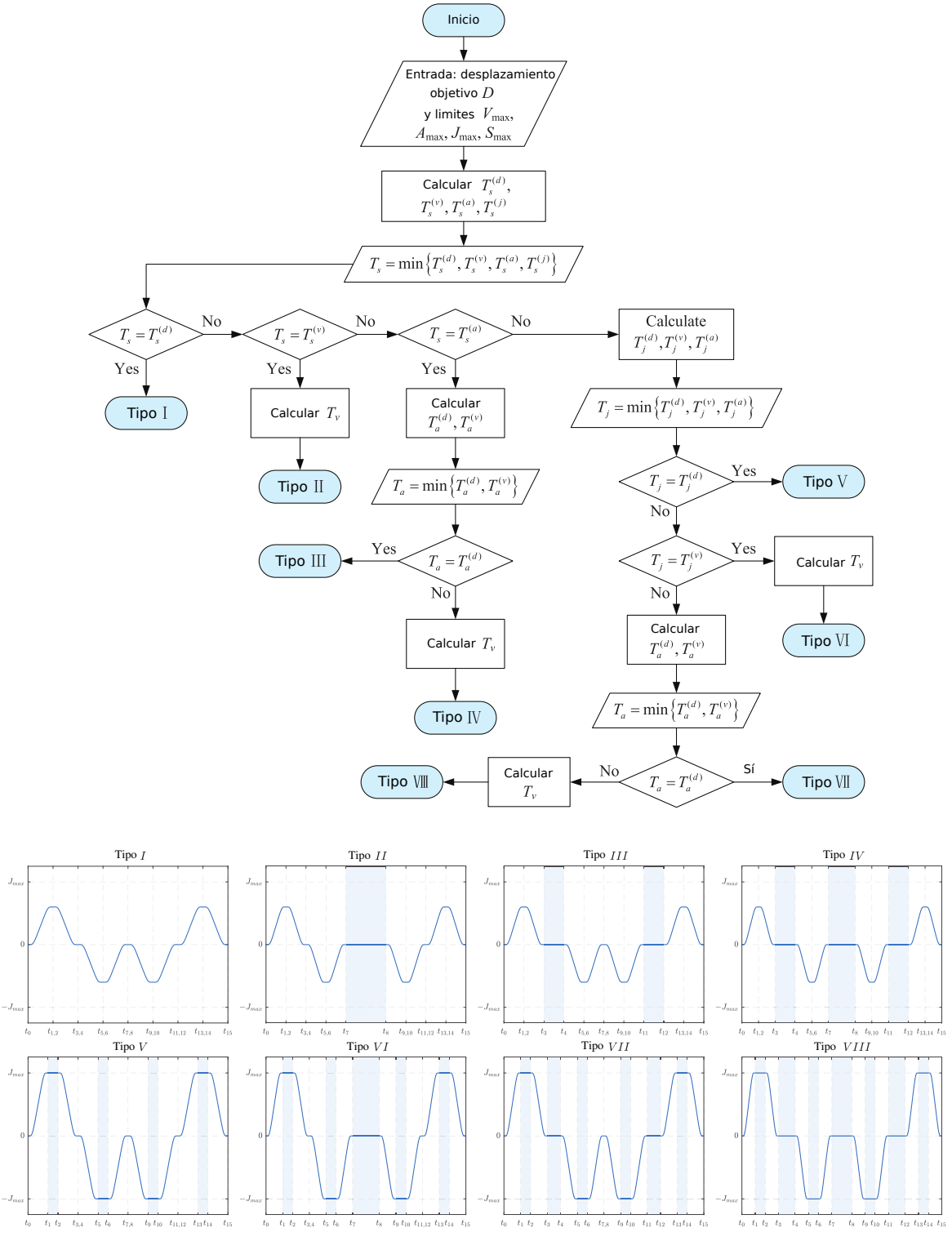


Figura 2.13: Mapa conceptual que resume el algoritmo para generar trayectorias suaves y óptimas manteniendo la sobreaceleración limitada. Traducida desde [27].

Capítulo 3

Hardware

Con el objetivo de vislumbrar cuales son los posibles componentes del robot SCARA, en el siguiente capitulo se ven cuales son las opciones de hardware que hay disponibles. Se abordarán, los tipos de motores eléctricos y sus respectivos drivers. Además se incluyen los tipos de encoders y sus características.

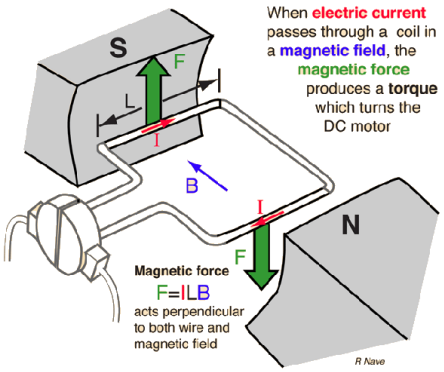
3.1. Motores eléctricos

Existen 2 grandes grupos de motores eléctricos, los de corriente alterna (AC) y los de corriente continua (DC). Para efectos de este trabajo se verán solo alternativas que ocupen motores de corriente continua (DC) y no se considerarán alternativas en corriente alterna. Por una parte, los motores de corriente alterna están pensados para conectarse de la red y su velocidad depende de la frecuencia de esta. Para controlar la velocidad de funcionamiento se deben incluir variadores de frecuencia por lo que su implementación tiene un costo mas elevado. Tienen una densidad de potencia menor [28], lo que significa que para la misma potencia tienen mas peso. Por último, en el caso de los motores asíncronos, estos tienen poco torque a bajas velocidades [29], como un brazo robótico tiene que parar y cambiar de dirección en gran parte de su funcionamiento esto no es un comportamiento deseado.

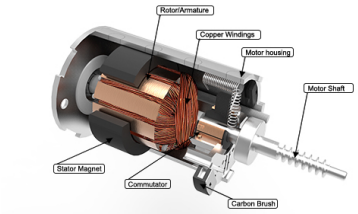
3.1.1. Motor de escobillas

Este tipo de motor corresponde al motor de corriente continua mas conocido. En la Figura 3.1 se muestra en la parte (a) su principio de funcionamiento y en la parte (b) sus principales componentes. Basa su funcionamiento en generar un campo magnético en el embobinado del rotor de tal forma que este sea perpendicular al campo magnético del estator. A medida que el rotor gira, se reduce la fuerza que genera este efecto, para solucionar este problema se disponen muchos embobinados en el rotor y estos se conectan y desconectan a medida que el motor gira, de tal forma que el que esta conectado siempre genera un campo magnético perpendicular al estator. La conexión y desconexión se realiza mediante un conmutador, el cual suele ser de carbón y que se desgasta rápido. Además, dado que los embobinados se conectan sin estar

energizados estos consumen mucha corriente y generan chispas durante su funcionamiento.



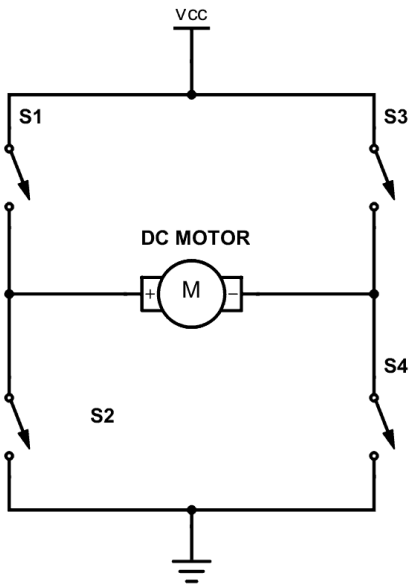
(a)



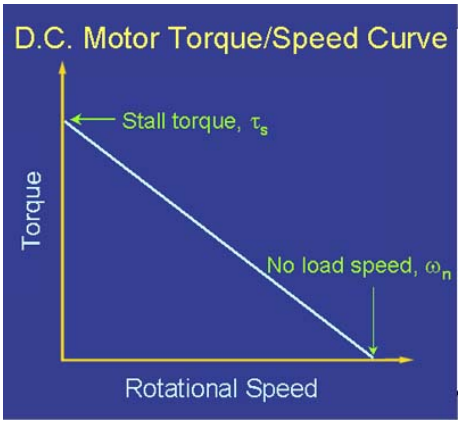
(b)

Figura 3.1: Motor DC de escobillas. En (a) se muestra el principio de funcionamiento, donde el paso de corriente genera un campo magnético perpendicular al del estator. En (b) se muestras sus principales componentes

Una ventaja de estos motores es que son fáciles de operar, basta con conectarlos una fuente de voltaje continuo. El voltaje aplicado esta relacionado directamente con la velocidad máxima del motor y un cambio en la polaridad genera un cambio en el sentido de giro. Típicamente son controlados mediante puentes H y una fuente de voltaje variable. En la Figura (3.2a) se muestra una topología de puente H, la cual consiste en 4 interruptores que permiten energizar el motor en ambos sentidos de giro y también implementar un freno. La relación del torque respecto a la velocidad angular es lineal como se muestra en la Figura (3.2b).



(a)



(b) Curva típica del torque del motor DC

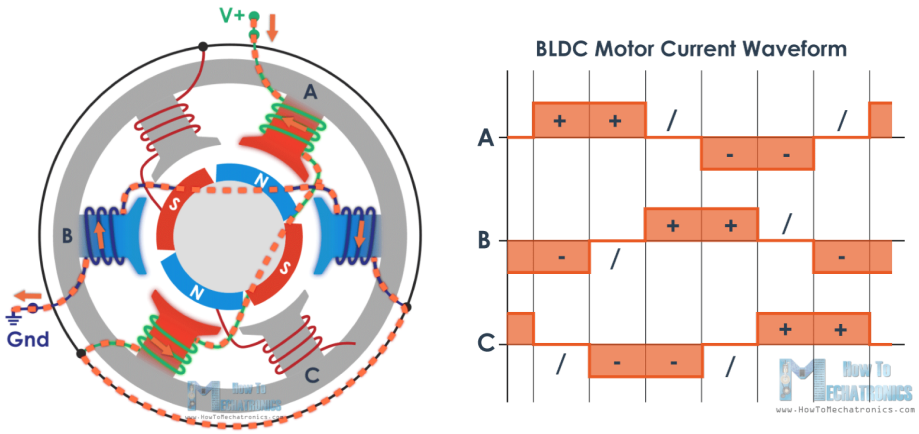
Para operar este tipo de motores existen una serie de alternativas, todas basan su funcionamiento en un puente H el cual regula el voltaje que el motor recibe mediante modulación por ancho de pulso (PWM). Existen circuitos integrados (IC's) con el puente H implementado para funcionar con motores DC, sin

embargo estas soluciones no son de mucha potencia, por ejemplo, la serie DRV8873, tienen un máximo de 10A y 38V. Opciones de mas potencias deben ser diseñadas específicamente para la aplicación, esto no es complicado, basta con encontrar transistores del voltaje y corriente necesarios e implementar la topología mostrada en la Figura 3.2a. Por último, encontrar un motor de alta potencia no es sencillo y son muy pesados para la potencia y torque que entregan, por ejemplo, el motor ZD2973, si bien es de 4kw y 6 NM, pesa 7.5Kg.

3.1.2. BLDC: Motor dc sin escobillas

El motor DC sin escobillas, BLDC por sus siglas en inglés (*BrueshLess Direct Current*), es similar al anterior, pero con la diferencia de que la conmutación del embobinado que se energiza se realiza mediante circuitos fuera del motor. Al no poseer un conmutador, se eliminan las escobillas, el componente que mas se gasta en un motor DC. Otra diferencia sustancial respecto a su par con escobillas es que se cambian los embobinados del rotor al estator, reduciendo el peso de este último y por consiguiente la inercia, esto hace que el tiempo de respuesta sea menor. Además, tienen una relación potencia vs peso mayor a cualquier otro tipo de motor [30].

En la Figura 3.3.a se muestra un diagrama simplificado del funcionamiento del motor BLDC. en esta se aprecia que existen 3 embobinados, los cuales tienen un nodo común conectado, similar a la conexión en estrella en los motores AC. Primero, se conecta a la fuente de voltaje la conexión restante del embobinado A con la del embobinado B, esto causa que A genere una fuerza que atrae al polo norte del rotor mientras que B lo empuja lejos de el. Posteriormente se energiza de forma similar los embobinados A y C, pero en esta ocasión se conecta la tierra a la conexión restante de A y C a la fuente de voltaje, de esta forma, el embobinado A repele el polo sur del rotor mientras que C lo atrae. Se repite este procedimiento consecutivamente, haciendo en total 6 pasos, según muestra la Figura 3.3.b.



(a) (b)

Figura 3.3: (a) Diagrama simplificado de la estructura, construcción y funcionamiento del motor BLDC. (b)Secuencia de energización del motor BLDC

Para caracterizar el desempeño de un motor BLDC se utilizan las constantes del motor, estas son la constante de velocidad y la contante de torque, K_v y K_t respectivamente. La constante K_v relaciona el voltaje aplicado al motor con la velocidad que este alcanza según la formula (3.1). Al girar el motor, los imanes permanentes inducen un voltaje en los embobinados, a mayor velocidad mayor es el voltaje, conocido como la fuerza contra-electromotriz. Por su parte la contante K_τ relaciona la corriente aplicada con el torque que se aplica en el eje del motor según la ecuación (3.2). Cuando se aplica corriente a una bobina esta genera un campo magnético, en un motor BLDC este campo magnético es contrario al campo magnético que tienen los imanes permanentes, esto genera una fuerza y por consiguiente un torque. Por último, existe una relación entre K_v y K_τ según la ecuación 3.3, la cual incluye una util aproximación. Esta relación se ve en detalle en [31]

$$K_v = RPM/Voltaje \tag{3.1}$$

$$K_\tau = Torque/corriente \tag{3.2}$$

$$K_\tau = \frac{3}{2} \times \frac{1}{\sqrt{3}} \times \frac{60}{2\pi} \times \frac{1}{K_V} \approx \frac{8.3}{K_v} \tag{3.3}$$

En [32] se obtiene una curva general de desempeño de los motores BLDC, la cual se muestra en 3.4. Además en este articulo se resaltan los puntos a tener en cuenta en una aplicación con este tipo de motores, los cuales son:

- La velocidad sin carga de un motor o *No load speed* esta restringida por el voltaje.
- El motor solo puede realizar trabajo bajo su velocidad sin carga.
- El torque constante de salida que el motor puede proporcionar esta restringido por la temperatura de los embobinados.
- La máxima velocidad a la que el motor puede realizar trabajo es la velocidad base.
- Un motor puede hacer mas torque su torque constante durante cortos periodos de tiempo.

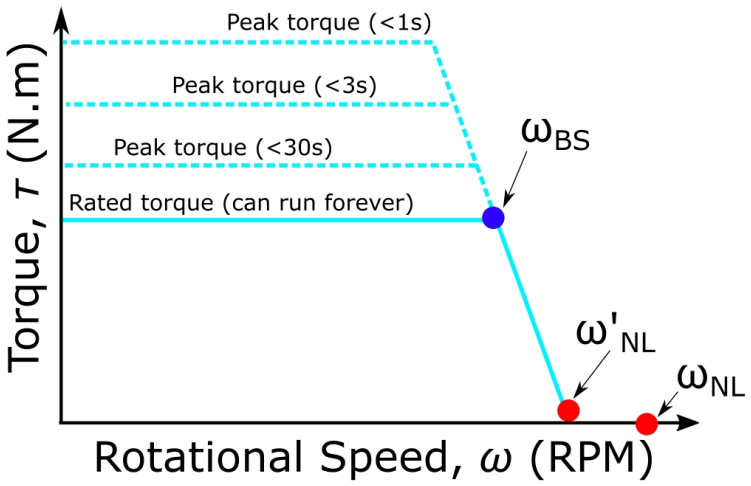


Figura 3.4: Curva de operación tipica de un motor BLDC. En el eje X se encuentra la velocidad angular, en el eje Y se muestra el torque.

La gran popularidad que tienen hoy los vehículos radiocontrolados (drones, autos, lanchas) ha hecho que el precio de este tipo de motor, como el de la electrónica necesaria para energizarlos sea muy barata. Para energizar un motor BLDC se utilizan 6 transistores en una topología similar a la del puente H. El dispositivo mas utilizado para operar motores BLDC es el ESC (*Electronic Speed Controler*). Este dispositivo controla la velocidad a la que opera un motor, para lograr esto realiza la secuencia mostrada anteriormente, modulando el voltaje de entrada y la frecuencia de este. Tal como su nombre lo indica, está pensado para controlar velocidad y no posición, principalmente en aplicaciones en vehículos, por lo que no es directamente aplicable en brazos robóticos.

$$\tau \approx \frac{8.3 \times I_A}{K_V} \tag{3.4}$$

Hasta hace unos años, no existía ninguna alternativa comercial para hacer control de posición de un motor BLDC, el control de posición de uso masivo era dejado principalmente para los motores paso a paso, los cuales se verán mas tarde en este capitulo. Una alternativa que puede realizar control de posición, que cuenta con distintos protocolos de comunicación, puede energizar hasta 2 motores de hasta 5kw y es *open-hardware* y *open source*, es *Odrive*. *Odrive* es un driver de motor pensado para aplicaciones que requieren precisión. Al utilizar motores de alta densidad energética (BLDC) en conjunto con encoders, logra alta precisión y velocidad por una fracción del precio de un motor servo, que es la alternativa comercial y típicamente utilizada en robots industriales que cumple la misma función. En la Figura 3.5 se muestra un Odrive y sus conexiones. Para operarlo es necesaria una fuente de voltaje, una resistencia de potencia, los motores y los encoders, además del dispositivo con el que se envían instrucciones al dispositivo. Esto puede realizarse desde un computador utilizando `Python`, o mediante un *arduino* utilizando I^2c entre otras opciones. Por último, por la gran velocidad, precisión y bajo precio que ofrece el control de motores BLDC mediante *Odrive*, se ha formado una amplia comunidad en torno a este. Hasta el momento se han realizado

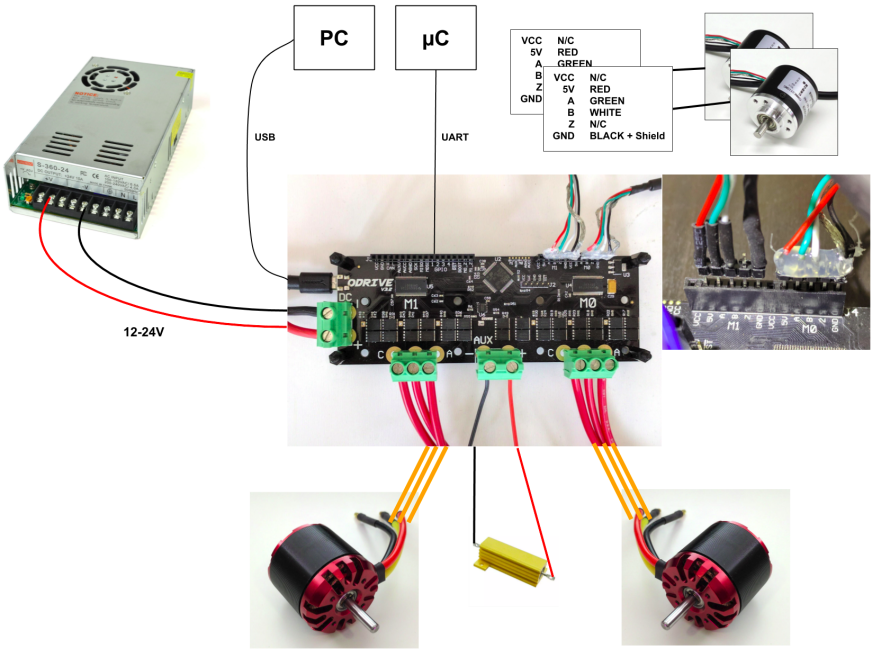
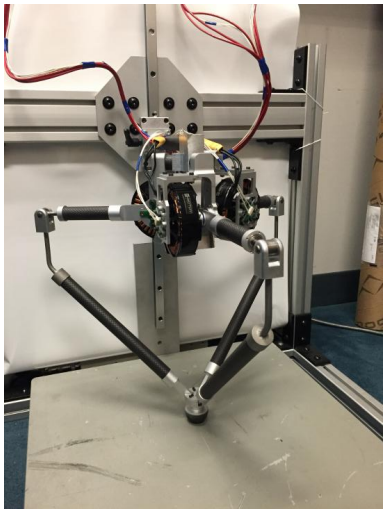


Figura 3.5: Odrive y como conectarlo

interesantes aplicaciones con este dispositivo, como lo son el caso del *Stanford Doggo* [33], el cual es un robot cuadrúpedo, de bajo costo y que se muestra en la Figura 3.6.a; o el *GOAT (Gearless Omnidirectional Aceleration Topology)*, una topología de pierna para robots y que se muestra en la Figura 3.6.b.

(a)



(b)

Figura 3.6: Aplicaciones de Odrive en robots de investigación. (a) Stanford Doggo (B) GOAT

3.1.3. Stepper motor

Un motor paso a paso, o motor *stepper*, corresponde a un tipo de motor DC que gira en pasos discretos. Tienen la ventaja de que pueden implementar control de posición sin necesitar con un encoder, es decir,

sin *feedback*. Además, están diseñados para mantener una posición fija, teniendo su máximo torque en velocidades bajas o totalmente detenido [34]. Su principio de funcionamiento consiste en un rotor con muchas piezas polares, y 2 sets de embobinados en el estator, los cuales en total, tienen menos piezas polares que el rotor y están dispuestos de tal forma que las piezas polares del rotor solo se pueden alinear con un set de embobinados a la vez, como ilustra la Figura 3.7. De esta forma, al energizar un embobinado el rotor se girará levemente para alinearse con la bobina encendida [35].

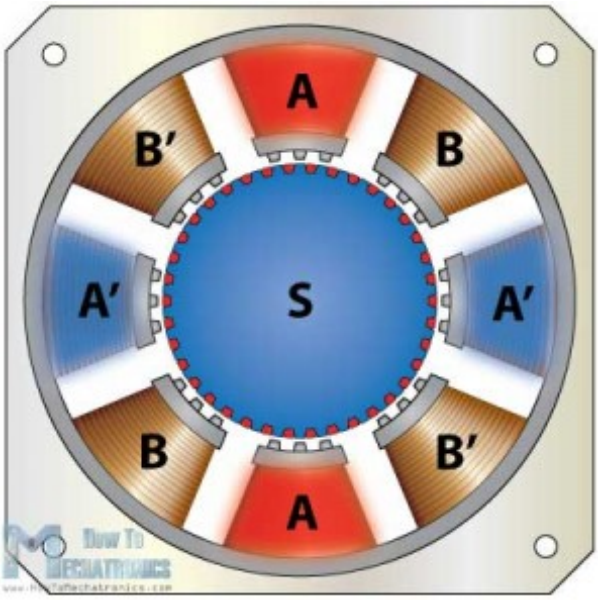


Figura 3.7: Diagrama que muestra al interior de un motor *Stepper* y su principio de funcionamiento

Su principal desventaja es la densidad de potencia que tienen, son mas pesados que otro tipo de motores para una misma potencia. Además, a medida que aumentan la velocidad su torque disminuye drásticamente. Otro problema que tienen este tipo de motores es que pueden perder pasos, de no tener un sistema de retroalimentación de la posición, es imposible saber cuando sucede esto. Aun con estas desventajas son una opción barata y sencilla para implementar control de posición.

Para energizar un motor paso a paso es necesario realizar una secuencia de encendido de una bobina a la vez. Existen múltiples circuitos integrados para realizar esto, uno de los mas usados es el DRV8825, el cual se vende a bajo costo en una pequeña PCB que se muestra en la Figura 3.8. Este recibe como entrada dos señales digitales, las cuales corresponden a `step` (paso) y `dir` (dirección), puede ser fácilmente operado con un microcontrolador, como *Arduino* por ejemplo.

3.2. Encoders

Los encoders son dispositivos que transforman movimiento mecánico en una señal eléctrica. Existen 2 grandes tipos, lineales y rotatorios. A continuación se verá las principales características a tener en cuenta a la hora de elegir un encoder.

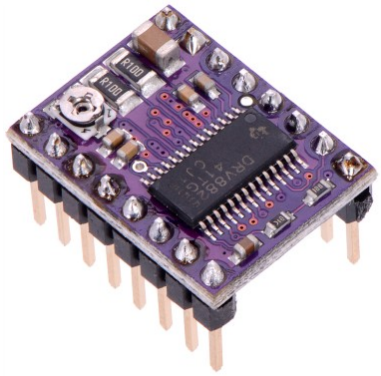


Figura 3.8: Driver de motor paso a paso DRV8825

(a) (b) (c)

Figura 3.9: Tipos de encoder según acople mecánico. (a) de eje, (b) directo a la salida, (c) Magnético.

- Conexión mecánica: Existen 3 tipos de conexiones mecánicas. El primero es de eje, este es un eje que se conecta mediante una correa o un acople al eje que se quiera sensar. El segundo tipo es a través del eje, este tipo se conecta directamente al eje de salida. El tercer tipo es magnético, se posiciona un imán sobre el eje, de tal forma que giren solidariamente, sobre este se posiciona un circuito integrado que sensa la posición del imán. En la Figura 3.9 se muestran los tipos de encoders discutidos, en (a), (b) y (c) según orden de aparición.
- Resolución: este parámetro corresponde a la cantidad de pasos que hay en una vuelta, lo que es inversamente proporcional a la mínima variación angular que se puede detectar. Se pide en $[count/rev]$ cuentas por revolución.
- Ruido: Es el error respecto a la variable real y la lectura del sensor causado por distintos motivos.
- Máxima velocidad: Es que tan rápido puede girar el motor sin dañar el encoder ni perder información de la posición
- Cuadratura: Se dice que un encoder es de cuadratura cuando cuenta con 2 salidas desfasadas eléctricamente en 90° de esta forma es posible saber la dirección de giro.
- Index: Corresponde a una salida digital extra que se activa una vez por vuelta, esto para utilizarse

como referencia.

- Absoluto o incremental: un encoder incremental genera una señal cada vez que se sobrepasa cierto umbral de mínimo ángulo, que tiene que ver con la resolución, para determinar el ángulo se mantiene una cuenta de los pasos dados. Un encoder absoluto tiene un patrón especial de tal forma que se puede inferir la posición de forma absoluta en cada lectura. En la Figura 3.10 se muestra en (a) el patrón de un encoder incremental y en (b) el de un encoder absoluto.

(a) (b)

Figura 3.10: Encoders incremental (a) y absoluto (b)

Capítulo 4

Metodología

4.1. Metodología utilizada

Como se comentó en la introducción, este trabajo de memoria es parte de un proyecto multidisciplinario entre ingeniería eléctrica e ingeniería mecánica. El objetivo principal de este es el diseño y la construcción de un brazo robótico tipo SCARA de calidad industrial mediante el apoyo de Beauchef Proyecta. El brazo robótico seguirá siendo desarrollado en todas sus áreas por alumnos de distintas carreras tanto en cursos como en futuros trabajos de memoria, por esta razón, se alinearon los objetivos para lograr una rápida construcción del brazo en cuestión. En la Figura 4.1 se muestra el diagrama de la metodología ocupada en este trabajo.

Para poder armar el brazo es necesario contar primero con una lista de componentes, por lo que la primera parte de la memoria se enfocó en esto. La elección de los componentes que la conforman se hizo teniendo en cuenta los **parámetros de diseño**, los cuales se obtuvieron mediante una revisión bibliográfica y de catálogos de robots similares de calidad industrial así como de otros proyectos similares tanto de las opciones abiertas *open-source* como de los documentados en revistas científicas y fabricados en otras universidades del mundo. En este ejercicio se determinó cuales eran los **componentes claves**, todos estos brazos contaban con servomotores con alto torque que les permiten moverse a gran velocidad. Con esto en consideración se revisaron los proveedores nacionales e internacionales, se separaron los componentes entre los que solo se podían comprar en el extranjero y los que se podían adquirir localmente. Luego, se prosiguió a elegir específicamente los componentes que serían usados en este robot que debían ser comprados en el extranjero, en esta etapa fue de crucial importancia los proyectos similares, en los cuales se detallaban proveedores y nuevas tecnologías alternativas, se tuvo como principal criterio de elección las limitantes presupuestarias de este proyecto, las cuales fueron determinadas previamente con la oficina de Beaucheff Proyecta. Luego, con los componentes a importar escogidos, se incluyeron el resto a comprar localmente y se determinó que el costo total de los elegidos cumplía con las restricciones presupuestarias.

Cabe mencionar el carácter práctico del proyecto y la importancia de cumplir con los plazos y el presupuesto.

Lo ideal hubiera sido poder haber modelado en un principio el desempeño del robot según las características de los motores a elegir y así poder compararlos, sin embargo, dado que era necesario realizar las compras lo antes posible, se priorizó tener la lista de componentes para realizar las compras de forma rápida. Esto resultó ser muy oportuno, pues poco tiempo después de generar la lista de compras y realizar las importaciones hubo en Chile un estallido social y una posterior pandemia, lo que hubiera ralentizado gravemente los procesos de compra.

Luego se realizó el modelamiento del brazo para determinar la estrategia de control más adecuada, en este ejercicio se consideraron solo las estrategias de control que pudieran ser implementadas en el robot dado los componentes elegidos, esto es sensato pues el objetivo del modelamiento y control es precisamente la operación en el robot real. Para realizar el modelamiento se utilizó el diseño en CAD del robot, el cual es uno de los principales resultados del memorista mecánico de este proyecto. Dicho diseño no solo guarda las relaciones dimensionales sino que además tiene información respecto a la densidad de los materiales y por ende las características dinámicas del robot. Para lograr hacer un modelo en base a este diseño se utilizó Gazebo, una de las herramientas de modelamiento de ROS. Con este modelo basado en el diseño exacto del robot se generó una trayectoria crítica y se afinó un controlador tipo PID con resultados satisfactorios en cuanto al seguimiento de dicha trayectoria.

Posteriormente se avanzó a la última etapa de este desarrollo, el proceso de armado. En este se hizo la última elección de componentes, principalmente cables para conectar todo. Esto presentó nuevos desafíos, principalmente de ubicación de componentes, pasadas de cables y conectores, lo que significó comprar materiales eléctricos en el mercado local además del diseño de placas de circuito que permitiera reducir la cantidad de conexiones a realizar. Se probó cada uno de los componentes por separado y utilizando los códigos para estas pruebas se armó el código final. Fue durante esta etapa cuando llegó a Chile la pandemia del Covid-19, como medida de protección la Universidad se cerro, por lo que se perdió el acceso a los laboratorios necesarios para continuar con el armado del robot. Por esta razón, a la fecha de entrega de este documento el brazo robótico aun no se encuentra terminado y las labores para lograr esto se retomaran apenas se bajen las cuarentenas y la universidad abra de nuevo sus puertas. Hasta entonces los memoristas involucrados en este proyecto siguen trabajando en la medida de lo posible, afinando el código y planeando futuras mejoras para implementar cuando todo vuelva a la normalidad. Apenas terminada la construcción se pasa a la última etapa, las pruebas, estas son de precisión, repetibilidad y velocidad.

4.1.1. Metodología original, no utilizada

Cuando empezó este proyecto se intentó seguir la metodología mostrada en 4.2. Esta se basaba en conseguir rápidamente un modelo para probar estrategias de control y también el desempeño de los componentes a elegir. Si bien esta es una metodología adecuada es muy demorosa, pues el modelo que se necesita para lograr todo lo que se propone es completo y no trivial de obtener. Las limitantes de tiempo y la priorización de la construcción por sobre cualquier otro objetivo hizo que esta metodología no fuera la mas adecuada.

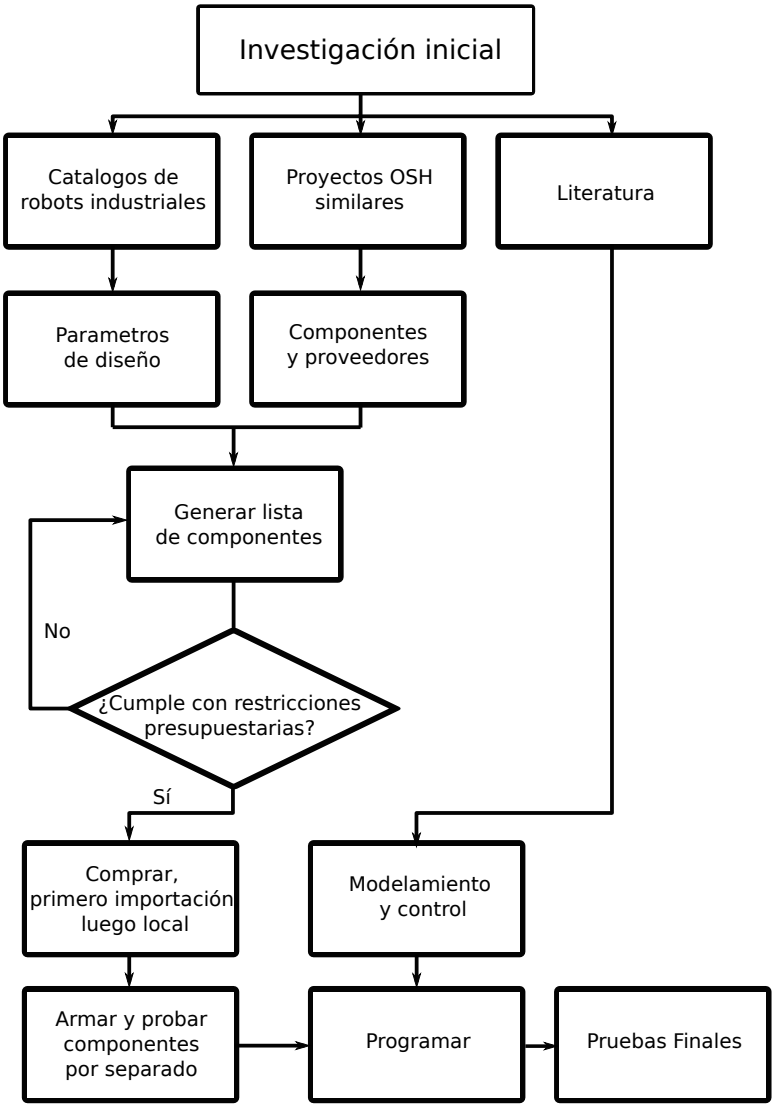


Figura 4.1: Diagrama de la metodología seguida en este trabajo.

Para trabajos futuros similares esta metodología puede servir, ocupando un proceso de modelamiento similar al ocupado en esta memoria, ya sea una versión corregida y adaptada del modelo fenomenológico o un diseño en CAD y simulado en Gazebo mediante un URDF.

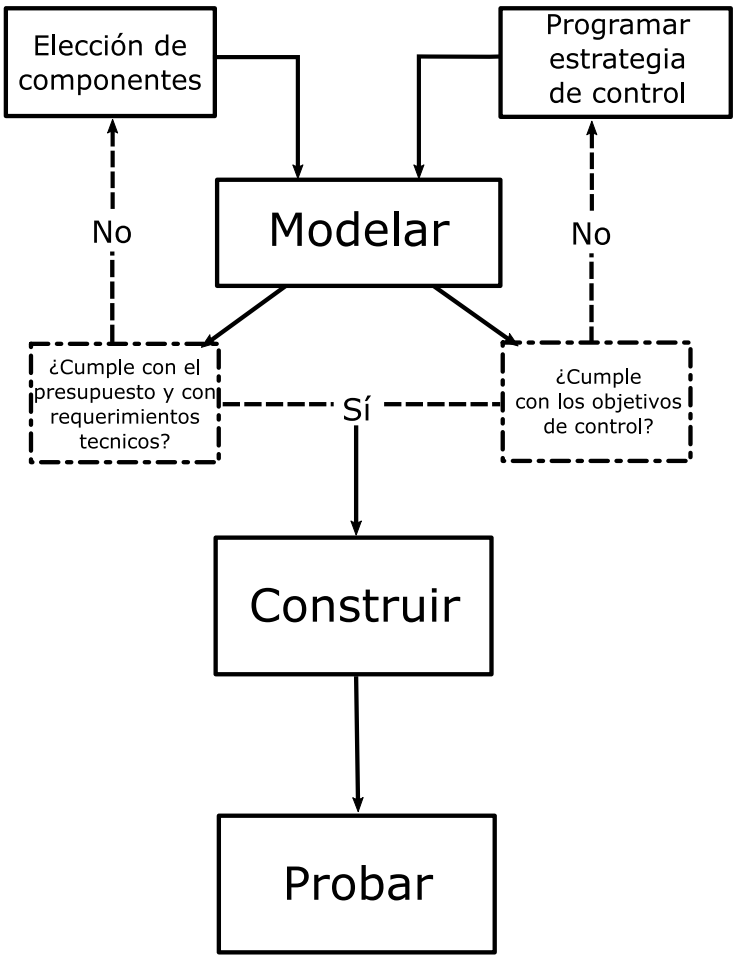


Figura 4.2: Diagrama de la metodología pensada originalmente.

Capítulo 5

Elección de componentes

El presente capitulo corresponde a uno de los principales resultados de este trabajo que es la elección de los componentes eléctricos que componen el SCARA. Dentro de cada sección se detalla el material o dispositivo escogido y cual fue el razonamiento para dicha elección. En esta sección particularmente, se ahonda en los elementos que componen físicamente el robot que constituye una de las decisiones más importantes ya que impone limites a su desempeño. Por ejemplo, el torque de los motores fija la velocidad máxima o por otro lado la resolución de los encoders limita la precisión del robot.

- Odrive
- Motor Herlea 8325
- Motor Herlea 8318
- Motor Herlea 5408
- Motor Stepper NEMA 17
- Encoder AMS5047p
- *Raspberry Pi*
- MLX90614
- Fuente de poder

5.1. Motores

La primera decisión que se tomó en el diseño de este robot fue la utilización de motores BLDC controlados mediante *Odrive*. En la lista expuesta a continuación se incluyen los motivos que impulsaron la elección de estos componentes:

1. Precio: los motores BLDC son fabricados en masa por la alta demanda de drones lo que ha aumentado su accesibilidad en el mercado.

2. Potencia: dentro de los motores estudiados el que cuenta con la mayor densidad de potencia (Potencia/Peso) es el motor BLDC. Por otra parte, el Odrive puede alimentar hasta 2 motores de 5KW.

3. Flexibilidad: el controlador Odrive puede ser utilizado en diferentes modos, con cualquier tipo de motor BLDC y con encoders con distintos tipos de salida, resolución, principio de funcionamiento, etcétera. Además cuenta con comunicación mediante USB, CAN, ASCII, STEP DIR, entre otros.

4. Buena documentación: el equipo de Odrive cuenta con guías de encoders, motores, puesta en marcha además de un activo foro donde otros usuarios exponen sus experiencias y problemáticas lo que hace que la mayoría de los inconvenientes tengan solución documentada.

5. Tendencia: como se expuso anteriormente existen interesantes aplicaciones de robots de bajo costo que utilizan esta placa y existen otros desarrollándose actualmente. Además, si bien su última versión es de hardware cerrado, las anteriores son de hardware libre como también lo es y seguirá siendo todo el software para operar esta gran herramienta.

Asimismo, es necesario elegir los motores a utilizar, lo que a nivel de mercado nacional supone una restricción importante ya que la oferta de motores BLDC es escasa y los pocos encontrados corresponden a motores de tiendas de vehículos radio-controlados los cuales poseen torque y potencia reducidos. Por esta razón, se decidió importar todos los motores juntos de un mismo proveedor para abaratar costos de envío. El criterio ocupado para la búsqueda fue encontrar el motor que pudiera producir mayor torque en el rango de funcionamiento del Odrive (90A 56V). En esta búsqueda y con base en el proyecto Open-Torque [36], un actuador impreso en 3D de hardware abierto y que utiliza motores BLDC, se encuentra al proveedor *Herlea* quien vende mediante la tienda en linea *Alibaba* y que recibe diferentes métodos de pago, entre ellos *PayPal*. En la tabla 5.1 se indican los motores elegidos y sus especificaciones eléctricas. El motor en el eje del hombro no tiene restricciones de peso pues su masa no influye en el desempeño del brazo por estar justo en su punto fijo. Sin embargo, los otros motores sin se eligen con esa restricción ya que deben tener el mínimo peso posible sobre ellos.

Tabla 5.1: Motores elegidos y sus características

Motor	Constante de velocidad Kv	$I_{max}[A]$	$V_{max}[V]$	Potencia $[KW]$	Torque $[Nm]$
8325	120	90A	50	4.35	7.21
8318S	120	69	48	3.35	4.95
5408	260	28	33.3	810	0.89

5.2. Encoder

A partir de la documentación de *Odrive* se utiliza la guía de encoders [37] que contiene múltiples modelos con distintos principios de funcionamiento, además de sus características eléctricas y precios.

El encoder que el equipo de *Odrive* recomienda utilizar es el CUI AMT 102 con conexión mecánica a través del eje que tiene un costo de importación de 24 USD. El sitio oficial de *Odrive* comercializa motores con eje por ambos lados lo que hace sencillo implementar este encoder, sin embargo, realizar esto con los motores escogidos implica el diseño de una pieza extra para adaptarlo. Por otra parte, según la guía, este encoder tiene una resolución de 8192 pulsos por vuelta y su salida es de cuadratura, por lo que en cada reinicialización del dispositivo se reinicializa también la cuenta del encoder, dicho de otra forma, solamente se puede saber cuanto se ha movido desde el instante de reinicialización.

El *Odrive* está pensado para utilizar encoders de cuadratura. Una vez que se energiza el dispositivo se reinicializa la cuenta de los pulsos del encoder. Como se expuso anteriormente, existe una secuencia de calibración del motor BLDC, por lo que es necesario determinar el *desfase* de la posición actual (cuenta=0) con el 0 de la secuencia de energización del motor. Por esta razón, para que el *Odrive* entre en el estado donde mueve el motor de forma controlada es necesario que estime este desfase y para realizar esta estimación posee un procedimiento determinado. Este procedimiento consiste en energizar el motor partiendo deesde un punto aleatorio de la secuencia de energización hasta que el motor comienza a moverse, una vez en movimiento lleva el motor hacia una dirección y luego a la contraria. La realización de esta acción no supone ningún problema si el motor no tiene restricciones físicas ya que puede girar libremente, sin embargo, el SCARA tiene limitaciones en 3 de sus 4 movimientos y puede ocurrir por ejemplo, que el brazo parta energizado muy cercano a uno de sus limites físicos lo que haría al motor chocar y terminaría en un proceso de calibración erróneo o algún tipo de daño. Esta situación que si bien parece poco probable, sucede siempre para el eje Z, porque por efecto de la gravedad siempre parte en su posición mas baja y retenido por un limite físico. Por esta razón **es necesario utilizar un encoder absoluto**.

Al revisar otras opciones indicadas en la guía se opta por el encoder magnético absoluto AMS AS 5047P que tiene una resolución de 14 bits (16384 pasos), se vende en una placa de evaluación que incluye el sensor magnético y sobre todo, porque cuenta con 2 salidas, una absoluta que se comunica mediante SPI y una de cuadratura, además su costo es menor ($15.75 USD). Ambas salidas funcionan independientes la una de la otra y también pueden funcionar en paralelo lo que permite conectar la salida de cuadratura al *Odrive* y la salida absoluta al computador del robot mediante SPI.

Esta evaluación se realizó previamente para elegir y comprar este encoder, sin embargo, en esta decisión se omitieron 2 detalles importantes expuestos en la hoja de datos [38]. En primer lugar, la resolución de la salida absoluta es en efecto de 14 bits (16384 pulsos por vuelta), pero no así la resolución de la salida de cuadratura que tiene una resolución de fabrica de 4000 pulsos por vuelta. Esta diferencia puede hacer que la resolución se vea reducida, pues si bien el computador del robot podrá ver con mayor resolución,

el controlador no verá lo mismo. Esto se puede resolver al utilizar el *Odrive* en el modo de control de corriente y ocupar la salida de mayor resolución desde el computador para elegir la corriente. Otra solución a este problema es utilizar el encoder en su versión U (AMS5047U), el cual tiene igual resolución para ambas salidas. Al momento de adquirir los encoders solamente se encontraban en su versión D y P ambos con 14 bit de resolución en su salida absoluta y con resolución de 11 y 12 bits para su salida en cuadratura respectivamente. Se elige entonces la versión P por su mejor resolución, pero en caso de replicar este robot es altamente recomendado comprar la versión U, la cual tiene la misma resolución para ambas salidas y tiene calculo de la velocidad angular accesible mediante SPI.

En segundo lugar, como se muestra en la hoja de datos, en la pagina 7, Figura 8, el descentrado del imán puede causar errores de hasta 1.2 grados, esto equivale a 54.61 pasos de los 16384 de la resolución. En [39] se muestra que este error tiene alta repetibilidad y se presenta en posiciones fijas de los 360° de recorrido del imán, como se indica en la Figura 5.1 que muestra una prueba realizada con un motor *stepper* y cuya posición angular se mide con el encoder. En la figura, se muestra el error en la duración de una vuelta completa donde cada color corresponde a una vuelta y se aprecia que el error causado por el descentrado es repetitivo, periódico y que además, esta dentro de los rangos mencionados en la hoja de dato. El enfoque que se toma en este estudio es generar un *look up table* (una tabla con todas las posiciones y su error asociado) del error mostrado y restarlo a la posición medida, de esta forma el error en la medición será solo producto del ruido.

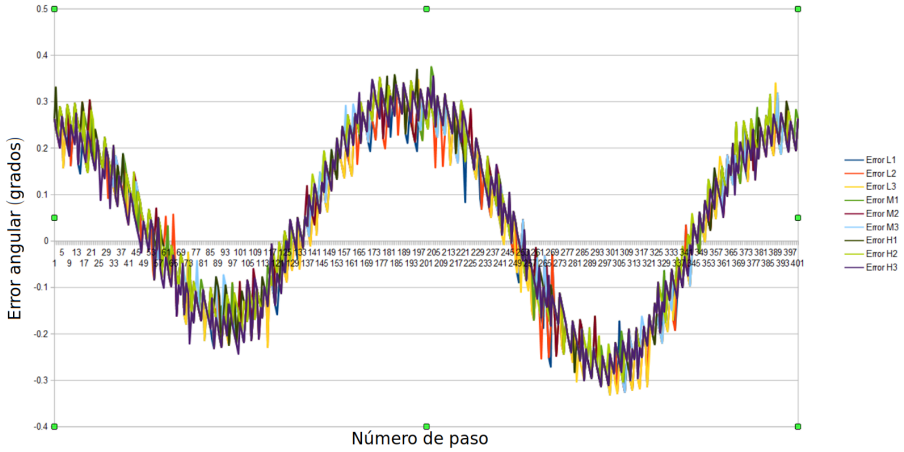


Figura 5.1: Error en grados de una rotación completa de un motor *stepper*. Cada color corresponde a una realización (vuelta) distinta. Traducida de [39].

5.3. Computador y otros circuitos integrados

Para completar este proyecto se eligió utilizar una *Raspberry Pi* Pi como computador para el robot lo que se hizo principalmente por una restricción de precio. En el mercado un computador usado común y corriente puede encontrarse por alrededor de 200.000 CLP y en comparación a una *Raspberry Pi* que tiene

un precio de 41.100 CLP en su presentación 3B se posiciona como una opción mucho más asequible. Por otra parte, uno de los grandes objetivos a la hora de diseñar el robot es la replicabilidad y al utilizar una *Raspberry Pi* es posible generar una ïmagen.es decir, una copia exacta del sistema operativo con todos los programas y drivers instalados lista para usarse en el dispositivo y así el usuario solamente necesita copiar la información a una tarjeta SD e implementar directamente. Además, cuenta con un puerto GPIO de 28 entradas salidas, compatible con los protocolos de comunicación mas usados entre ellos I^2C y SPI, además de 4 puertos USB, conexión Ethernet, bluetooth y wifi.

Como la vida útil de los motores y el máximo torque que pueden hacer sin dañarse dependen estrechamente de la temperatura es necesario monitorearla y disminuirla en caso de que esta sobrepase los limites de operación seguros. Para esta tarea se decidió usar el sensor de temperatura MLX90614 para monitorear la temperatura y distintos ventiladores de 12 V rescatados de procesadores de computador antiguos para controlarla. El MLX90614 es un sensor óptico de alta precisión que se comunica por I^2C como se muestra en su hoja de datos [40]. La elección de este sensor se realizó debido a su flexibilidad de implementación, dado que no es necesario tocar la superficie que se quiere sensar, en este caso las bobinas del motor.

Se incluyó un expansor de entradas/salidas digitales y se eligió utilizar el circuito integrado MCP23017, el cual se comunica por protocolo I^2C y suma 16 entradas/salidas digitales. Este dispositivo se integró posteriormente para juntar todo el cableado de los sensores de fin de carrera (*limit switch*) y enviar esta información por I^2C, reduciendo así la cantidad de cables.

Por último se incluyó un Relé de estado solido marca Fotek para operar herramientas que se incluyan en la punta del robot. Este dispositivo tiene la capacidad de hacer modulación por ancho de pulso para regular la carga que se le conecta, por ejemplo, se puede utilizar para regular la potencia de un cautín y así controlar la temperatura de este o también, para regular el voltaje que recibe un taladro y con esto su velocidad de giro.

5.4. Fuentes de poder

En esta sección se discuten las cuatro fuentes de poder que se utilizaron para alimentar los consumos del robot y se enumeran a continuación: el computador (*Raspberry Pi*), los distintos ventiladores, los *Odrive* que a su vez alimentan los motores BLDC y el motor *Stepper*.

Para la alimentación de la *Raspberry Pi* se decidió utilizar un cargador de celular USB que es la fuente oficial y recomendada de alimentación. Más adelante en este capitulo se expondrá el diseño de una PCB y como trabajo futuro se recomienda diseñar un circuito de alimentación según el procedimiento visto en [41].

Para la alimentación de los ventiladores se decidió utilizar una fuente de 12V y 100W. Además esta fuente de poder se utilizó para la energización del motor stepper.

Para alimentar los *Odrive* se eligió una fuente de voltaje constante. Para hacer esto se necesitó determinar el voltaje y la corriente donde ambas variables dependen del uso que se le dará a los motores que se alimentarán. Se sigue el proceso ocupado en [42] que se basa en elegir el voltaje de la fuente según la velocidad máxima y la potencia según la corriente máxima. Además, se utilizaron las ecuaciones 3.1 y 3.2 para calcular el voltaje y la corriente.

El procedimiento citado considera solamente un motor pero para este escenario donde estarán funcionando 3 motores al mismo tiempo se elige el voltaje máximo entre los 3 motores y se suman las potencias de los 3. De esta forma la fuente es dimensionada para la condición de operación mas demandante del robot. Además, hay una variable a considerar que altera el torque y la velocidad a la que gira el motor que son las cajas reductoras para los motores del codo y el hombro y también el husillo de bolas en el caso del eje Z. Una caja reductora, valga la redundancia, reduce la velocidad de salida según su relación de reducción y aumenta el torque según el inverso de la relación de reducción. Por otra parte, un husillo de bolas genera un efecto similar al de la caja reductora, donde el paso del tornillo que es la distancia lineal que avanza por cada vuelta, es equivalente a la relación de reducción. En la expresión 5.1 se muestra la relación entre el torque necesario T_{carga} en el eje de un husillo de bolas con paso p, con una eficiencia eff para hacer una fuerza F.

$$T_{carga} = \frac{F * p}{2000\pi eff} \tag{5.1}$$

En la Tabla 5.2 se muestra el calculo de la potencia y el voltaje que requiere cada motor, las que se calcularon según el siguiente procedimiento. La velocidad a la que giran los motores se obtuvo de los parámetros de diseño originales $8m/s$ de movimiento en el plano y $1m/s$ en el eje Z. Respecto a la velocidad en el plano, con las velocidades angulares que se muestran en la tabla se alcanzaría una velocidad en el plano de $10.94m/s$, este valor es mayor que los parámetros de diseño lo que hace que se genere un margen de seguridad. Respecto al eje Z la relación entre la velocidad del motor y la velocidad del eje Z es directa. Para el torque máximo de los motores del codo y del hombro se considera la máxima corriente que los motores pueden entregar y el torque asociado a estas. Para el caso del motor del eje Z se considera una aceleración ($5m/s^2$) y una masa que se mueve a dicha aceleración ($7kg$), con esto se obtiene una fuerza ($103.6N$) que se usa en la ecuación 5.1 para determinar el torque en el eje del husillo de bolas y por consiguiente el motor. Cabe mencionar que tanto para las reductoras como para el husillo de bolas se consideró una eficiencia del 90 %, que es el valor típico mas bajo para estos mecanismos.

De la Tabla se concluye que la **fuente de poder a elegir debe ser de al menos 18.02 V y 1423 W** con un factor de seguridad de 25 %. Para adquirir esta fuente de poder se buscaron proveedores nacionales y se encontraron las empresas Zona Industrial y Power Inverter. En la Tabla 5.3 se indican las opciones que se consideraron y sus precios. Por las restricciones de precio se opto por la opción de 960W principalmente, por la alta diferencia de precio que existe con todas las otras alternativas. Si bien la potencia es 2/3 de la calculada, esta última considera el peor escenario ya que en su estado normal los motores serán acelerados

Tabla 5.2: Tabla resumen de condiciones de operación criticas para el calculo de la fuente de poder.

Articulación	Hombro	Codo	Z
Reducción o paso	1:7	1:7	16 mm
Eficiencia	0,9	0,9	0,9
Motor	8325	8318s	5408
KV	120	120	260
Salida V_{max}	95 RPM	127 RPM	1000 mm/s
Salida tau (output)	39 Nm	30 Nm	103,6 N
Entrada V_{max}	665 RPM	889 RPM	3750 RPM
Torque (input)	6,19 Nm	4,76 Nm	0,29 Nm
Voltaje motor	5,54 V	7,41 V	14,42 V
Corriente máxima	89,50 A	68,85 A	9,18 A
Potencia	495,98 W	510,04 W	132,44 W

Tabla 5.3: precios fuentes de poder disponibles

Modelo	Voltaje	Potencia	Precio CLP	Proveedor
SE-1500-48	48 V	1.5 KW	$300.110	Zona industrial
RSP-1500-48	48 V	1.5 KW	$422.510	Zona industrial
SE-1000-48	48 V	1 KW	$192.180	Zona insudtrial
RSP-1000-48	48 V	1 KW	$270.690	Zona industrial
Power inverter 1440	48 V	1.44 KW	$226.100	Power inverter
Power inverter 960	48 V	0.96 Kw	$59.500	Power inverter

todo el tiempo y cuando estén acelerando no será cuando se encuentren en su mayor velocidad, en otras palabras, cuando se requiera la mayor corriente no será con el mayor voltaje. Para suplir estas abruptas demandas de potencia se recomienda utilizar un banco de condensadores adecuados o una batería, sin embargo, este calculo y posterior diseño se proponen como trabajo futuro en caso de que exista una falta de potencia en el uso normal del robot.

5.5. Printed Circuit Boards's

Para este robot se diseñaron **3 PCB's** (placas de circuito impreso en inglés): una placa de circuito de expansión para la *Raspberry Pi* o **pseudo HaT**, una placa de conexión para los encoders o **placa de distribución encoders** y una placa para conectar los dispositivos que ocupan I^2C, los limites de carrera, y los ventiladores o **placa** I^2C.

El objetivo de la placa de circuito de expansión para la *Raspberry Pi* o pseudo HaT, es proveer todas las conexiones que llegan al GPIO. Un *Hardware on top* (HaT) es una placa de circuito que cumple con especificaciones de tamaño, corriente y componentes, que se usa para agregar *hardware* a una *Raspberry Pi* de forma de asegurar compatibilidad. El circuito diseñado es algo similar, no se procurará cumplir con la especificación para que sea considerado un HaT pero si se tomarán muy en cuenta las recomendaciones de diseño. En la siguiente lista se enumeran las funciones de esta placa de circuito.

- Conectar puertos I^2C y SPI

Tabla 5.4: Resumen de características eléctricas del optoacoplador PC817. Obtenidas de la hoja de datos [44].

Caracteristica	Valor	Unidades
Corriente de encendido led	20	mA
Voltaje encendido led	1.2	V
Corriente de salida máxima	50	mA
Voltaje de salida	0.1-0.2	V
Tiempo de subida	4 - 18	μs
Tiempo de bajada	3-18	μs

- Encender y apagar ventiladores
- Comunicarse con el driver de motores DRV8825
- tener salidas digitales para encender y apagar herramientas que se pongan en la punta.

En primer lugar es necesario saber los limites de corriente de la *Raspberry Pi* para evitar dañarla. En [43] se discuten los limites eléctricos de este dispositivo, **el GPIO tiene un limite de voltaje de 3.3 V y un limite de corriente de 16 mA para cada salida y 50 ma entre todos**. Para evitar sobrepasar el limite de voltaje se diseñaron circuitos basados en optoacopladores. Este circuito integrado consiste en un led que ilumina un fototransistor, lo que permite que se puedan conectar lógicamente 2 circuitos y mantenerlos separados eléctricamente, esto se hace precisamente para separar el delicado circuito de la *Raspberry Pi* de todo lo que está alimentado con la fuente de 12V. Además, se aprovecha la fuente para hacer salidas y entradas digitales con un nivel de voltaje mas alto y mas corriente. Se decidió utilizar el optoacoplador PC817, que es uno de los mas usados y más fácil de encontrar en el mercado nacional. De la hoja de datos se extraen, entre otras cosas, las características de encendido y los tiempos de subida y bajada tal como se indican en la Tabla 5.4.

Para hacer una entrada digital se utiliza el circuito de la Figura 5.2. En este circuito la entrada del optoacoplador es encendida por la fuente de 12V cuando se conectan los pines del jumper que está a la izquierda. Esto genera una corriente en la salida del optoacoplador y en este caso como está limitada por una resistencia de 10K, genera una corriente de 0.32 ma, el cual es un valor conservador y adecuado para los limites de corriente de la *Raspberry Pi* mencionados anteriormente. Además se incluye una resistencia de 10K en serie a la entrada para limitar la corriente en caso de falla.

En la Figura 5.3 se muestra el circuito de salida. Como se mencionó anteriormente, la corriente máxima que puede suplir cada salida digital del GPIO de la *Raspberry Pi* tiene como máximo 16 mA, por lo que es necesario utilizar un transistor para generar esta corriente desde el nodo de 5V cuya corriente máxima depende de la fuente con la que se alimenta la *Raspberry Pi*, en este caso, considerando las cargas que tendrá el puerto usb, se supone como máximo 1 A para todo el nodo de 5V. Se decide utilizar el transistor NPN MMBT2222A, también uno de los mas utilizados, que tiene una corriente de salida máxima de 600 ma, soporta voltajes de hasta 40V y el rango de corriente en la base es cercano a los 10ma y

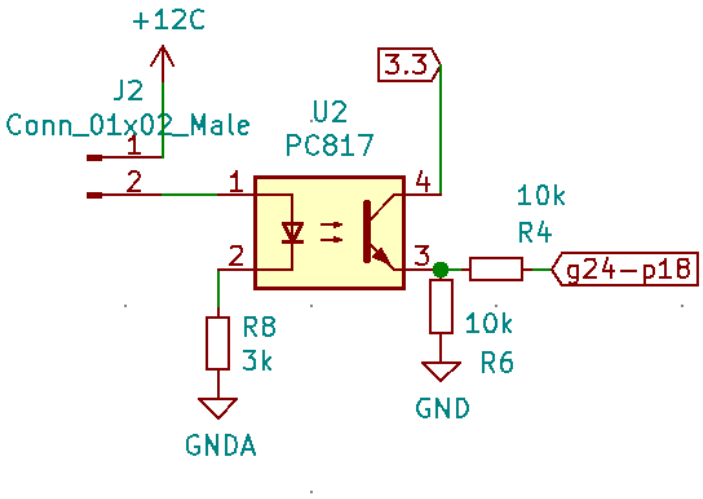


Figura 5.2: Esquematico entrada digital

tiene una ganancia de 70. De esta forma, la parte izquierda del circuito se encarga de encender el led del optoacoplador cuando se enciende la salida digital de la *Raspberry Pi*, nótese que la resistencia conectada a la salida del GPIO es de 10K por lo que la corriente al encenderse es de 0.33 mA y por ende la corriente de la salida del transistor será de 23 mA. Por otra parte, en la parte derecha se encuentra la salida que es alimentada por la fuente de 12 V. Primero, se encuentra un divisor de voltaje que se utiliza para generar un nivel de voltaje adecuado limitando la corriente que fluye por el fototransistor del optoacoplador.

Luego, se encuentra un MOSFET de canal N (que se enciende con voltaje positivo) modelo NTR4501N, el cual tiene un limite de corriente de 3.2A y de voltaje de 20V que se utiliza para prender un ventilador, cabe destacar que de los ventiladores utilizados en este robot ninguno supera 1 A de corriente cuando se encienden. Se decide utilizar un MOSFET sobre un relé pues es mas asequible, mas pequeño, mas duradero y permite hacer modulación por ancho de pulso para regular la salida. Un MOSFET tiene 3 formas de operación, apagado, modo ohmnico y modo saturado y la zona de operación depende de la relación entre el voltaje en *gate* V_{gate}, el voltaje en *drain* V_{drain} y el voltaje de umbral o *threshold* V_{th}, que es una característica del MOSFET (este valor es menor a 1.25 V en el MOSFET). Si se cumple que $V_{gate} \leq V_{th}$ entonces el dispositivo está apagado, si $V_{gate} > V_{th}$ y $V_{drain} < V_gate - V_{th}$ entonces el dispositivo está en modo ohmnico y actúa como una resistencia que depende de V_{gate}, si $V_{gate} > V_{th}$ y $V_{drain} \geq V_gate - V_{th}$ entonces el dispositivo esta en su modo de saturación y actúa como una resistencia pequeña y permite el paso libre de corriente (hasta su limite), en otras palabras un interruptor DC que es precisamente lo que se requiere. Esta descripción es una simplificación funcional del funcionamiento de los MOSFET y se obtiene de [45], donde se detalla una explicación mucho más acabada de el. Del divisor de corriente de la salida se obtiene un nivel de voltaje de 11 V, que esta conectado al *gate* del MOSFET, por otra parte el *drain* está conectado a los 12V en serie con el ventilador o carga a conectar mediante el jumper del lado derecho, a su vez como el voltaje de umbral es 1.2V se cumple que $V_{gate} = 11 > V_{th} = 1.2$ y

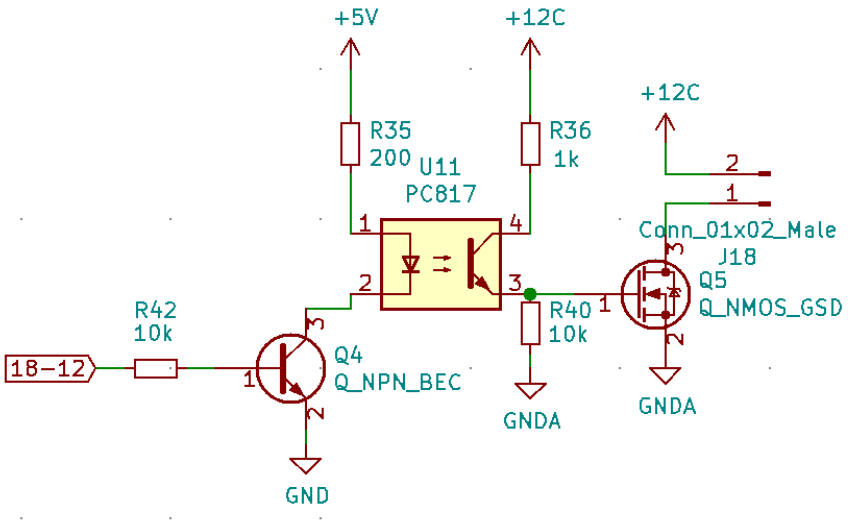


Figura 5.3: Esquemático de salida digital de potencia.

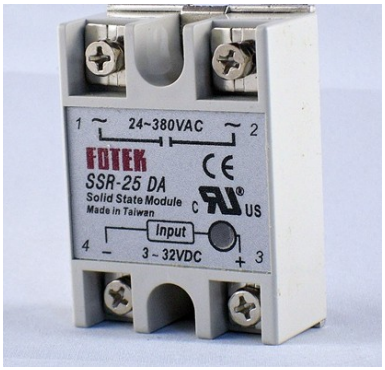


Figura 5.4: Relé de estado solido marca Fotek

$V_{drain} = 12 \geq V_gate - V_{th} = 10.8$ por ende este circuito se comporta como un interruptor DC.

El circuito recién visto se puede utilizar para hacer una salida digital con un nivel de voltaje regulable según el divisor de voltaje visto anteriormente y se usa precisamente así en 2 aplicaciones. La primera aplicación es como salida digital multiproposito con el objetivo de encender un relé de estado solido, como por ejemplo el expuesto en la Figura 5.4 aunque puede ser conectado directamente a la *Raspberry* pues se enciende con voltajes de hasta 3V y consume una corriente de 7.5 mA, sin embargo, se utiliza un optoacoplador para mas seguridad. Por otra parte, el circuito de salida digital se ocupa para operar el driver de motor stepper para el efector final.

En la Figura 5.5 se muestra el circuito de conexión con el DRV8825. Este circuito es la topología de salida digital vista anteriormente repetida 3 veces, una para la señal de paso *step*, la cual al subir y bajar produce que el motor avance un paso; otra para la señal *dir*, la cual cambia el sentido de giro del motor según si está encendida o apagada; y otra para la señal *enable*, que determina si el motor es energizado o no. Al estar energizado, el motor stepper mantiene su posición y no es posible girarlo con la mano, además, se ha incluido un condensador en la entrada de alimentación según las recomendaciones del fabricante.

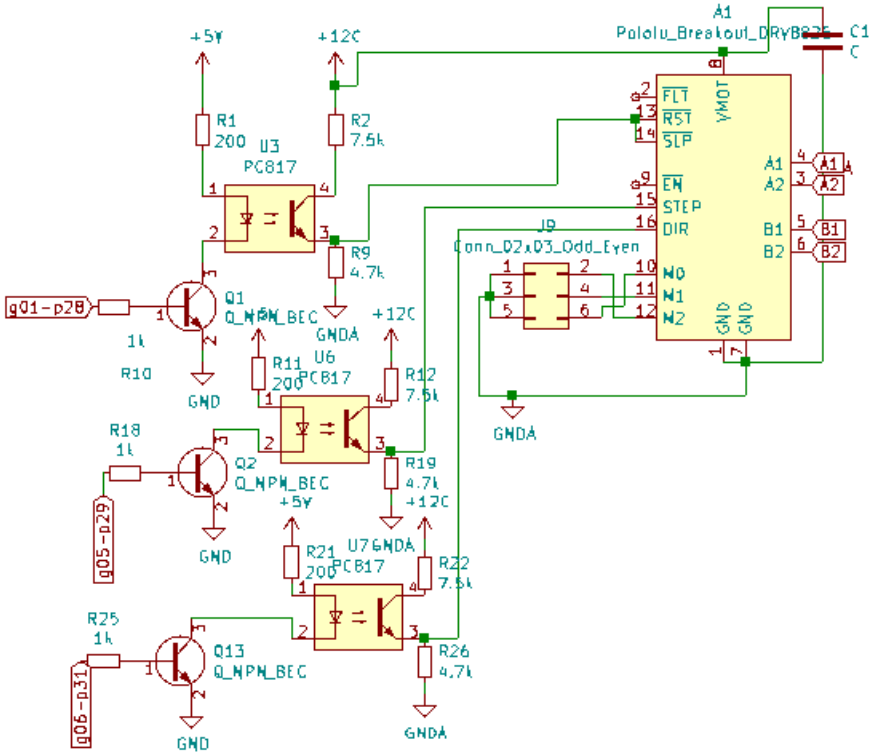


Figura 5.5: Circuito de conexión con el driver de motor DRV8825 utilizando la topología de salida digital vista anteriormente.

Finalmente en el esquemático entero se repite 6 veces el circuito de entrada digital, 4 veces el de salida con MOSFET para conectar hasta 4 ventiladores u otra carga DC, 2 veces el circuito de salida digital simple, una vez el circuito de conexión con el DRV8825 y jumpers para 6 conexiones de I^2C y 4 para SPI, el esquemático final se muestra en la Figura 5.6. En caso de que todas las salidas estén prendidas el nodo de 3.3 V consumirá una corriente de 26.9 mA y el de 5V 225 mA lo que coincide con las restricciones de corriente máxima. Respecto a posibles mejoras para este circuito esta en primer lugar hacer la buena conexión del jumper que conecta con los pines MO, M1 y M2 para operar el motor *stepper* en su modo *microstepping* y aumentar la resolución de la salida. Esto se incluyó de forma incorrecta en este circuito ya que esta pensado para ocupar *jumpers* los cuales conectan los pines a tierra y por ende no tiene efecto, por esto es necesario poner el nodo común conectado a un nivel de voltaje adecuado. Otra mejora posible es aumentar la cantidad de drivers de motores y ocupar un chip diferente (se recomienda utilizar alguno de la marca Trinamics, porque tienen muchas mas funcionalidades). Por último cabe destacar que esta placa no es un *Hardware attached on Top* (HaT) pues no cumple con las especificaciones indicadas en el sitio oficial de *Raspberry Pi* ya que carece de una memoria EEPROM que contenga información sobre la placa y no cumple con las restricciones de tamaño por lo que se propone como trabajo futuro.

Posteriormente se ocupa este esquemático para generar una placa de circuito. Originalmente se pensaba utilizar resistencias *though hole* porque son fáciles de encontrar en el mercado nacional, particularmente en la calle San Diego en santiago centro y se generó una primera versión de esta placa de circuito que se

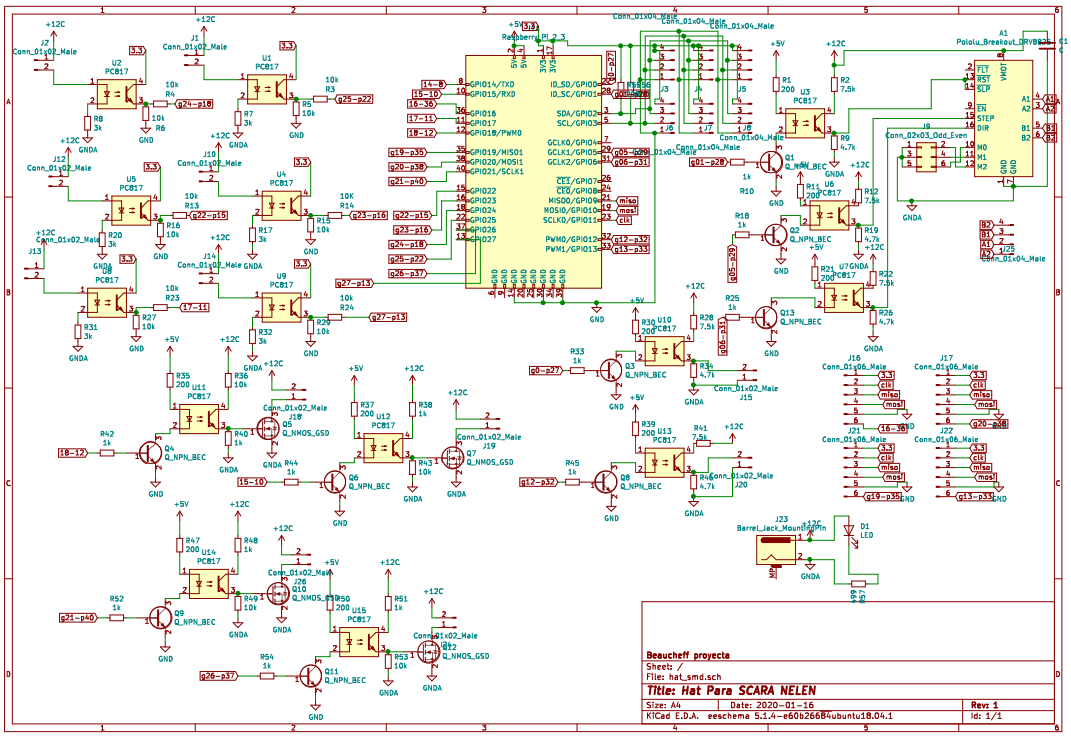


Figura 5.6: esquemático completo de la placa de extensión para GPIO

puede ver en la Figura 5.7. Este diseño es funcional, sin embargo, es demasiado grande ya que la placa total ocupa un espacio de aproximadamente 20 cm de ancho y 10 cm de largo, razón por la cual se optó por generar una nueva versión utilizando componentes de montura de superficie o *surface mount device* (SMD). Los componentes SMD son mas pequeños y están pensados para una fabricación automatizada mediante maquinas *pick and place*.

En la Figura 5.8 se muestra el nuevo diseño de la placa de circuito expansora basada en componentes SMD. Esta placa es mas pequeña tiene 9.8 cm de ancho y 9.2 cm de largo, y se redujo en tamaño a aproximadamente la mitad de su variante con componentes *through hole*. La elección de los componentes que conforman esta placa no fue al azar y se baso en los proveedores de componentes SMD en Chile, Victronics y Labtronics. Esta elección tuvo 2 limitaciones, una fue los tamaños de resistencias disponibles que eran solamente 1206 y 805, y la otra limitación fue la oferta de transistores que era muy limitada. Finalmente los escogidos y nombrados anteriormente fueron los únicos que encontraron en el mercado.

La siguiente placa que se diseñó es la llamada placa I^2C y está pensada para ser ubicada en el radio-cubito del robot. Su funcionalidad es conectar los dispositivos que se comunican por I^2C desde el pseudo-HaT a los dispositivos que están en la parte extrema del robot. Como este protocolo comparte todos sus cables es posible compartir un solo cable entre varios dispositivos. Al hacer esta placa se reduce la cantidad de cables que pasan por la manguera del robot y al tener menos cable, la capacitancia de los cables baja, lo que mejora el funcionamiento de este protocolo. Los dispositivos que se conectan por I^2C son los sensores de

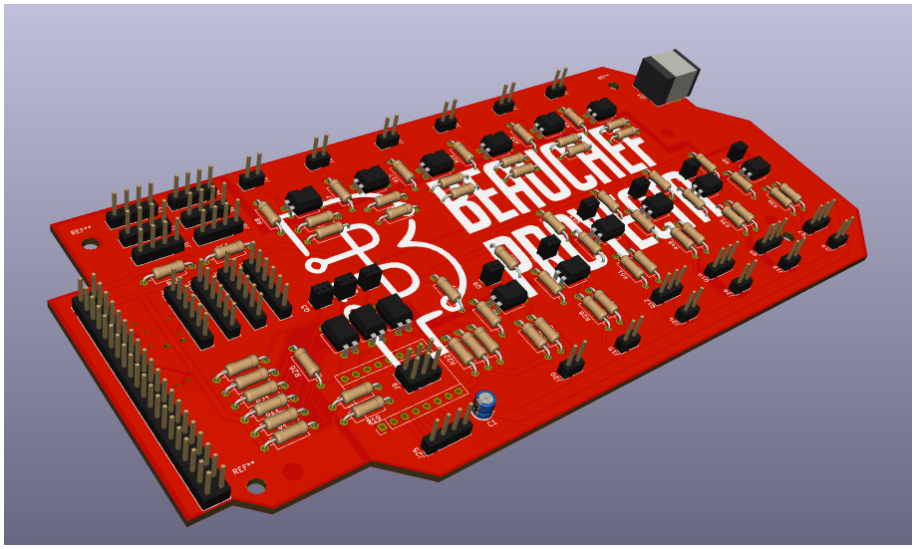


Figura 5.7: Primera versión placa expansora para *Raspberry Pi* basada en componentes *through hole*

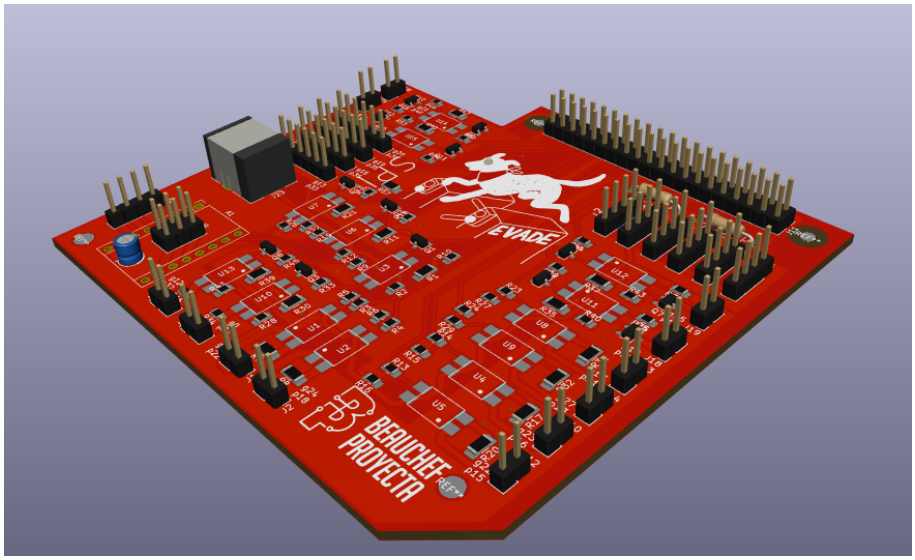


Figura 5.8: Segunda versión de la placa expansora con componentes SMD

temperatura MLX90614 y el expansor de GPIO MCP23017. Los sensores de temperatura son conectados mediante un cable que además lleva alimentación al ventilador que enfría el motor al que se le monitorea la temperatura además, esta placa tiene salidas pensadas para esta combinación de sensor con ventilador. Por su parte, el expansor de GPIO MCP2017 es utilizado para leer el estado de los limites de fin de carrera por lo que esta placa también considera las conexiones a estos. En la Figura 5.9 se muestra el esquemático de las conexiones eléctricas. El circuito se compone por un circuito integrado MCP23017 identificador U1, una serie de resistencias R1-R9, las que se usan como resistencias de *pulll-up*, excepto por r9 que es utilizada para conectar el pin de reset y evitar que el MCP23017 borre su configuración cuando se reinicia, además de 7 jumpers donde se conectan los cables. Los jumpers J1 y J3 se conectan al conjunto ventilador/sensor de temperatura de las articulaciones Z y codo respectivamente, los J2 y J4 se conectan a un solo cable que llega hasta la placa de la *Raspberry Pi* y trae todas las señales que se distribuyen en esta placa, el J5 es el

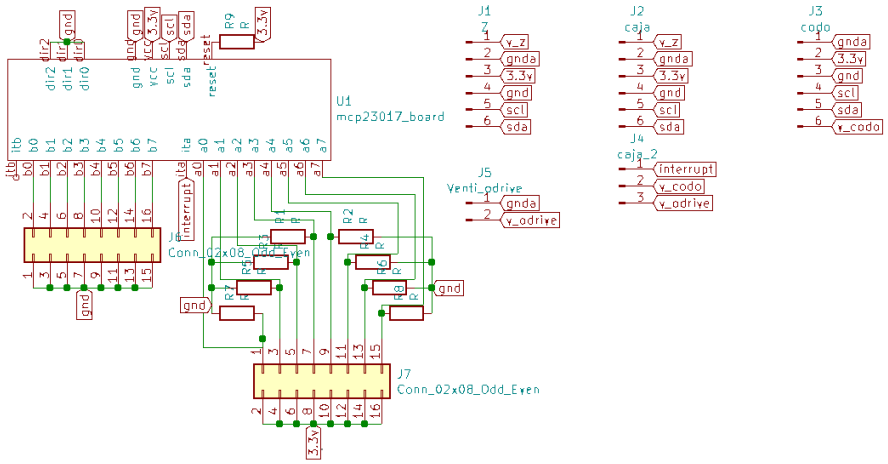


Figura 5.9: Esquemático de la placa en la que se monta el MCP23017 y se distribuyen las conexiones a los conjuntos de ventilador+sensor de temperatura y limites de carrera

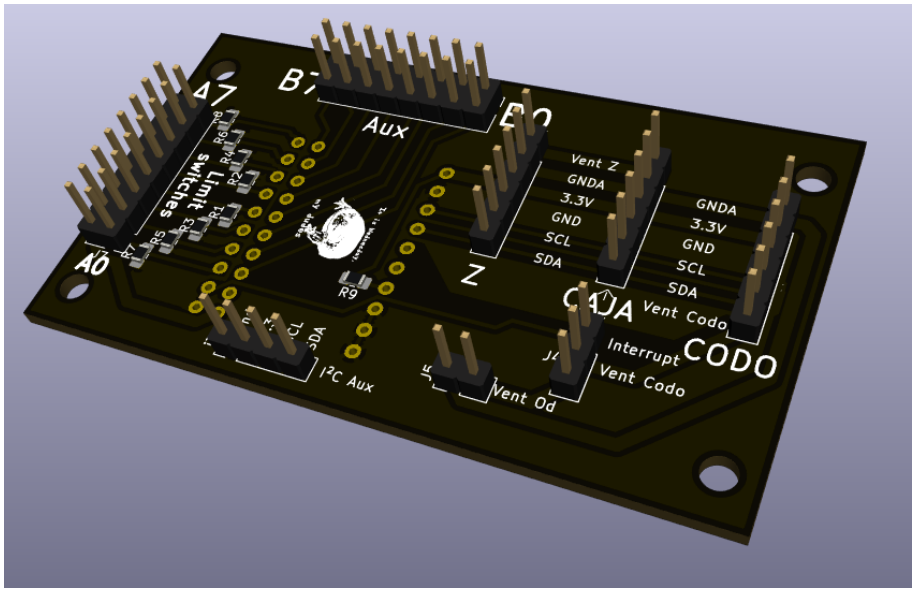


Figura 5.10: Placa de conexión de I^2C donde se incluye el expansor de GPIO MCP23017

ventilador de la placa Odrive y el J7 es el jumper donde se conectan los sensores de fin de carrera, que son interruptores que cierran el contacto entre los pines pares y su antecesor impar. El jumper J6 son las otro 8 salidas/entradas que no se están utilizando pero se incluye un conector a ellas para futuras aplicaciones además, se incluyó posteriormente un conector de 4 pines para tener una conexión I^2C extra para, por ejemplo, conectar un modulo de control de servomotores.

En la Figura 5.10 se muestra el diseño final de esta placa.

La ultima placa de circuito diseñada fue la **placa de distribución encoders**. Cuando se discutió la sección de encoders se mencionó que una de las principales características del AS5047 es que tiene 2 salidas independientes, una absoluta, la cual se conecta a la *Raspberry Pi* y una de cuadratura, la cual se conecta al *Odrive* que es bastante engorroso en el caso del radio-cubito pues el *Odrive* se ubica lejos de la *Raspberry*

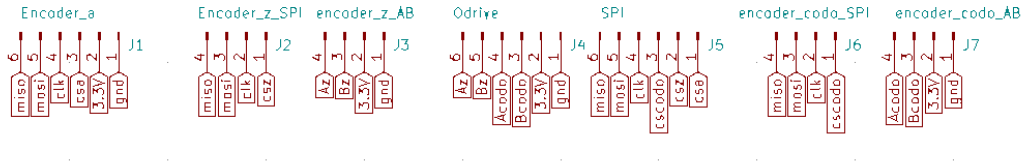


Figura 5.11: Esquemático del circuito de la placa de distribución encoders

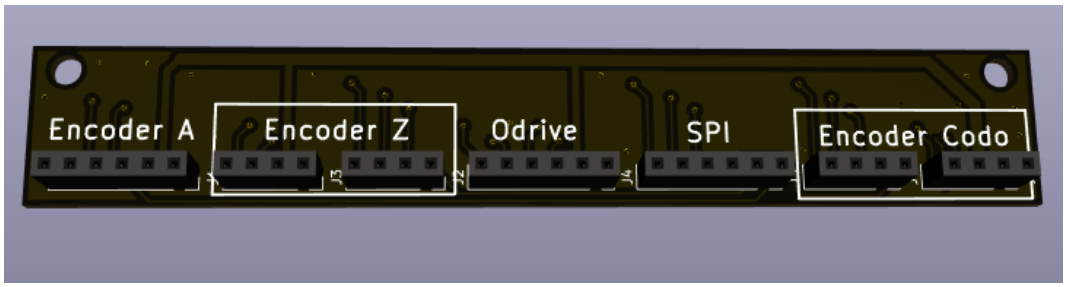


Figura 5.12: Diseño final placa de distribución encoders.

Pi. Para solucionar esto se utilizó esta placa de circuito, la cual envía un solo cable hasta la caja eléctrica con el cableado de SPI de los encoders en A, en Z y en el codo y un cable al Odrive con la salida de cuadratura de los encoders en Z y en el codo. En la Figura 5.11 se muestra el esquemático de este circuito con ningún componente mas que los terminales de los cables que llegan a el, además, se han incluido etiquetas de los nombres de las señales que conectan. 3.3V y gnd son la alimentación, miso, mosi, clk y los cs son los pines del protocolo SPI para las salidas absolutas, A y B corresponden a las salidas de cuadratura.

Se pasa este circuito a una PCB y su diseño final se muestra en la Figura 5.12.

Las 3 placas de circuito diseñadas fueron enviadas a hacer a China, se eligió como fabricante o çasa de PCB"(*PCB house*) a JLCPCB ya que era el proveedor más asequible y porque además las placas diseñadas no presentan mayores dificultades en su fabricación, ya que no tienen pistas muy delgadas, no serán utilizadas para altas frecuencias ni tampoco son de formas extravagante. Durante el proceso de diseño de la placa se revisó cuidadosamente cuales eran las capacidades del fabricante de acuerdo a lo detallado en su sitio web (jlcpcb.com/capabilities/Capabilities) y se aseguró que las placas las cumplieran para evitar que se rechazara la orden. El precio pagado por cada una fue 4 USD y además se compró un *estencil* para la placa del pseudo-HaT que buscaba facilitar su proceso de soldado. **El precio total pagado fue $69.039 CLP, $28.196 los itemes comprados y $40.843 el envío prioritario**. En la Figura 5.13 se muestran las placas de circuito fabricadas.

Finalmente, las dos últimas placas de las que se hizo mención nacieron como soluciones a problemas encontrados durante el proceso de armado y fueron fabricadas originalmente con placas preperforadas y cables de cobre además, se mantuvieron las proporciones para que fueran compatibles con las monturas que fueron diseñadas con base en los modelos de fabricación manual. En la Figura 5.14 se muestra la

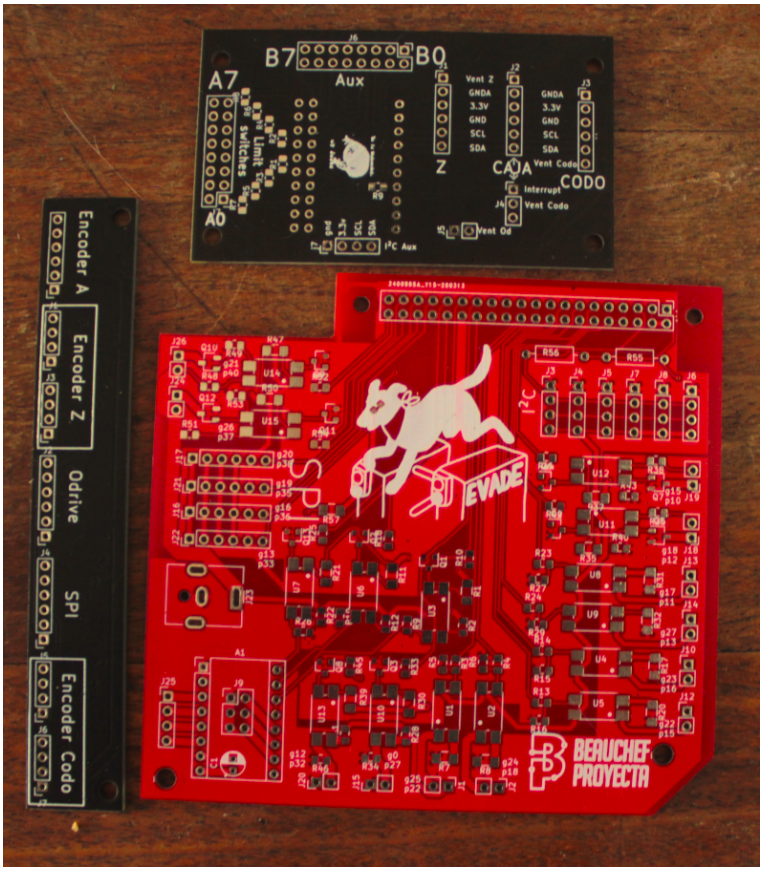


Figura 5.13: Placas fabricadas

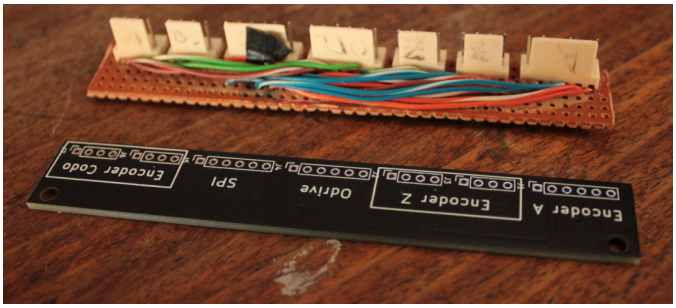
comparación entre los modelos fabricados a mano y los de fabricación industrial. La principales mejoras que se lograron fueron el orden, la confiabilidad y durabilidad pues no tienen conductores en el aire que pudieran cortarse, y además la aislación que se hace mediante planos de tierra entre las señales de la placa eléctrica.

5.6. Caja Eléctrica y protecciones

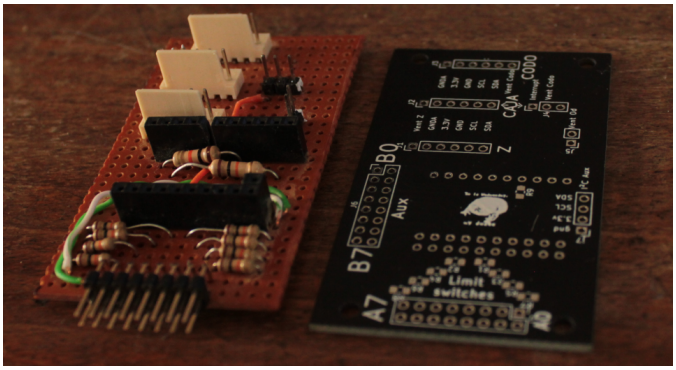
En esta sección se discutirá la elección de las protecciones elegidas para desarrollar el robot y el lugar en el que se montará todo el circuito.

Las protecciones elegidas son protecciones para circuitos monofásicos principalmente utilizadas para viviendas. Las protecciones que se consideraron fueron automáticas como protección contra cortocircuito y un diferencial como protección en caso de que alguna superficie quede accidentalmente conectada a un potencial peligro.

Para determinar el tamaño de las protecciones se consideraron las cargas a enchufar y se eligió la opción mas cercana disponible en el mercado local. En el caso de este robot, la fuente de poder elegida es cercana a 1KW, lo que equivale a una corriente cercana a los 4.5 A a un voltaje de 220 V, por lo que se elige un



(a)



(b)

Figura 5.14: Placas diseñadas y de fabricación externa junto a placas hechas a mano. En (a) la de distribución encoders y en (b) la de I^2C

automático de 6 amperes que es el valor más cercano disponible. Además, el robot cuenta con un relé de estado solido para conectar alguna herramienta en la punta, por lo que se elige un automático de 16 A el cual se conecta aguas arriba del automático de 6 amperes.

El automático elegido fue el de menor corriente de fuga disponible en el mercado, 30ma y para que su protección sea efectiva es necesario conectar todo el chasis del robot a tierra.

Es necesario considerar también una protección muy importante que es la parada de emergencia, que si bien no es una protección de circuito, es un elemento imprescindible en cualquier maquina eléctrica. Es por esto que se requiere que la parada de emergencia corte el suministro de electricidad a todos los motores además, es necesario detectar que se está en estado de parada de emergencia para que desde el computador se detengan las instrucciones de trabajo que se esten realizando y la maquina espere a salir del estado y recibir nuevas instrucciones. Para realizar esto se elige utilizar un botón de parada de emergencia volante de forma que el operario pueda activar esta parada a una distancia segura. Típicamente, la parada de emergencia se conecta en serie a las cargas que se quieren desconectar lo que tiene la desventaja de que no es posible detectar digitalmente la parada de emergencia. Por otra parte, las fuentes de voltaje tienen grandes capacitores a la salida que son utilizados para mantener un nivel de voltaje constante, estos capacitores hacen que la fuente de poder se mantenga encendida un momento posterior a su desconexión por lo que no tendría el efecto deseado de cortar el suministro de corriente de forma inmediata. Para hacer frente a estas problemáticas se ha decidido utilizar un contactor a la salida de la fuente y utilizar una de

sus salidas auxiliares como entrada digital de la *Raspberry Pi* para determinar leer el estado de la parada de emergencia. Se necesita que el contactor tenga 2 polos y un auxiliar, por ende es suficiente con un contactor monofásico sin embargo, para realizar este proyecto se decidió utilizar un contactor trifásico con el cual se contaba previamente.

Además, es necesario elegir el tamaño correcto de la caja eléctrica y para esto se consideraron los elementos que contendría y el espacio que ocuparían en ella. En la lista a continuación se detallan los elementos de la caja eléctrica:

- Protecciones: riel din con espacio para 7 protecciones de 1 terminal: 149.2 x 85mm
- Odrive : 140.5mm x 50mm
- *Raspberry Pi* + HaT :140mm x 98mm
- Relé SSR: 63mm x 46mm
- Fuente de poder 48V (puesta de lado):24 x 6.5
- Fuente de poder 12V (puesta de lado): 42mm x 130 mm
- Fuente de *Raspberry Pi*+ enchufe: 90mm x 65mm

Para tener una caja de circuito ordenada se utilizan canaletas ranuradas por las que pasarán los diferentes cables, las cuales se ordenaron en el perímetro de la placa metálica de la caja eléctrica para mantener los cables en su lugar. Al buscar en el comercio se encontró el proveedor Pailamilla e Hijos, quienes venden componentes eléctricos. En esta tienda se seleccionó el gabinete eléctrico de 500 mm x 400 mm ya que al considerar la canaleta ranurada se tiene una superficie de 370 mm x 270 mm y con estas medidas los componentes caen ordenados si se sigue la distribución de la Figura 5.15.

Cabe mencionar que no se incluyeron fusibles ni ningún dispositivo de monitoreo de corrientes lo que supone una posible mejora para una próxima versión.

5.7. Cables

La elección de los cables a utilizar en el robot no es trivial. Por una parte, es necesario que tengan el grosor suficiente para soportar la corriente que deben transportar sin quemar ni derretir la aislación que los recubre y por otra parte algunos de estos son utilizados para transportar información y por lo tanto es necesario que estén protegidos de las interferencias del entorno. Por último, puede ser que exista un conductor ideal para la aplicación en el robot pero se debe considerar el precio y la disponibilidad en el mercado nacional o si tiene sentido importarlo, tanto desde el punto de vista económico como temporal.

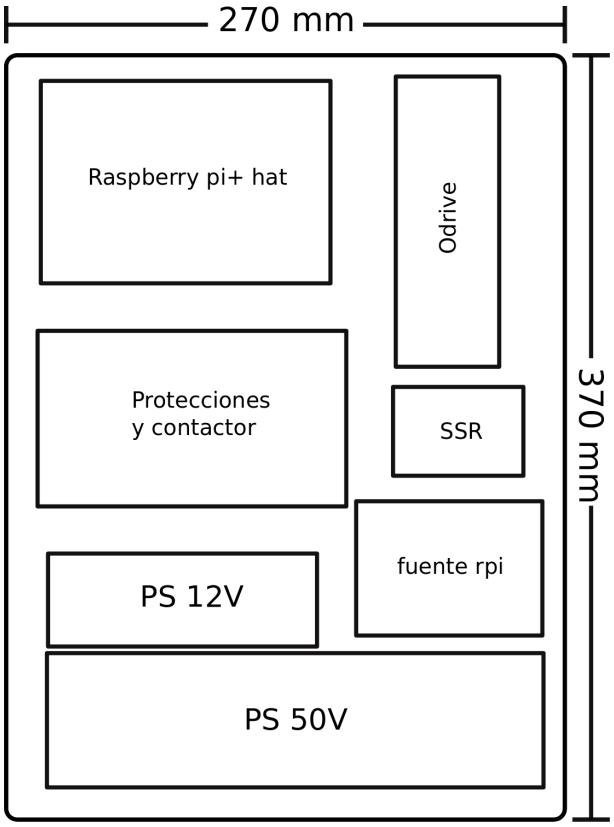


Figura 5.15: Diagrama de posición de componentes en caja eléctrica.

En cuanto a los cables, se consideran 2 aspectos principalmente, su grosor o calibre, el cual se mide en AWG (*American Wire Gauge*) y a mayor (AWG) menor diámetro. La segunda característica es el material del que está hecho su aislación lo que determina la temperatura máxima a la que opera el conductor, esta se identifica mediante un código y dentro de este código, "H"significa *heat resistant* (resistente al calor), soporta temperaturas hasta 75° y "HH"significa *high heat resistant* (resistente a alto calor), soporta hasta 90°, esta información se obtiene de [46] donde existe una guía práctica detallada para la identificación de las etiquetas de los cables.

Como se discutió anteriormente, los motores serán utilizados en su punto de máximo torque lo que implica corrientes cercanas a los 90 amperes que supone una condición de operación exigente para un cable.

Se debe tener en cuenta la máxima corriente que debe soportar el conductor y encontrar el mínimo diámetro que soporta dicha corriente. Esta máxima corriente respecto al diámetro se conoce como *ampacidad* y varía en gran medida según la norma que se utiliza. Se consultaron 3 fuentes pensadas para instalaciones domesticas, estas fueron [47], [48] y [49], siendo esta última la norma oficial. Estas 3 normas no coinciden entre si pero tienen valores de ampacidad similares, para compararlas se revisó la corriente recomendada para una sección transversal similar y arbitrariamente se elige $700[circularmils]$ o $0.35[mm^2]$. En [47] se recomienda una corriente máxima de 1 ampere para transmisión de potencia y 2.51 amperes para la denominada çorriente de chasis", que se refiere a cables que están expuestos al aire o que recorren cortas

distancia y además aclaran que ambas ampacidades son bastante conservativas. En [48] se recomienda una corriente máxima de 3.66 Amperes. Por último, en la norma de la *National Electric Code* se recomienda una corriente máxima de 3.22 Amperes. Estos sitios son buenos para entender de mejor manera que es lo que limita la corriente máxima, ya que en palabras simples un cable es una pequeña resistencia y mientras mas largo y delgado, mayor es su resistencia, ya que cuando circula corriente se produce una caída de tensión en el cable y se transforma en calor, y si suficiente corriente pasa por el cable y no se ventila este se calienta hasta derretir su aislación. En [50] se muestra un resumen de corrientes máximas para cables de silicona (resistentes a la temperatura) y conectores pensado específicamente para el uso en drones y vehículos radiocontrolados. Los limites de ampacidad son mucho mas altos con 10.2 amperes por cada 700[$circular mils$]. Dada la alta diferencia entre las referencias encontradas se decide tomar la última de estas, puesto que se basa en experiencias de aplicaciones con motores similares a los que utiliza este robot, sin embargo, se evaluaron los cables elegidos según el enfoque propuesto en [51], donde se calcula el tiempo que le toma al conductor subir a una temperatura limite desde la temperatura ambiente para una sección transversal y corriente impuesta.

Respecto a los cables que transportan señales, como por ejemplo las del encoder, se decide usar algún tipo de cable apantallado. Un cable apantallado tiene un recubrimiento de un material conductor que aísla del ruido a los cables que transporta. Para que su uso sea efectivo es necesario conectar la pantalla a tierra en solo uno de sus extremos, donde tierra es el nivel neutro de voltaje es decir, el nivel de potencial que se considera 0 para la señal transportada. Al buscar en el mercado nacional se encontró que la alternativa más asequible era comprar cable *Shielded Twisted Pair* par trenzado apantallado (STP). Este cable consiste en 4 pares de cable trenzado de un solo conductor cada uno con una pantalla externa y su uso principal es para armar cables de internet (Ethernet) de larga distancia. Este cable se compró a un proveedor llamado Sercompra a un precio de 980 pesos el metro. La única otra alternativa disponible fue en Vitel, donde se encuentra un cable apantallado entre 2 y 15 conductores y con precios que oscilan entre los 985 a 12895 pesos. Finalmente, se eligió el cable STP y se adapto a las necesidades que se presentaron durante el armado.

Por último, para las necesidades de potencia de las placas *Odrive* y para los motores se eligió utilizar cable SuperFlex de 10 AWG, que soportara una corriente máxima de mas de 100 A según [50] y demoraría 47.19 segundos en alcanzar 90 grados según [51]. El brazo robotico ocupará mucho menos tiempo en recorrer su rango de trabajo completo por lo que no se alcanzará esta temperatura en una condición normal de operación del robot. Para el cableado de señales se utilizó cable STP, para el cableado interno de la caja eléctrica se ocupó alambre de cobre para 16A y para la parada de emergencia se ocupó cable de linea musical, que tiene 2 conductores, soporta voltajes de hasta 300V y tiene también pantalla.

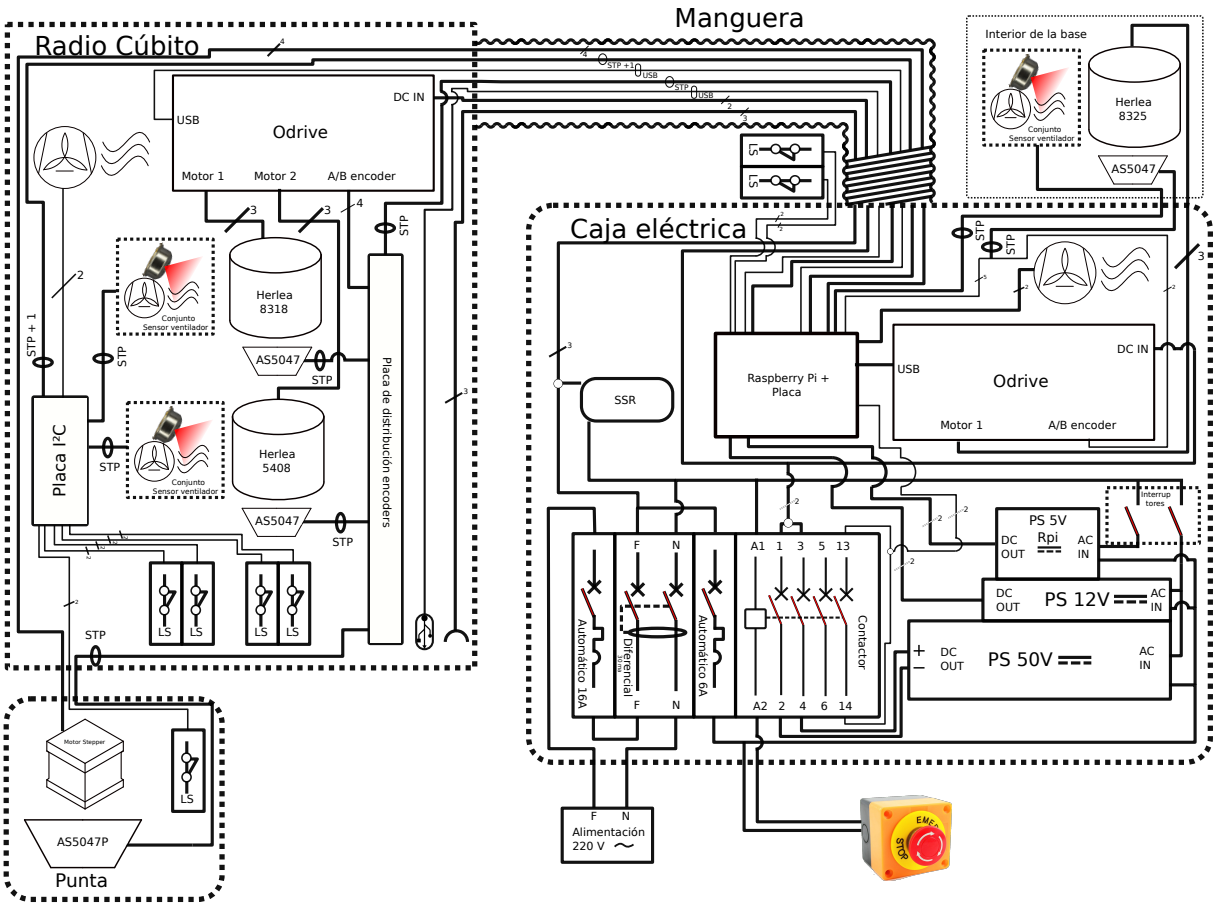


Figura 5.16: Circuito robot SCARA

5.8. Diagrama de conexión

Rescatando todas las apreciaciones, diseños y elecciones antes realizadas, en la Figura 5.16 se muestra el circuito completo del robot, salvo los detalles de las placas eléctricas. En la Figura las lineas negras solidas representan cables, las que poseen una raya y un número es porque llevan ese numero de cables, por ejemplo todos los motores son conectados mediante 3 cables, los que tienen otra simbología y una palabra son del tipo que indica la palabra, como STP o USB. Las lineas punteadas son subconjuntos donde hay circuitería, nótese que la cantidad de cables que circulan entre las placas diseñadas y el *Odrive* en el radio cubito es mayor que la cantidad de cables que pasan por la manguera precisamente para lograr reducir la cantidad de cables a utilizar se cambia de lugar el *Odrive* desde la caja eléctrica hasta el radio cúbito. Por último, los componentes se catalogados por su código o nombre encima, por su simbología eléctrica, o por un dibujo.

Capítulo 6

Programación

En este capitulo se aborda la programación que debe tener el SCARA. El programa debe cumplir con los objetivos que se muestran en la siguiente lista. El lenguaje de programación escogido fue Python ya que permite implementar todos los protocolos de comunicación necesarios de forma sencilla y porque además, las placas *Odrive* tienen una clase implementada en este lenguaje que permite acceder a todas sus funcionalidades. Python es un lenguaje de programación sencillo e ideal para lograr soluciones completas en poco tiempo, pero no se debe descartar como una posibilidad de mejora futura la utilización de otro lenguaje de programación de mas bajo nivel como C++ que permitiría tener un programa mas liviano y tal vez aumentar la velocidad.

- Comunicar todos los componentes electricos.
- Leer constantemente la posición del robot.
- Detectar cuando se ha tocado un fin de carrera o la parada de emergencia y detener la operación inmediatamente.
- Mantener la temperatura de los motores en un nivel de funcionamiento seguro.
- Hacer una rutina de calibración.
- Mover el brazo a una posición determinada

Antes que empezar a programar es necesario cargar un sistema operativo en la *Raspberry Pi* y para esta tarea se ha elegido utilizar Ubuntu Mate, pues se puede integrar fácilmente ROS Melodic en el. El objetivo a largo pĺazo de este brazo es poder utilizar ROS para operarlo, razón por la cual, se ha priorizado la compatibilidad con este sistema al elegir un sistema operativo. Se siguió el procedimiento indicado en [52]. Por último es necesario activar los protocolos I^2C y SPI en la *Raspberry Pi*, para esto se abre una consola y se corre la linea de comando "sudo raspi-config.ªhí se accede a un menú de configuración de hardware, se debe entrar en Ïnterfacing Optionsz activar ambos protocolos. Ahora la *Raspberry Pi* se encuentra lista ppara correr el programa principal.

Tabla 6.1: Direcciones dispositivos I^2C

Dispositivo	Dirección
Expansor GPIO	0x20
Sensor T hombro	0x5A
Sensor T codo	0x5B
Sensor T Z	0x5F

Se ha decidido crear mas que un programa, una clase `nelen`, en la cual se guarden todas las variables importantes del robot, y que las operaciones del robot sean funciones de esta misma clase, así será mas fácil hacer rutinas mas complejas de funcionamiento. Este programa y todos los usados para las simulaciones se encuentran en el repositorio de este proyecto en github SCARA NELEN.

6.1. Dispositivos I2C

Antes de poder comunicarse con los distintos dispositivos es necesario hacer ciertos preparativos. En primer lugar se deben configurar los dispositivos I^2C. En este protocolo cada dispositivo tiene asociada una dirección única, por lo que es necesario cambiar algunas de estas, en particular los sensores de temperatura vienen con la misma dirección por lo que al menos 3 de ellos deben cambiarse. En la hoja de datos [40] de estos dispositivos se muestra el procedimiento para realizar esto, consiste en utilizar la dirección universal 0x00 y luego modificar el valor del registro interno en la dirección 0x2E del chip escribiendo la nueva dirección deseada. Existe un código para Arduino para realizar esto, el cual se puede encontrar en [53]. Este procedimiento Respecto al expansor de GPIO este tiene 3 pines los cuales se pueden conectar al voltaje de alimentación o tierra y así elegir una de las 8 direcciones posibles, se utiliza el valor por defecto. En la Tabla 6.1 se muestran las direcciones elegidas para los dispositivos que se conectan al bus I^2C.

6.2. Configuración Odrive

Para poder comunicarse con el *Odrive* existen varias opciones, en esta ocasión se prefirió utilizar USB por su alta velocidad y fácil integración con *Raspberry Pi* mediante Python. Se sigue el procedimiento de instalación oficial para Ubuntu en la pagina *docs.odriverobotics.com*. Para comunicarse con este se abre una consola de comando y se corre la linea `odrivetool`. Al hacer esto se entra en una ventana de Python y empieza una rutina de detección en la que se detectan todos los dispositivos conectados. Al detectar uno se crea un objeto de la clase `odrive`, la cual cuenta con 2 ejes axis0 y axis1, cada uno cuenta con 4 atributos principales, `config`, `motor`, `encoder` y `controller`.

Respecto a su funcionamiento, el *Odrive* funciona como una maquina de estados, de esta forma se asegura que solamente se energicen los motores cuando todos los parámetros de la configuración a utilizar están correctos. Para cambiar de un estado a otro es necesario cambiar el atributo <axis>.requested_state, con alguno de las siguientes opciones:

- AXIS_STATE_IDLE Disable: Motores apagados
- AXIS_STATE_FULL_CALIBRATION_SEQUENCE: Rutina de calibración completa (calibración del motor + calibración encoder)
- AXIS_STATE_MOTOR_CALIBRATION: se mide la resistencia y la inductancia del motor, estos parametros se guardan en <axis>.motor.config.phase_inductance, además cambia el atributo <axis>.motor.is_calibrated a verdadero.
- AXIS_STATE_ENCODER_OFFSET_CALIBRATION: calibración del desfase que existe entre el encoder y la secuencia de energización del motor. Se hace girar el motor brevemente y se mide este valor en cantidad de cuentas del encoder y se cambia el atributo <axis>.encoder.is_ready.
- AXIS_STATE_CLOSED_LOOP_CONTROL: modo de control de lazo cerrado, en este modo el motor, solamente se accede a estado si <axis>.motor.is_calibrated y <axis>.encoder.is_ready son verdaderas.

Para el correcto funcionamiento de las placas de circuito *Odrive* es necesario cargar parámetros de los motores y encoders.Es necesario cargar 2 parámetros de configuración por motor, estos son, la cantidad de polos del motor y los pulsos por vuelta de los encoders. Los motores Herles 8325 y 8318 tienen 20 pares de polos y el 5408 tiene 14 pares de polos. Todos comparten el mismo encoder el que tiene 4000 pulsos por vuelta. Estos parámetros se modifican directamente en el objeto `Odrv0`.

Para operar el Odrive en su modo de lazo cerrado es necesario correr una rutina de calibración completa antes de pasar al modo de control por lazo cerrado. Esto no es posible realizar con el robot armado, pues para hacerla se requiere que el motor pueda girar sin carga, en este caso se tienen los motores conectados mecánicamente a las cargas que mueven, por lo que no se podría obtener una correcta calibración del motor. Para solucionar este problema, es necesario realizar el siguiente procedimiento

- Realizar los cableados finales, tanto de los encoders como de los motores.
- Para cada uno de los motores correr una secuencia de calibración completa, esto es, dentro de *Odrivetool* correr el comando <axis>.requested_state=AXIS_STATE_FULL_CALIBRATION_SEQUENCE.
- Guardar los valores de resistencia e inductancia, para hacer esto correr la siguiente linea <axis>.motor.config.pre_calibrated = True.
- guardar configuración con odrv0.save_configuration().

6.3. Lectura de Encoders

Los encoders son los sensores mas importantes que se tienen en el robot, pues entregan información sobre el estado actual. La lectura de estos se debe hacer de forma periódica e ininterrumpida en un proceso que

guarde una prioridad superior a la lectura de los sensores de temperatura por ejemplo. Por esta razon en esta sección se describe el desarrollo de la librería para la lectura del sensor y la rutina de lectura.

Para leer el sensor se escribe una pequeña librería que funciona con python llamada `encoder`. Tiene 2 atributos internos, `cs`, que es el numero del puerto GPIO conectado al pin CS del encoder a leer; y `spi` que es un objeto de la clase spidev, la cual se usa para leer el puerto spi. Ademas esta librería tiene 2 funciones

- `leer(self)`: función que se encarga de realizar una lectura del registro 0xFF el cual es la posición absoluta del encoder. El encoder responde con 2 *bytes* o 16 bits, de los cuales el mas significativo corresponde al bit de paridad, para chequear una correcta comunicación, el segundo corresponde a una alerta de error, `leer` chequea que ambas condiciones estén en orden. Los últimos 14 bits corresponden a la información de la posición, la cual es retornada si se cumple con el check de paridad y que no este levantada la alerta del segundo bit.
- `abs2offset(self,dato)`: para poder entrar en el estado de control de lazo cerrado del *Odrive* es necesario que el encoder esté calibrado. Lo que se busca en la práctica es obtener el desfase que existe entre la posición del motor y la secuencia de energización. Como el encoder envía al *Odrive* una señal incremental, este no tiene idea de donde se encuentra al energizarse, solo puede contar los pulsos desde que se enciende. Esta función utiliza una tupla llamada `dato` que tiene un desfase conocido y extrapola cual debería ser el desfase según la posición absoluta que se lee. Es muy útil para saltarse la rutina de calibración.

Dentro de la clase `nelen` se crea una función que está encargada de hacer la lectura, esta se muestra al final de esta sección. El proceedimiento es bastante simple, previemente en la inicialización de la clase se han generado objetos de la clase encoder, estos son los que su nombre empieza con `self.enc_`. Cada vez que se llama la función esta evalúa si esta leyendo, mediante la condición interna `self.reading`, la cual es una variable booleana, luego se revisa cual es la diferencia de tiempo entre la

última muestra y el tiempo en el que se llamo a la función, si esta es mas que el periodo de muestreo entonces realiza una lectura de cada encoder y actualiza los valores de los atributos de la clase. Finalmente este programa se llama a si mismo al final, por lo que es un programa recursivo. Claramente, una vez que se corre este programa, Python se queda atrapado pues nunca termina el proceso, siempre se llama de nuevo, este programa fué escrito de esta forma para aprovechar los multiples procesadores de la *Raspberry Pi* utilizando la librería multiprocessing. Para esto, al final de la inicialización se incluye la linea de codigo `encoder_process = multiprocessing.Process(target=self.sample)`. Cabe mencionar que durnte el tiempo que no se esta realizando lecturas, el procesador al que se le asigna este proceso está constantemente ocupado evaluando si debe leer o no, a esta espera se la conoce como *busy waiting* o espera ocupada. Es mucho menos eficiente pero asegura máxima precisión en el tiempo en el que se ejecuta la tarea, lo que asegura una tasa de muestreo constante.

```
1 def sample(self):
```

```
        t=time.time() #leer el tiempo
        if self.reading:
            if t-self.last_time>self.Ts and self.reading:
                self.hombro_enc_val = self.enc_hombro.leer()
                self.codo_enc_val = self.enc_codo.leer()
                self.z_enc_val = self.enc_z.leer()
                self.a_enc_val = self.enc_a.leer()
                self.last_time=t
            self.sample()
```

6.4. Sensores fin de carrera y parada de emergencia

Los sensores de fin de carrera y la parada de emergencia son interruptores que se conectan a las entradas digitales del sistema. Un interruptor se puede conector en *normally open*, que está siempre abierto y cierra el contacto al activarse el interruptor; o *normally closed*, que está siempre cerrado y abre el circuito al activarse. Se elige que todos los interruptores sean *normally closed* porque si existe algún problema de cable defectuoso la entrada digital asociada al sensor sea 1 y detenga el funcionamiento de la maquina. Ademas, es necesario que al activarse uno de estos sensores se interrumpa la tarea actual del programa y se frene el robot.

Para saber el estado de un sensor de fin de carrera se debe leer la entrada digital asociada al GPIO, esto se hace mediante la librería `RPi.GPIO`, esta permite utilizar todo el bus de GPIO de 40 pines de la *Raspberry Pi* como entradas o salidas. En este caso, no interesa hacer un muestreo periódico del estado de los sensores, sino que interrumpir el programa cuando esto suceda, para esto se utiliza el modo `interrupt`. Se debe definir una función de *callback* que se corre cuando se detecta una activación de sensor, se entrará en detalle de esta función mas adelante. Luego se asocia la siguiente linea en la inicialización de la clase `nelen`, una por cada sensor de fin de carrera. Las opciones que se entregan son `LS_PIN` que es el numero de GPIO asociado al sensor, `GPIO.FALLING` que indica que se detectan cambios de nivel lógico 1 a 0 que es el caso de los sensores de fin de carrera conectados en *normally closed*, `bouncetime` que es un parámetro que se ajusta para que la función sea llamada solo una vez y `callback`, que es la función que se quiere correr al detectarse la activación del sensor.

```
GPIO.add_event_detect(LS_PIN, GPIO.FALLING, callback=limit_switch_callback, bouncetime
    =100)
```

Los sensores de fin de carrera que se encuentran en el radio-cúbito no están conectados directamente sino a través del expansor de GPIO MCP23017. Se ocupa la hoja de datos [54] para determinar los registros internos que cambiar para setear sus puertos como entradas. Es importante notar que este dispositivo cuenta con 2 configuraciones las que cambian completamente como están mapeados los registros, este valor se llama `IOCON.BANK`, en el caso del comprado e implementado `IOCON.BANK=0`. Asi, se configuran los

Tabla 6.2: Configuración MCP23017

Registro	Dirección	Valor	Función
IODIRA	0x00	0xff	Configurar puertos A como entradas.
GPIENTA	0x04	0xff	Especificar que se ocupa el pin interrupt.
INTCON	0x08	0xff	Se compara respecto a un valor determinado.
DEFVALA	0x06	0xff	Valor que deben tener las entradas.

registros correspondientes para que los 8 primeros puertos sean entradas. Además el MCP23017 cuenta con 2 salidas digitales extra para hacer uso del *interrupt*. Se configuran los registros según la hoja de datos para que se active el interrupt cuando las entradas A sean 0. En la Tabla 6.2 se muestran los registros que se configuraron y el valor que se les asoció.

Por último se debe especificar las funciones de *callback*, esto es el procedimiento que se llevará a cabo al detectar un sensor de fin de carrera. En el caso del *interrupt* asociado al MCP23017 esta función es algo distinta pues se debe leer cual de los sensores fué el que gatillo la interrupción, salvo eso ambas funciones son iguales. El comportamiento deseado es que al activarse, todo el brazo vuelva a la posición inicial y se detengan los trabajo que se estaban realizando.

Por último, también se ha incluido una entrada digital para detectar la parada de emergencia. Cuando sucede esto se corta el suministro de energía de los *Odrive* y borra en este la calibración y cuenta de los encoders. La función de *callback* asociado a este evento reinicializa todas las variables de calibración del programa, espera a que se desactive la parada de emergencia y busca los *Odrive* nuevamente.

6.5. Monitoreo de temperatura

Como el desempeño y vida util de los motores depende de su temperatura es necesario monitorear y controlar esta. Para esta tarea se integraron ventiladores que aumentan el flujo de aire del motor y sensores infrarojos de temepratura MLX90614. Para la comunicación con el sensor se utiliza el protocolo I^2C y la librería de Adafruit disponible en [55].

La temperatura se lee constantemente de forma similar a los encoders, sin embargo con una frecuencia mucho menor dado que estos procedimientos son lentos, para realizar esto se ocupa el modulo `threading` de Python que permite incluir tareas y liberar el procesador cuando estas no esten siendo ejecutadas. En primer lugar se define una función de lectura la que se muestra a continuación. Esta también es recursiva pero incluye un tiempo muerto `Ts_temp`. Al final de la inicialización de la clase nelen se incluye esta función a un *thread* con la linea *temp_thread = threading.Thread(target=self.read_temp)*. Luego de cada lectura se evaluan 3 limites de temperatura, con diferentes acciones. Si se está bajo el primer límite `lim1` entonces se apaga el ventilador. Si se pasa el primer límite `lim1` entonces se prende el ventilador. Si se pasa el segundo *lim2* límite se baja la corriente máxima del motor en cuestión.

```
1 def read_temp(self):
2       if self.reading_temp:
```

```
            self.temp_hombro = self.mlx_hombro.object_temperature
            if self.temp_hombro<self.temp_lim1:
                #apagar ventilador
                GPIO.output(self.vent_hombro,0)
            elif self.temp_hombro<self.temp_lim2:
                #prender ventilador
                GPIO.output(self.vent_hombro,1)
            elif self.temp_hombro<self.temp_lim3:
                #bajar corriente maxima
                self.odrv_hombro.axis0.motor.config.current_lim=self.low_current
            else:
                self.apagar()
            self.temp_codo = self.mlx_codo.object_temperature
            if self.temp_codo<self.temp_lim1:
                #apagar ventilador
                GPIO.output(self.vent_codo,0)
            elif self.temp_codo<self.temp_lim2:
                #prender ventilador
                GPIO.output(self.vent_codo,1)
            elif self.temp_codo<self.temp_lim3:
                #bajar corriente maxima
                self.odrv_rc.axis0.motor.config.current_lim=self.low_current
            else:
                self.apagar()
            self.temp_z = self.mlx_z.object_temperature
            if self.temp_z<self.temp_lim1:
                #apagar ventilador
                GPIO.output(self.vent_z,0)
            elif self.temp_z<self.temp_lim2:
                #prender ventilador
                GPIO.output(self.vent_z,1)
            elif self.temp_z<self.temp_lim3:
                #bajar corriente maxima
                self.odrv_rc.axis1.motor.config.current_lim =self.low_current_z
            else:
                self.apagar()

            time.sleep(Ts_temp)
            self.read_temp()
```

6.6. Operación del brazo

El funcionamiento del SCARA es similar al del *Odrive* en el sentido de que es una maquina de estados, esto se hace así para asegurar que el robot esté bien calibrado antes de empezar a moverse.

La rutina de calibración es una función de la clase nelen llamada `calibration_abs()` y lo que hace es llamar a la función `abs2offset` para cada motor, luego actualiza el desfase calculado en el *Odrive* correspondiente. Esta función se ejecuta en mucho menos tiempo que ejecutar la rutina de calibración propia del *Odrive*. De todas formas existe una función llamada `calibration_old()` que realiza la rutina de calibración para

cada eje desde el *Odrive* y actualiza los datos de desfase que utiliza la función `calibration_abs()`. Por último se cambia la condición de la clase nelen de la calibración

```
def calibration_abs(self):
       self.odrv_hombro.axis0.encoder.config.offset=self.enc_hombro.leer(self.
           dato_hombro)
       self.odrv_hombro.axis0.encoder.is_calibrated=True
       self.odrv_rc.axis0.encoder.config.offset=self.enc_codo.leer(self.dato_codo)
       self.odrv_rc.axis0.encoder.is_calibrated=True
       self.odrv_rc.axis1.encoder.config.offset=self.enc_z.leer(self.dato_z)
       self.odrv_rc.axis1.encoder.is_calibrated=True
       self.is_calibrated=True
```

Luego es necesario dejar el robot en una condición conocida, esta rutina se llama *homing*. Se evalúa primero que se haya hecho la calibración y luego se cambia el estado de todos los motores a control por lazo cerrado. Posteriormente se cambia el modo de control a control de velocidad y se fija una velocidad lenta. Eventualmente, se tocará algún sensor de fin de carrera, lo que llama a la función de *callback* respectiva, esto fija la velocidad a 0 por lo que el movimiento se detiene, posteriormente se mueve el motor una distancia conocida que es la distancia desde el sensor de fin de carrera a la posición inicial. Este proceso se realiza para cada motor y al final se actualiza la condición de la clase nelen `self.is_homed`

```
if self.is_calibrated:
           #Control por lazo cerrado en los odrive
           self.odrv_hombro.axis0.requested_state=AXIS_STATE_CLOSED_LOOP_CONTROL
           self.odrv_rc.axis0.requested_state=AXIS_STATE_CLOSED_LOOP_CONTROL
           self.odrv_rc.axis1.requested_state=AXIS_STATE_CLOSED_LOOP_CONTROL
           #homing motor hombro
           self.odrv_hombro.axis0.controller.control_mode=CTRL_MODE_VELOCITY_CONTROL
           self.odrv_hombro.axis0.controller.vel_setpoint=self.homming_speed
           while True:
               #esperar que se toque el ls
               if self.touched_hombro:
                   break
           #cambiar modo de control
           self.odrv_hombro.axis0.controller.control_mode=CTRL_MODE_POSITION_CONTROL
           #Resetear cuenta de encoder
           self.odrv_hombro.axis0.encoder.set_linear_count(self.pos_ls_hombro)
           #mover a posicion conocida
           self.odrv_hombro.axis0.controller.move_to_pos(0)
           #homing motor codo
           self.odrv_rc.axis0.controller.control_mode=CTRL_MODE_VELOCITY_CONTROL
           self.odrv_rc.axis0.controller.vel_setpoint=self.homming_speed
           while True:
               #esperar que se toque el ls
               if self.touched_codo:
                   break
           #cambiar modo de control
           self.odrv_rc.axis0.controller.control_mode=CTRL_MODE_POSITION_CONTROL
           #Resetear cuenta de encoder
           self.odrv_rc.axis0.encoder.set_linear_count(self.pos_ls_codo)
           #mover a posicion conocida
           self.odrv_rc.axis0.controller.move_to_pos(0)
```

```
32
33          #homing motor z
34          self.odrv_rc.axis0.controller.control_mode=CTRL_MODE_VELOCITY_CONTROL
35          self.odrv_rc.axis0.controller.vel_setpoint=self.homming_speed
36          while True:
37              #esperar que se toque el ls
38              if self.touched_z:
39                  break
40          #cambiar modo de control
41          self.odrv_rc.axis0.controller.control_mode=CTRL_MODE_POSITION_CONTROL
42          #Resetear cuenta de encoder
43          self.odrv_rc.axis0.encoder.set_linear_count(self.pos_ls_z)
44          #mover a posicion conocida
45          self.odrv_rc.axis0.controller.move_to_pos(0)
46          self.is_homed=True
```

Por último, se crea una función que permite mover al robot a cualquier punto de su área de trabajo, la cual se llama `move_to_XYZA`. Esta función tiene como entrada la posición deseada en metros. En su funcionamiento evalúa si se cumplen las condiciones `is_homed` e `is_calibrated`. Luego evalúa que la entrada esté dentro del área de trabajo y si esto se cumple se calcula cual tiene que ser la posición de los motores para llegar a dicho punto. Esta funcion se realiza con un objeto llamado `scara_ik` de la clase `tinyIk` que se usa para calcular la cinemática inversa, corre de forma optimizada las ecuaciones que se muestran en 2.3 y 2.4. Luego se envía esta información al *Odrive* correspondiente.

```
1       #evaluar que se haya calibrado
2       if is_calibrated and is_homed:
3           #confirmar que el punto es alcanzable
4           if self.is_in_workspace(move_here):
5               #calcular posiciones en las articulaciones
6               #pasarlas a cuentas para el Odrive
7               self.scara_ik.ee=(move_here[0:2])
8               pos_hombro=4000.*7*self.scara_ik.angles[0]
9               pos_codo=4000.*7*self.scara_ik.angles[1]
10              pos_z=move_here[3]/16.*1000.*4000.
11              #enviar instrucciones
12              self.odrv_hombro.axis0.controller.move_to_pos(pos_hombro)
13              self.odrv_rc.axis0.controller.move_to_pos(pos_codo)
14              self.odrv_rc.axis1.controller.move_to_pos(pos_z)
15              self.move_a(move_here[3])
```

6.7. comentarios finales y trabajo futuro

La clase nelen cumple con los objetivos mostrados al inicio de este capitulo, esto es la implementación básica del robot SCARA. Mediante esta clase se puede mover el robot cuidando no sobrepasar los limites físicos ni de temperatura. Además es fácilmente extensible para desarrollar otras tareas o comunicarse con otros dispositivos, para esto solo hay que crear nuevas funciones que los operen.

El programa entero no se ha probado, pues el SCARA aún está en proceso de construcción. Sin embargo, cada una de sus partes si ha sido probada por separada.

La principal funcionalidad que falta incluir en esta clase es la creación de un nodo de ROS, dicho nodo debe publicar la información que se lee de los encoders y suscribirse a un tópico de ordenes, donde se publiquen posiciones deseadas desde ROS. Luego el nodo debe usar estos mensajes para llamar a la función `move_to_XYZA`. Esto no se hizo por tiempo y también porque es necesario hacer un traductor desde Python3, que es el lenguaje en el que está escrito la clase nelen a Python2 que es el lenguaje que se ocupa en ROS. Otro enfoque para realizar esto es habilitar comunicación por ethernet, de esta forma se puede utilizar ROS en otro computador en la misma red local y solo generar un nodo que se encargue de la comunicación.

Actualmente el programa funciona con *scripts* de Python por lo que otro gran avance propuesto es la inclusión de una gui, ya sea por control remoto o una app de celular, de forma de poder operar el robot sin estar conectado físicamente a el.

Capítulo 7

Pruebas y resultados

En la presente sección se muestran los resultados obtenidos, estos son principalmente simulaciones en Gazebo del comportamiento dinámico del SCARA en base a su diseño mecánico en CAD, trayectorias óptimas y controladores PID afinados mediante la minimización del MSE utilizando PSO. Primero se abordará el procedimiento para generar un URDF del robot y hacer simulaciones, luego la generación de trayectorias óptimas y por último la utilización de PSO para la afinación de controladores.

Es importante mencionar que se decidió utilizar esta metodología precisamente porque su primer hito es tener un buen modelo URDF y generar las simulaciones en Gazebo mediante ROS. Esto es muy poderoso en la medida que permite analizar el desempeño del robot en distintos escenarios solucionando problemáticas mas complejas y lejanas al ámbito del control. Por ejemplo se pueden probar diferentes diseños de pinzas o efectores en la punta e incluir objetos en la escena con los que el robot puede interactuar. Se pueden probar algoritmos de selección basados en visión computacional con cámaras simuladas. Se pueden incluir varios de estos brazos robóticos u otros robots y simular tareas conjuntas, lineas de producción etc. Se elige usar ROS para brindar una estructura flexible que permita simular y solucionar problemas reales y que por sobretodo, pueda ser utilizada por otros para continuar el desarrollo del SCARA.

7.1. Metodología de simulación

El diseño del SCARA esta descrito con gran detalle en su modelo en CAD, cada parte tiene información sobre su peso en base a la densidad del material y se tiene acceso a la matriz de inercia de cada pieza y conjunto. Para utilizar esta información en simulaciones se arma un URDF el cual es simulado posteriormente en Gazebo. En los anexos A se muestra el proceso con el cual se obtiene un URDF y los archivos de configuración necesarios para poder correr simulaciones en las que ademas se cargan a cada articulación un controlador PID. Esta es la estructura a nivel de control que se tendrá posteriormente en el robot armado.

Al cargar estas simulaciones en Gazebo y se puede mover el robot publicando la posición deseada en el tópico `/nelen/<nombre del controlador>/command`. La simulación corre indefinidamente y para

volver a la posición inicial se debe hacer de la misma forma, publicar en el tópico `/nelen/<nombre del controlador>/command`. Para correr simulaciones iterativamente no es adecuado este comportamiento pues las condiciones iniciales de una simulación dependerán del resultado de la anterior, un mal controlador podría hacer que una buen controlador tenga un mal desempeño por partir en un punto arbitrario. Para solucionar este problema se ocupan los servicios de ROS para controlar la simulación de gazebo, estos son `/gazebo/reset_simulation`, para reiniciarla, `/gazebo/pause_physics` para pausarla y `/gazebo/unpause_physics` para despausarla. La posición en la que se encuentran las articulaciones es publicada en 2 tópicos por Gazebo, estos son `/nelen/joint_states`, donde se publican los estados de todas las articulaciones; y `/nelen/<nombre del controlador>/state`, existe uno de estos tópicos por cada controlador cargado. La principal diferencia entre estos 2 tópicos es que al llamar al servicio para reinicializar la simulación se deja de publicar en el tópico `/nelen/joint_states` hasta que no se alcance el tiempo que se llevaba simulando cuando se reinicializó, esto tampoco es un comportamiento deseado para correr simulaciones de forma iterativa.

Finalmente, se desarrolla un nuevo nodo de ROS que se llama `run_once` el que corre una función homónima la cual tiene como entradas las ganancias de los controladores y punteros a las funciones de trayectoria. Esta función publica a una frecuencia fija los valores de trayectoria deseados y cuando se cumple el tiempo de simulación retorna los estados de las articulaciones en el tiempo, posteriormente reinicializa y pausa la simulación dejándola lista para correr en una siguiente iteración. Este es el bloque central para poder correr simulaciones iterativamente y así afinar los controladores. En la Figura 7.1 se gráfico de nodos de ROS con el nodo desarrollado conectado al simulador.

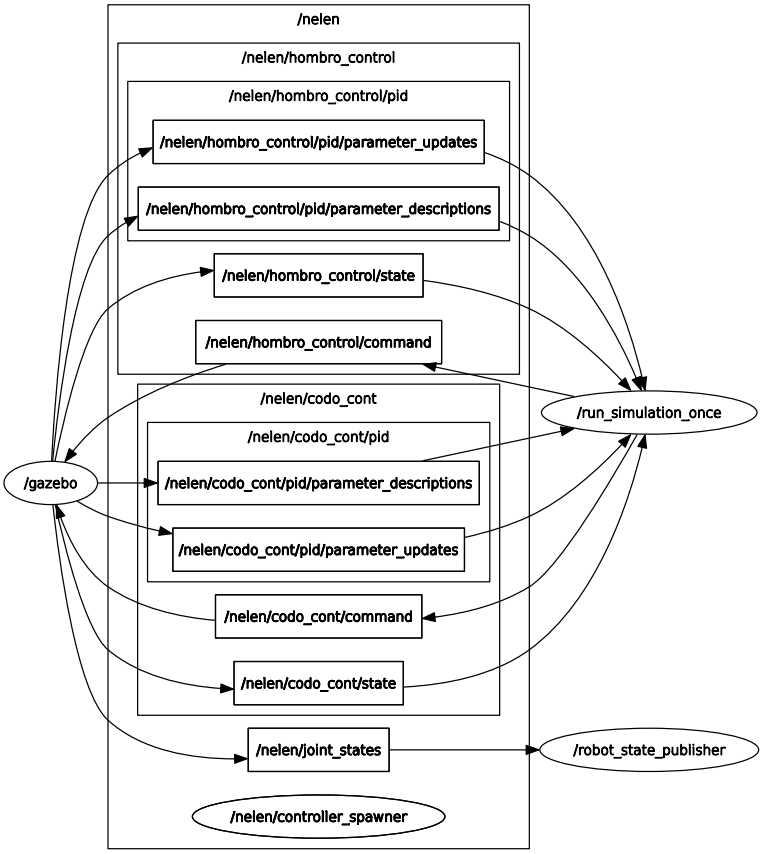


Figura 7.1: Diagrama de nodos del programa simulador

7.1.1. Trayectoria

Para generar la trayectoria a seguir se implementó el algoritmo de [27] en Python. Para su correcto funcionamiento es necesario conocer los máximos del robot, su máxima velocidad, aceleración y sobreaceleración. Para estimar la velocidad máxima se ocupa la ecuación 3.1, la que relaciona el voltaje de la fuente con la velocidad máxima que puede alcanzar el motor, posteriormente se divide por la relación de reducción de la caja reductora la cual es 7. Se asume que se alcanza a lo mas la mitad del voltaje. Con esto la velocidad máxima para ambos es igual a $37.38[rads/s]$. Si el codo y el hombro se movieran a esta velocidad entonces la velocidad en la punta sería aproximadamente $37.38[m/s]$, o $116.5[km/h]$, este es un escenario que no sucederá nunca, porque no hay recorrido suficiente para que, con la aceleración máxima, se alcance. Incluirlo como parámetro en el algoritmo no implica que en algún momento se alcance esta velocidad, solo que probablemente nunca sea la restricción de velocidad la que defina el tipo de curva. Para estimar el torque máximo se ocupó la ecuación de la aceleración angular $\dot{\omega} = \tau I$ donde τ es el torque, I es la inercia rotacional en el eje de giro y $\dot{\omega}$ es la velocidad angular. El torque máximo es uno de los parámetros de los motores, los cuales se muestran en 5.2. Para estimar el momento de inercia se utiliza el teorema de Steiner para sumar la inercia de los eslabones hijos a sus padres, esta información se obtiene desde el modelo CAD en *Inventor* y está guardada en el tag <inertial> del URDF. Así se determina que la máxima aceleración corresponde a $8.55[rads/s^2]$ para el hombro y $32.37 rads/s^2$ para el codo. Por último

para estimar la máxima sobreacelerecaión se toma en cuenta que esta es directamente proporcional a la variación en el torque y por ende directamente proporcional a la variación en la corriente, se asume que el *Odrive* tarda 50$[ms]$ en subir la corriente hasta el valor en el que el torque es máximo y lo es también la aceleración. Usando los valores de aceleración máxima calculados anteriormente se estima que la máxima sobreaceleración es 171.26$[rads/s^3]$ para el hombro y 647.4$[rads/s^3]$ para el codo. Bajo estos supuestos es que se generan las curvas de trayectoria, y deben ser verificados mediante mediciones una vez terminado el proceso de construcción del SCARA. En la Tabla 7.1 se muestra el resumen de los parámetros utilizados.

Tabla 7.1: Tabla resumen de los parámetros cinemáticos máximos para generar trayectorias y los supuestos con los que se calcularon.

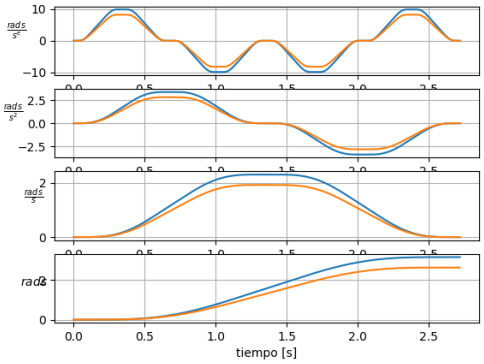
Parametro de restricción	Valor	Asunción
J_{max}^{hombro}	171.26$[m/s^3]$	50ms en alcanzar máxima corriente
A_{max}^{hombro}	8.5513$[m/s^2]$	I=5.26 kg m2 y T=45nm
V_{max}^{hombro}	37.38$[m/s]$	Mitad del voltaje
J_{max}^{codo}	647.4$[m/s^3]$	50ms en alcanzar máxima corriente
A_{max}^{codo}	32.37$[m/s^3]$	I=1.07kg m2 y T=35nm
V_{max}^{codo}	37.38$[m/s^3]$	Mitad del voltaje

Con los parámetros cinemáticos máximos solo falta uno, S_max, este es el parámetro que se deja libre para elegir que tan suave es la trayectoria generada. En [27] se ocupan valores entre 150 a 4000. En la Figura 7.2 se muestran trayectorias óptimas para el máximo desplazamiento posible en el codo (azul) y hombro (naranjo) con distintos valores de S_{max}. Para S_{max} igual a 50, 80, y 160, todas las curvas generadas son del tipo 1, esto significa que no se alcanzan a saturar ninguno de los parámetros máximos estimados anteriormente, solamente para $S_{max} = 1600$ el codo presentó una curva tipo 3, por lo que alcanzó su aceleración máxima, sin embargo, para que el movimiento de ambas articulaciones termine al mismo tiempo, la curva mas rápida se estira, por esta razón, tampoco en este caso se logra saturar ninguna de las restricciones.

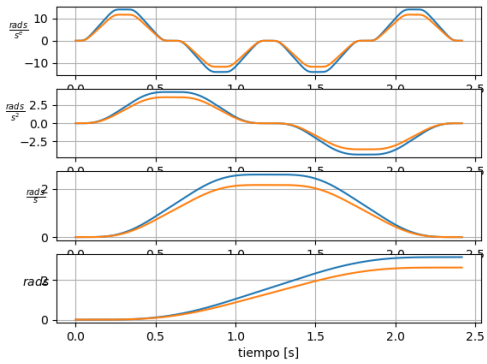
7.1.2. Afinación de PID

El objetivo de esta sección es obtener un sistema de control adecuado para el SCARA. Para realizar esto se ocupa un modelo de robot en Gazebo, este tiene implementados controladores PID en cada una de sus articulaciones y se pueden generar simulaciones iterativas para correr algoritmos de optimización. Se utiliza el algoritmo PSO para encontrar el set de ganancias que minimizan el error cuadrático medio (MSE) entre una referencia y el resultado de la simulación. Se hace este procedimiento para el codo y el hombro en conjunto.

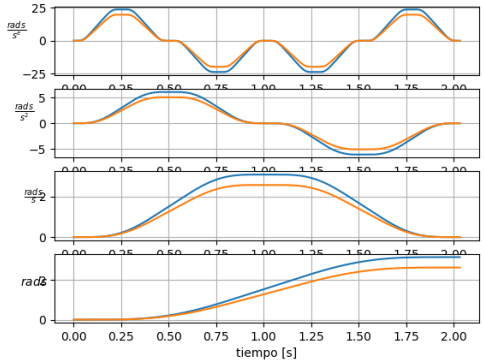
Primero, se busca replicar la estrategia seguida en [24]. Por esta razón se prueba primero con una referencia tipo escalón. Se busca que los controladores tengan un buen desempeño para las articulaciones del codo y el hombro cuando estas se mueven en conjunto, por esta razón se calculan las ganancias para ambos controladores en una prueba conjunta. Lo mas prudente es utilizar la trayectoria mas critica posible, esto



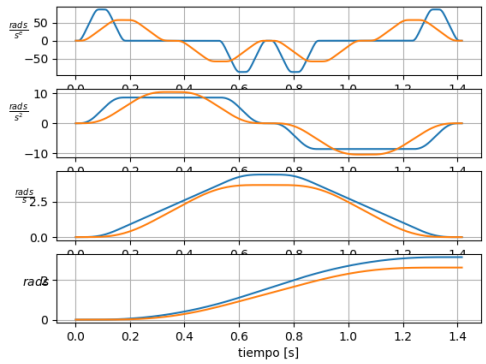
(a) $S_{max} = 50$



(b) $S_{max} = 80$



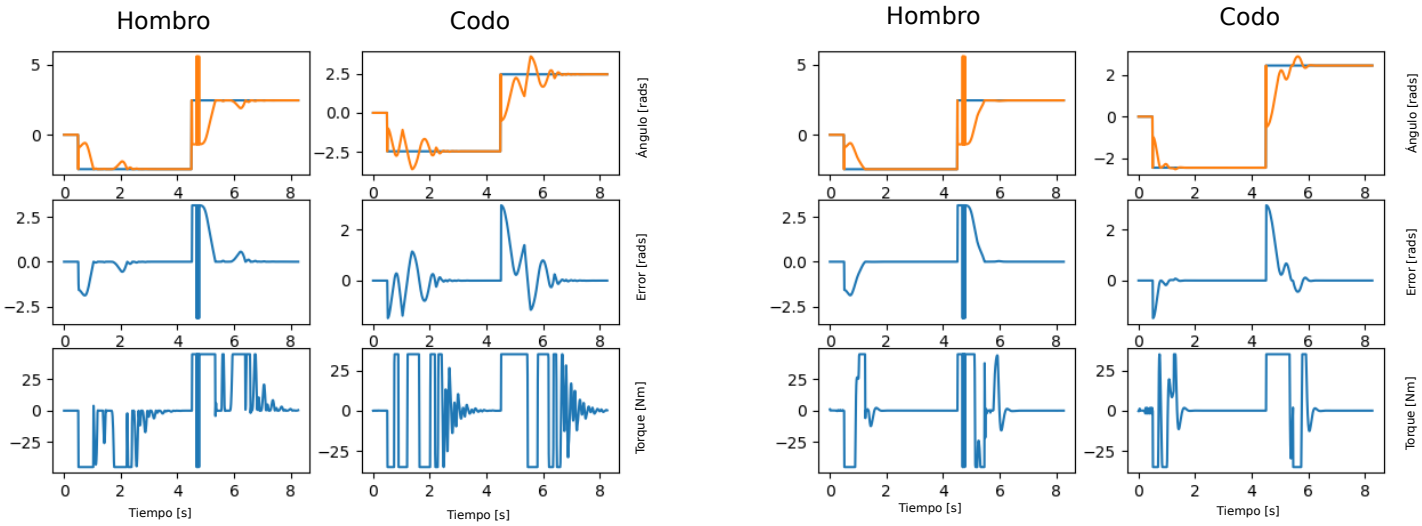
(c) $S_{max} = 160$



(d) $S_{max} = 1600$

Figura 7.2: Distintas trayectorias generadas para el máximo desplazamiento posible y distintos valores de S_{max}.

es, mover cada articulación desde uno de sus extremos al otro. Respecto al espacio de búsqueda este es de 6 dimensiones, por las 3 ganancias de cada controlador. Sus limites son $[0, 1000]$ para las ganancias proporcionales, y $[0, 500]$ para la integral y proporcional. La elección del límite de la ganancia proporcional es porque en su valor máximo 1000 se consigue el máximo torque para un error cercano a los 2 grados, los otros limites se eligen para mantenerse en el mismo orden de magnitud que la proporcional. Por último se elige utilizar 50 partículas y 20 iteraciones y se hace este procedimiento para un controlador PID y un controlador PI. En la Figura 7.3 se muestra el desempeño de la prueba conjunta para ambos controladores con las mejores ganancias obtenidas. En la Figura el par de gráficos superiores corresponden a la posición respecto al tiempo, el par de al medio es el error y el par de mas abajo corresponde al torque en la articulación, la parte izquierda corresponde al controlador PI y la derecha al controlador PID. Se aprecia que el resultado es malo, en primer lugar si bien las posiciones convergen a la referencia tienen sobrepasos muy elevados en los instantes en los que se hace el cambio de referencia. Se aprecia que el controlador PID tuvo un mejor desempeño pues oscila mucho menos. Respecto al torque de las articulaciones este tiene grandes oscilaciones cuadradas entre su valor máximo y mínimo, este tipo de control es malo porque se estaría exigiendo el máximo al motor y a todo el conjunto mecánico, lo que reduciría fuertemente la vida útil del robot.



(a) Controlador PI

(b) Controlador PID

Figura 7.3: Gráficos del desempeño de las mejores ganancias obtenidas para una señal cuadrada. A la izquierda se encuentra el controlador PI y a la derecha el PID

Con el fin de mejorar el pobre desempeño logrado anteriormente se decide cambiar el tipo de referencia, se utilizan las trayectorias óptimas mostradas en la sección anterior. En la Figura 7.4 se muestran las trayectorias generadas. En estas pruebas se parte la simulación con el robot en su posición central para los ejes del codo y el hombro. Luego se avanza hacia el extremo izquierdo y finalmente hacia su extremo derecho. Se eligió esta trayectoria porque mueve el brazo en ambas direcciones y es el cambio de posición mas extremo que se puede realizar, también se espera que en esta trayectoria se alcance la máxima

velocidad. Se utilizan 3 valores del parámetro de diseño S_{max}, estos son 50, 80 y 160. En la Figura se aprecia que a medida que decrece S la curva generada se hace mas suave pero aumenta la duración de la prueba. El tiempo en que se mantiene contante la referencia en su valor máximo y mínimo es 2 segundos. Finalmente, la duración de la prueba es 9.616 segundos para $S_{max} = 50$, 8.976 segundos para $S_{max} = 80$ y 8.192 para $S_{max} = 160$

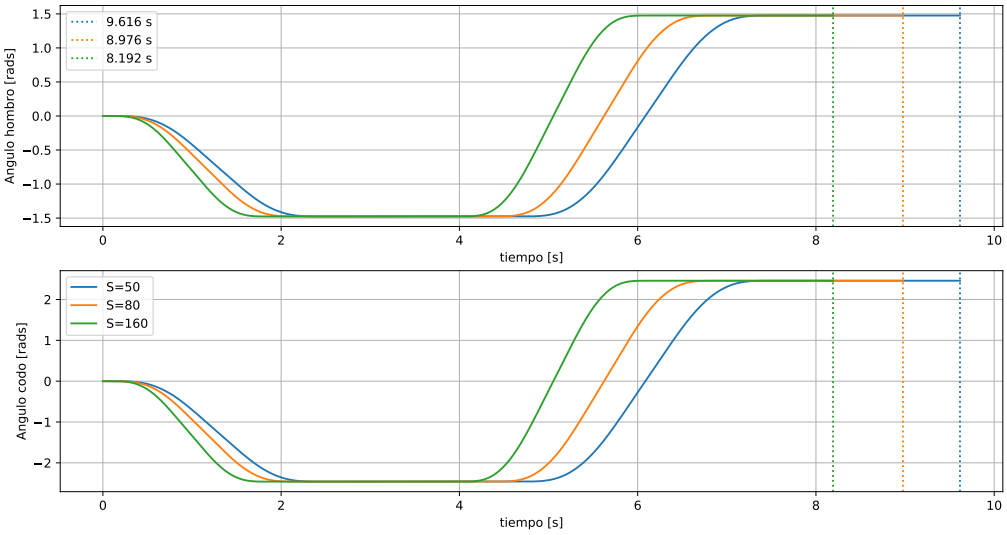
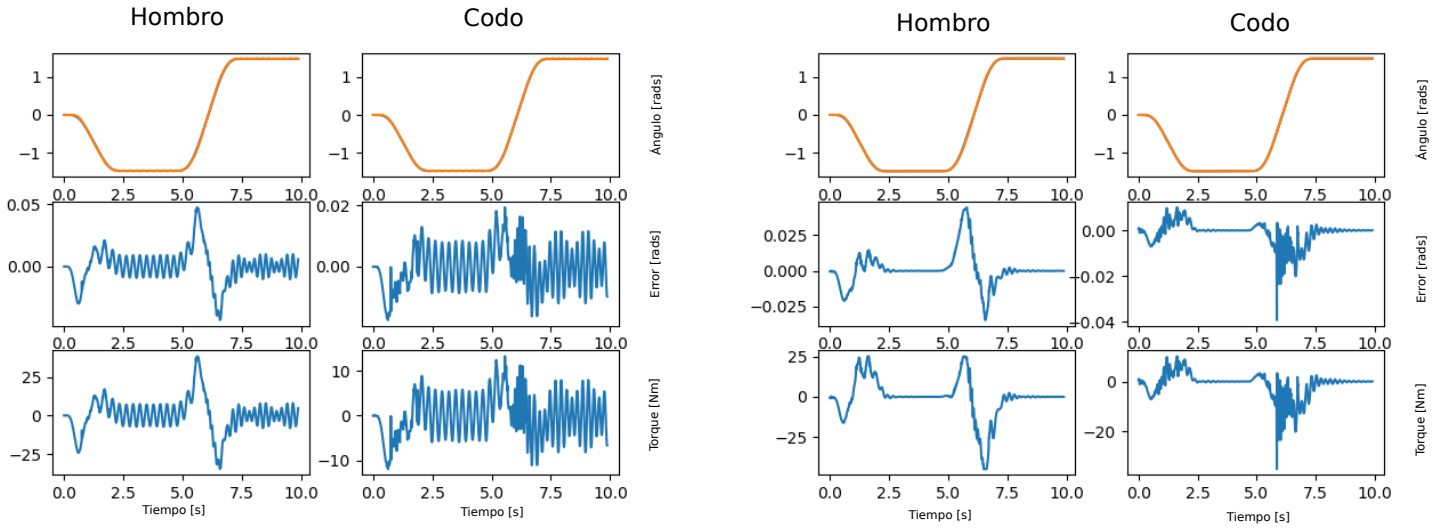


Figura 7.4: Trayectorias de referencia utilizadas para la optimización. Se ocupan 3 valores de S_max, estos son 50 en azul, 80 en naranjo y 160 en verde. Las lineas punteadas marcan el final de la trayectoria.

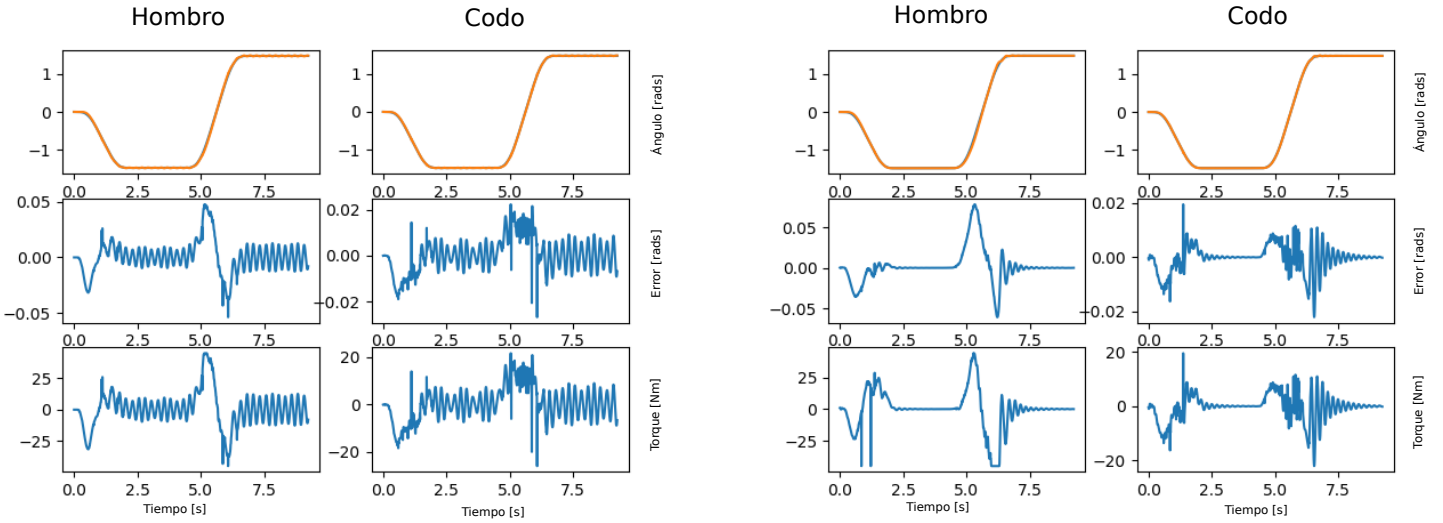
En las Figuras 7.5, 7.6 y 7.7 se muestran el desempeño de los mejores controladores obtenidos para $S_{max} = 50, 80 y 160$ respectivamente. Los resultados están ordenados de la misma forma que en 7.3. En general se aprecia una mejoría en el desempeño de los controladores, el error se reduce considerablemente y el sobrepaso se hace presente solo para $S_{max} = 160$. Se aprecia además la diferencia que provoca la presencia de la ganancia derivativa, la cual reduce las oscilaciones en todos los casos y sobretodo en el torque. La diferencia que existe entre el desempeño del mejor controlador para $S_{max} = 160$ y para $S_{max} = 50$ es muy grande, en la primera el controlador satura el torque constantemente y presenta oscilaciones sostenidas al final de la curva para el caso del controlador PI, en el caso de la segunda esta nunca satura el torque máximo ni presenta oscilaciones sostenidas apreciables en ninguno de sus 2 controladores. Aun con esta gran diferencia en desempeño la diferencia en tiempo entre ambas pruebas es de solo 1.47 segundos.



(a) Controlador PI

(b) Controlador PID

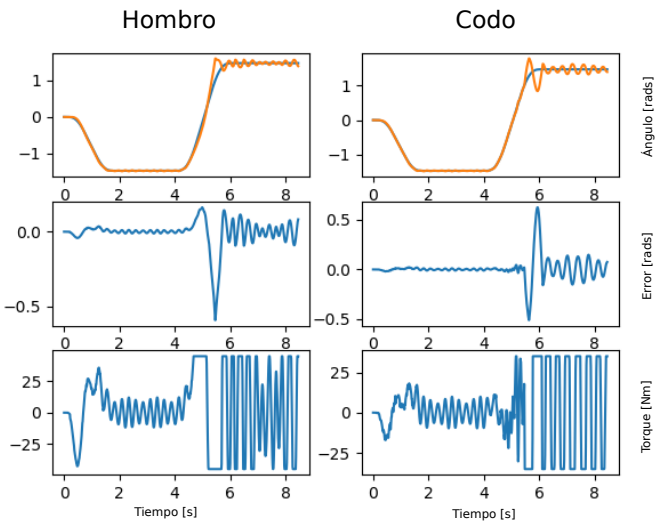
Figura 7.5: Gráficos del desempeño de las mejores ganancias obtenidas para $S_{max} = 50$. A la izquierda se encuentra el controlador PI y a la derecha el PID



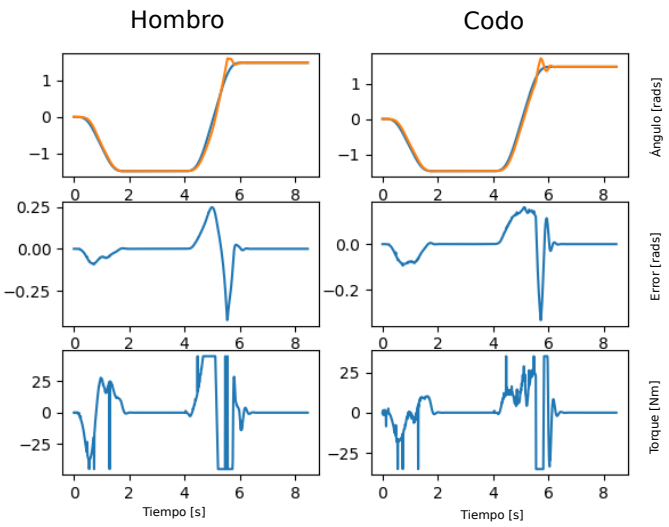
(a) Controlador PI

(b) Controlador PID

Figura 7.6: Gráficos del desempeño de las mejores ganancias obtenidas para $S_{max} = 80$. A la izquierda se encuentra el controlador PI y a la derecha el PID



(a) Controlador PI



(b) Controlador PID

Figura 7.7: Gráficos del desempeño de las mejores ganancias obtenidas para $S_{max} = 160$. A la izquierda se encuentra el controlador PI y a la derecha el PID

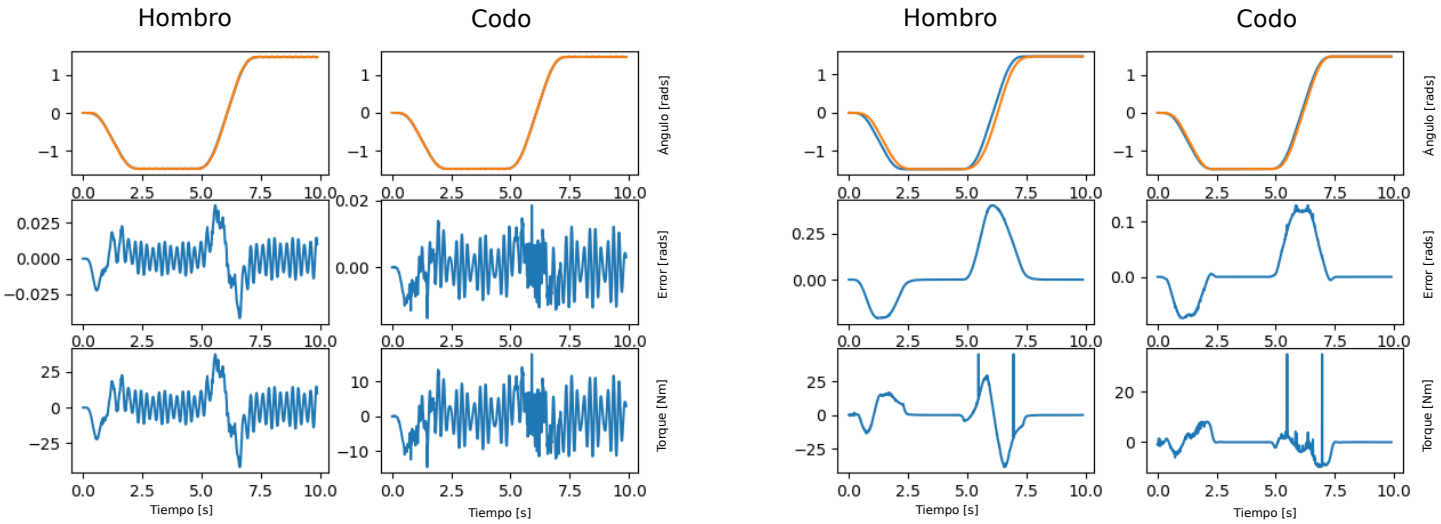
En la Tabla 7.2 se muestran los resultados del algoritmo PSO para todos los casos probados. Aquí se confirma el mejor desempeño de los controladores PID para $S_{max} = 50$ dado su mínimo MSE, esto significa que esta es la trayectoria que mejor se pudo seguir. En la tabla también se muestra que para el controlador del codo la ganancia derivativa no mejora los resultados, por esta razón en todos excepto uno de los casos se dejo en su valor mínimo 0. Si bien este valor se mantiene bajo para el controlador del hombro en este tiene un efecto crítico, bajando el error a 1/5 para $S_max = 50$ y en un orden de magnitud para $S_max = 50$.

Tabla 7.2: Ganancias para controladores PID del hombro y codo para distintas trayectorias y su respectivo desempeño. Para cada valor de S_max, la fila de arriba corresponde al controlador PI y la de abajo al PID.

Smax	Hombro			Codo			MSE
	P	I	D	P	I	D	
50	805.14	474.92	0	686.17	12.17	0	5.21e-05
	1000	500	10.31	1000	47.09	0	1.6e-06
80	1000	0	0	968.22	0	0	7.77e-05
	1000	461.2	17.24	1000	297.97	0	4.5e-06
160	1000	460.5	0	861.55	0	0	6.17e-03
	1000	500	47.31	786.28	194.08	23.16	9.6e-05
cuadrada	1000	30.34	0	957.27	500	0	6.7e-02
	523.32	73.95	105.97	512	0	19.75	1.1e-02

El procedimiento mostrado mostró ser adecuado para conseguir ganancias de controladores aptos para trayectorias suaves, la incorporación de estas permite mejorar dramáticamente el comportamiento dinámico del sistema y también ayuda a reducir el torque máximo y las oscilaciones, para dar otro ejemplo de esto se ha incluido la Figura 7.8, en la que se prueban las mejores ganancias obtenidas para el controlador PI y PID con una referencia cuadrada para una referencia suave con $S_{max} = 50$. Es importante notar que en general en los gráficos de torque se aprecia un comportamiento periódico de alta frecuencia, esto

parece no afectar en gran medida el movimiento de la articulación como se aprecia en la Figura 7.6b, este comportamiento difiere de la realidad, un motor eléctrico es una carga inductiva lo que lo hace muy sensible a entradas periódicas, por lo que en la realidad la salida no será como en la simulación, por esta razón en [10] se incluye el efecto de los motores DC en su modelo. Se recomienda como trabajo futuro incluir el efecto de los motores en las simulaciones de Gazebo. Si bien el esquema de control de los *Odrive* no incluye una ganancia derivativa, si tiene un modo de control de corriente, por lo que el mejor controlador obtenido en esta sección podría ser programado directamente en Python y utilizado en el robot real.



(a) Controlador PI

(b) Controlador PID

Figura 7.8: Gráficos del desempeño de las mejores ganancias obtenidas para con una referencia cuadrada usadas en una referencia suave con $S_{max} = 50$. A la izquierda se encuentra el controlador PI y a la derecha el PID

Capítulo 8

Conclusiones

En este trabajo de memoria se detalla el proceso de construcción de la parte eléctrica encargada del control de un SCARA. Esto comprende desde la elección de componentes a su implementación en un robot real fabricado en conjunto con un memorista de ingeniería mecánica. En referencia a los objetivos específicos:

- Modelar el funcionamiento de un brazo robótico con cuatro grados de libertad: Se logra generar un modelo de robot en formato URDF el cual se puede cargar en ROS. Las características dinámicas de este modelo se desprenden del detalle de los materiales con que fue diseñado el robot, el cual es especificado previamente en el software en que fue diseñado. Este resultado es importante porque al estar en ROS se pueden probar fácilmente futuros desarrollos, ya sean algoritmos de selección basados en visión computacional, o trabajos colaborativos entre múltiples robots entre muchas otras posibilidades. ROS cuenta con potentes herramientas y una comunidad de cientos de investigadores en todo el mundo.
- Determinar estrategias de control y elegir la mas pertinente en base a simulaciones: Se prueba una estructura de controlador muy similar a la que se puede aplicar con los componentes elegidos y se afinan las ganancias mediante un algoritmo iterativo (PSO). Para mejorar los resultados de este procedimiento se implementa un algoritmo para generar trayectorias suaves, lo que mejora en gran medida el desempeño de los controladores, este es el principal resultado de este procedimiento. Si bien el controlador obtenido no es directamente aplicable si lo es la generación de trayectorias óptimas.
- Generar lista de componentes que conformaran el sistema eléctrico y de control del robot: este objetivo se cumplió en gran medida pues no solo se generó una lista de materiales adecuada sino que se compraron e integraron en el robot real. A la entrega de este informe la construcción no ha terminado pero si se han conectado todos los componentes por separado y confirmado su compatibilidad.
- Desarrollar un programa que permita la operación interconectada y controlada de los componentes del robot: Se crea un programa basado en una clase `nelen`, la cual permite operar el robot de manera segura y moverlo dentro de su espacio de trabajo de forma coordinada.

- Implementar el sistema de control en un brazo robótico construido por un memorista de ingeniería mecánica: a la entrega de este informe el proceso de construcción no ha terminado pero si se ha probado la factibilidad de implementación, pues todos los elementos funcionan por separado y coinciden con el ensamblaje mecánico del brazo.
- Evaluar el desempeño del sistema desarrollado: Se realizan pruebas en base a simulaciones obteniéndose alta precisión, sin embargo estas pruebas no toman en consideración la resolución posible con los sensores con los que cuenta el robot. Este objetivo será cumplido una vez terminada la construcción.

En la Figura 8.1 se muestra una fotografía del estado actual de la construcción del robot, se está terminando de realizar el cableado con lo que se podrá operar el robot como una unidad. En este proceso se ha confirmado que no hay incompatibilidades entre la parte eléctrica y mecánica, esto es pues el proceso de diseño de ambas partes se realizó de forma iterativa entre los memoristas. Este es el principal logro de este trabajo de memoria, se cuenta con un robot fabricado y tangible. Por otra parte este robot existe en formato URDF y se puede simular en ROS, lo cual es una potente herramienta para quienes continúen el desarrollo de este robot. Por otra parte, todos los planos, eléctricos y mecánicos se encuentran en linea en el Github del proyecto, lo que extiende su replicabilidad de fabricación y modelamiento a quien lo necesite.

Finalmente, el proceso de diseño del sistema eléctrico del robot es muy amplio y cada arista tiene una complejidad no menor. En este sentido, esta memoria toca estas aristas en la complejidad suficiente para permitir la implementación de dicho sistema. Esto deja lugar a muchas mejoras y posibles temas de memoria para que otros alumnos continúen con este proyecto.

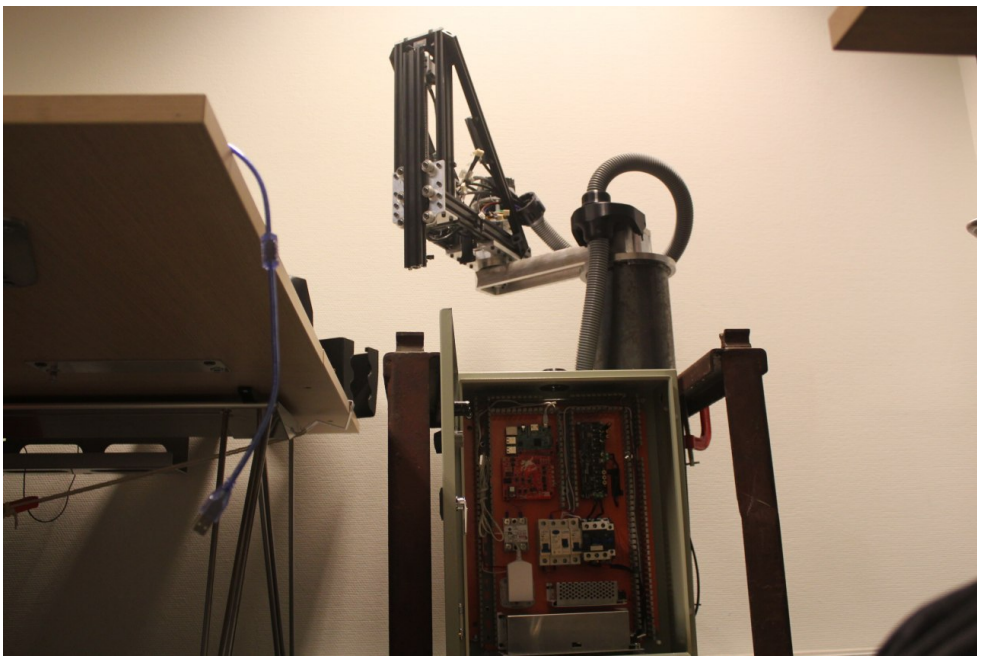


Figura 8.1: Estado actual del SCARA.

Anexos

A . Generación de URDF

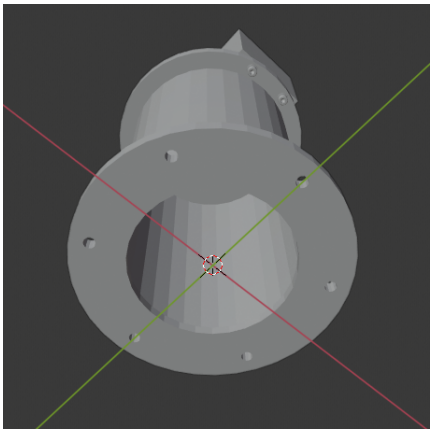
Para generar el URDF se importó el modelo del robot de Inventor a Solid Work y se utilizó sw_urdf_exporter (). Este programa genera un paquete de ROS con varias carpetas, entre estas una que se llama URDF y otra que se llama meshes. En la carpeda URDF se encuentra el urdf inicial, con la definición de los eslabones y articulaciones, no se generan los tag <transmission>. En la carpeta Meshes se incluyen los modelos 3d de los links del robot en formato STL, este es el mismo que se usa para hacer impresiones 3D. Al revisar tanto el urdf como los STL se nota que el primero está definido de forma semi-arbitraria, tanto las rotaciones que tienen los eslabones como la posición en la que se ubican está definida según el centro del modelo en STL, el cual se obtiene de múltiples relaciones en el proceso de conversión de *Inventor* a *SolidWorws*. Por esta razón el primer paso es modificar los modelos 3d, procurando dejarlos todos apuntando hacia el mismo vector (Z) y poniendo los centros en puntos con alguna característica particular, se elige el punto mas bajo del centro de rotación de los eslabones rotacionales y para el eslabón Z en la superficie trasera del V-slot a la altura del efector final. En la Figura 8.2 se muestra la ubicación de los 4 centros elegidos, este procedimiento fué hecho en *blender* y los archivos se guardaron en .dae.

También, al hacer la conversión de un programa a otro tampoco se mantienen correctamente los pesos ni los tensores de inercia por lo que este parámetro debe ser actualizado a mano en el tag <inertial>. La posición del centro de masa respecto al origen definido previamente y el tensor de inercia, se obtienen directamente en *Inventor*, en la Figura 8.3 se muestar un ejemplo este procedimiento. Un detalle importante es que en el URDF la inercia se debe definir en $Kg * m^2$ y en *Inventor* está en $Kg * mm^2$.

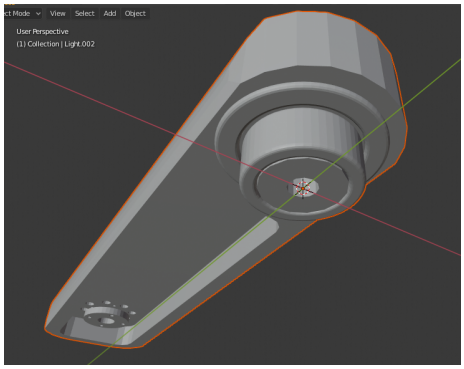
Posteriormente se modifica el URDF, en particular se cambia deja tanto en <visual> como en <colision> los orígenes en XYZ=0,0,0 y rpy=0,0,0. El ajuste de posición entre eslabones se realiza en el *tag* <joint> respectivo. También se modifica el archivo a cargar en la subpestaña <geometry> <mesh> para incluir el archivo modificado en la articulación que corresponda.

Luego se carga el asistente de configuración de *MoveIt!*, para esto se sigue el procedimiento mostrado en Tutorial asistente MoveIt!.

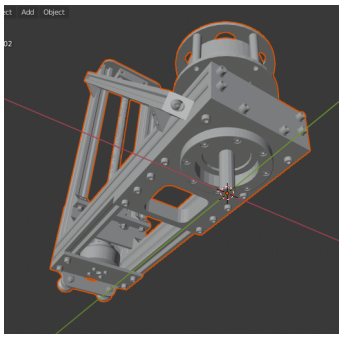
Se carga el URDF en la opción “create new Move It configuration package y se indica la dirección del



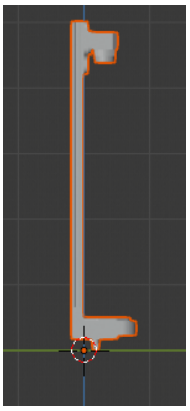
(a) Base



(b) Humero

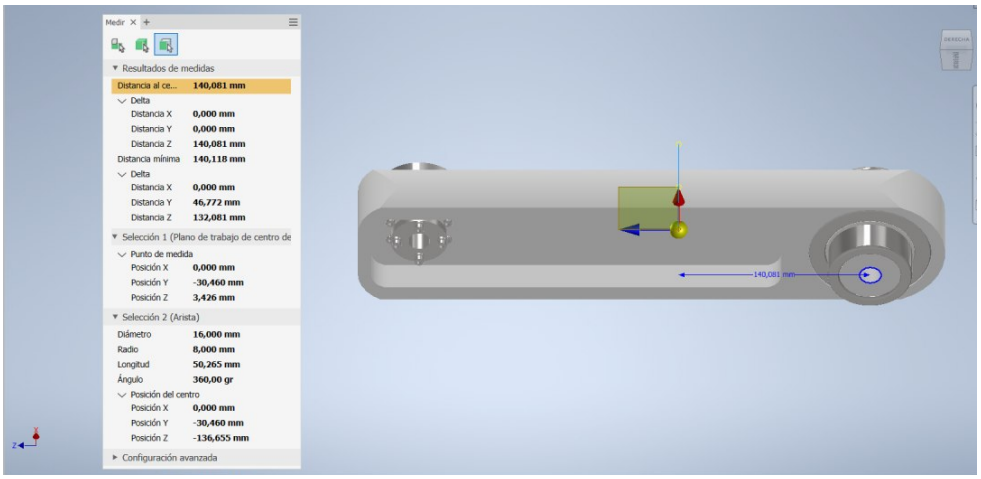


(c) Radio-cúbito

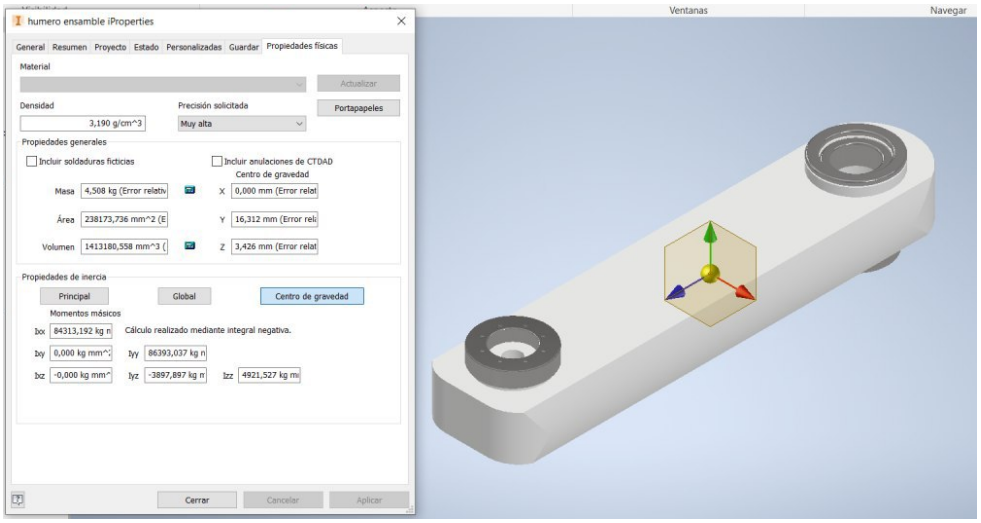


(d) Zeta

Figura 8.2: Posición de los orígenes en los eslabones del SCARA.



(a) Distancia al centro de masa



(b) Tensor de inercia

Figura 8.3: Pantallas en *Inventor* donde se obtiene la información de inercia del Húmero.

URDF modificado anteriormente. y se selecciona load files 8.4.

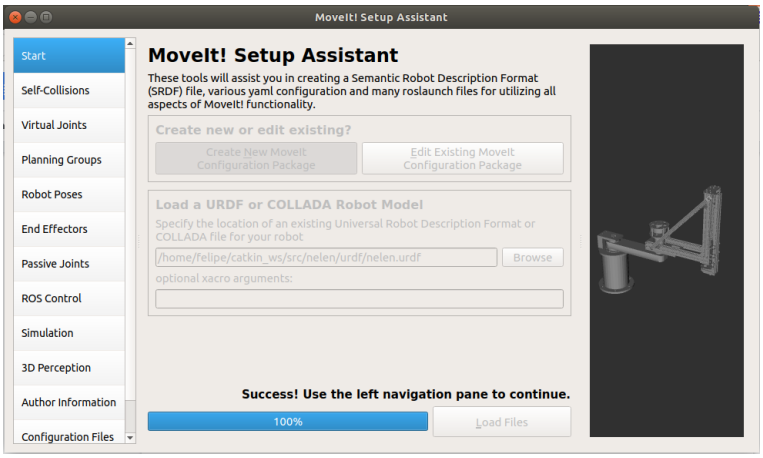


Figura 8.4: Paso número 1 del asistente de *MoveIt!*

Luego se avanza a la pestaña *self-colisions* (Figura 8.5, en esta pestaña se lleva a cabo un proceso en el que se pone el robot en todas sus posiciones posibles y se determina puntos de colisión, además las colisiones que se presentan un 95 % del tiempo son ignoradas pues se asume que son eslabones adjuntos. se presiona la pestaña y luego el botón "*generate colition matrix*"

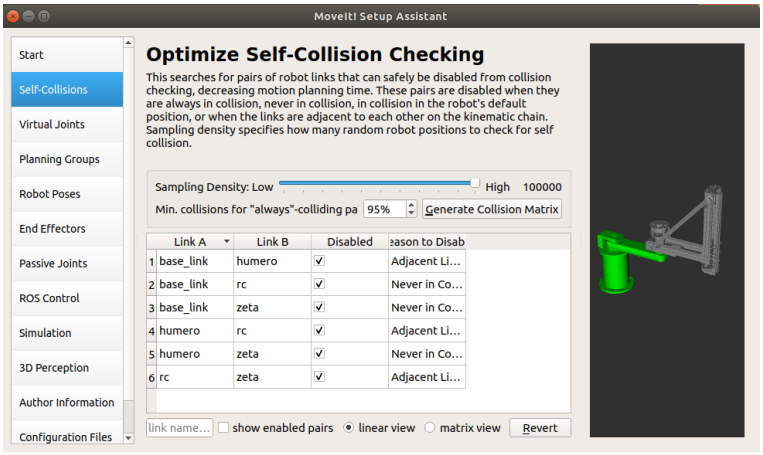


Figura 8.5: Paso número 2 del asistente de *MoveIt!*

Se avanza a la pestaña *virtual joints* como en la Figura 8.6, en esta pestaña se generan eslabones virtuales, en este caso se crea uno para fijar el robot al piso. Se presiona *add virtual joint*, se selecciona un nombre, en este caso "suelo", el eslabón hijo *child link* se mantiene en *base_link* y en *parent frame* se escribe *"world"*, el tipo de articulación es *fixed* (fijo).

Luego se avanza a la pestaña *planing groups*, Figura 8.7. Un *planing group* es un conjunto de articulaciones sobre las cuales move it planifica un movimiento coordinado para evitar colisiones, esto tiene más sentido cuando se tienen varios brazos robóticos o un robot compuesto por muchas partes, por ejemplo piernas, brazos, cabeza etc. Para este caso se ocupa solo un *planing group* en el que se seleccionan todos los joints, para esto se aprieta el botón *add group*, se elige un nombre para el brazo, en este caso nelen, se selecciona

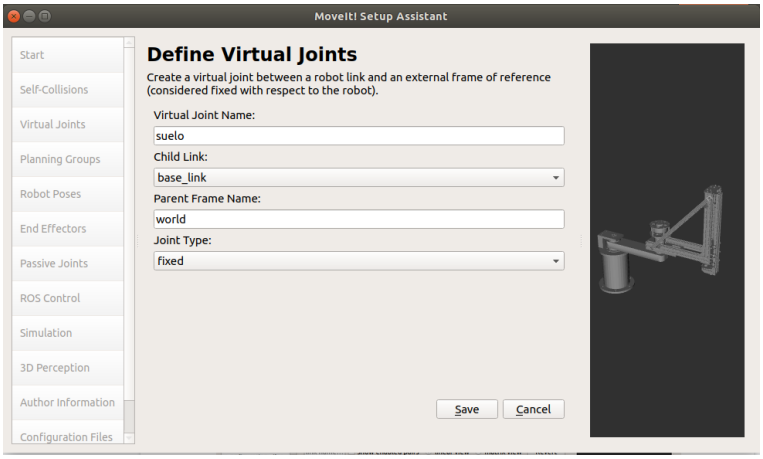


Figura 8.6: Paso número 3 del asistente de *MoveIt!*

el *KDL kinematics solver*, esta es la opción estándar, un *kinematic solver* es un algoritmo para determinar la configuración de las articulaciones que es necesaria para que la punta quede en un punto determinado, o sea la solución de la cinemática inversa. Se mantiene el resto de las opciones como están y se aprieta el botón *add joints* y se seleccionan todos, luego *save*.

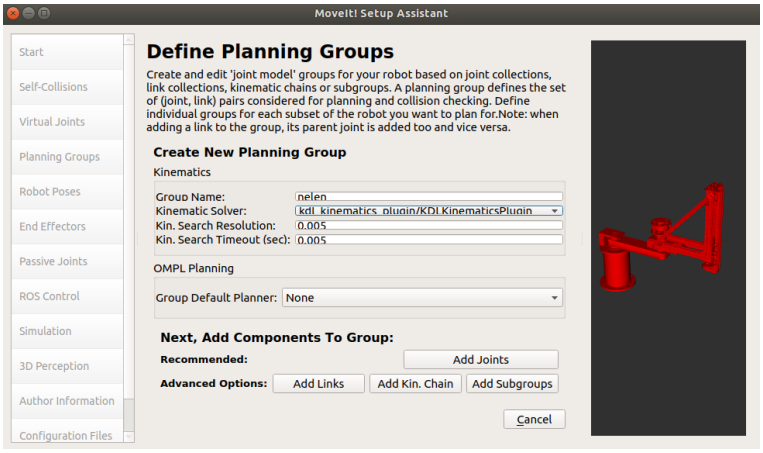


Figura 8.7: Paso número 4 del asistente de *MoveIt!*

Se avanza a la pestaña *poses*, Figura 8.8. Aquí se seleccionan poses determinadas, para elegirlas se tiene acceso a una GUI donde se puede mover las articulaciones del robot, mediante esta se puede detectar rápidamente cualquier error en la definición del URDF.

Se avanza a la pestaña de *end efectors*, aquí se elige la opción *add end efector* y se agrega nelen como el *end effector group*. Posteriormente se avanza a la pestaña *add controller* Figura 8.9, este es realmente el único paso diferente a la guia oficial de *MoveIt!*. Se selecciona *add controller*, luego se elige un nombre de controlador y se lo asocia para una articulación con el botón *Add individual joints* y se selecciona una articulación de la lista. El tipo de controlador elegido es `effort_controller/JointPositionController`, esto significa que el controlador recibe posiciones y tiene como salida un esfuerzo, torque en el caso de los links rotacionales y fuerza en el caso de los prismáticos. Esto se hace para cada joint y se elige como

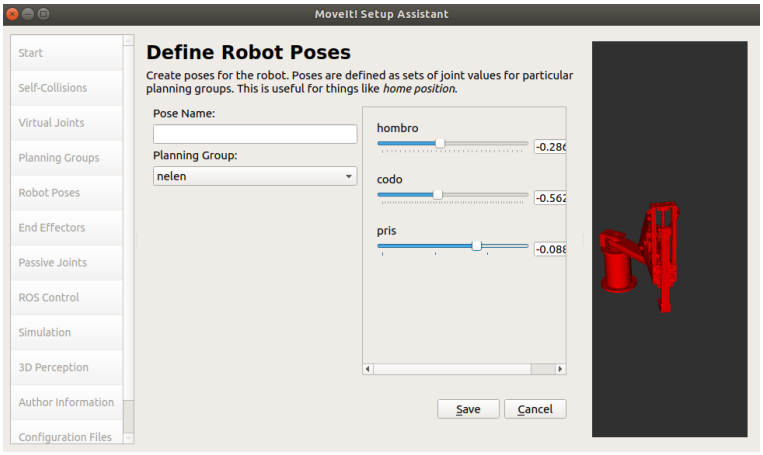


Figura 8.8: Paso número 5 del asistente de *MoveIt!*

nombre del controlador "nombreArticulación_control"

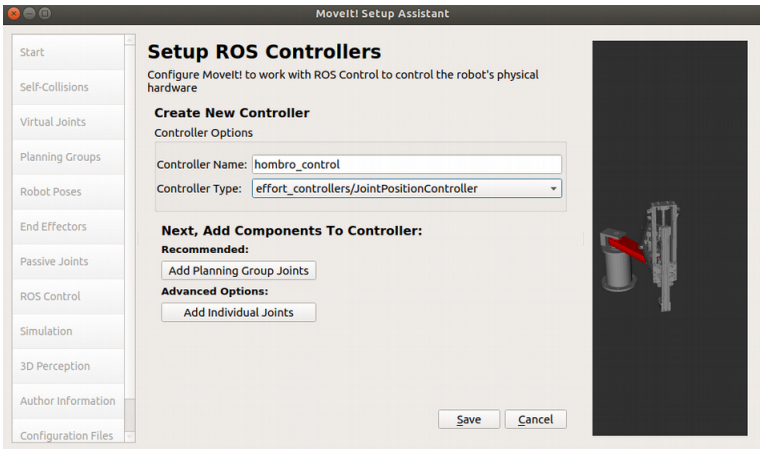


Figura 8.9: Paso número 6 del asistente de *MoveIt!*

Luego se avanza a la pestaña *simulation*, Figura 8.10 aquí se presiona el botón *generate urdf*, este botón extiende la definición del urdf, agregando los tags <transmission>, estos se utilizan para especificar los controladores de las articulaciones y el tipo de transmisión mecánica. También añade un tag llamado <gazebo> que es necesario para hacer simulaciones en Gazebo.

En *author information* se añade la info del autor y finalmente en *configuration files*, Figura 8.11, se elige una dirección en la que generar el paquete de configuración y un nombre para esta, en este caso "nelen_config"se presiona el botón *generate package* y listo.

Ahora es necesario modificar los archivos generados. En primer lugar en el urdf es necesario modificar el nombre del robot en el tag <gazebo><robotNamespace>, en esta ocasión se elige **nelen**

También es necesario modificar el archivo de configuración de los controladores, este se encuentra en la carpeta generada mediante el asistente, en la subcarpeta llamada config y se llama ros_controllers.yaml, en este archivo, en la última parte, se crea un tag con el nombre del robot descrito anteriormente y se incluye

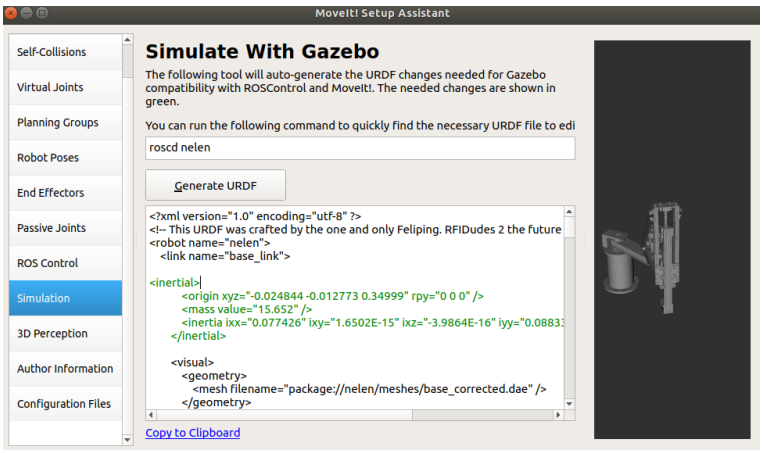


Figura 8.10: Paso número 7 del asistente de *MoveIt!*

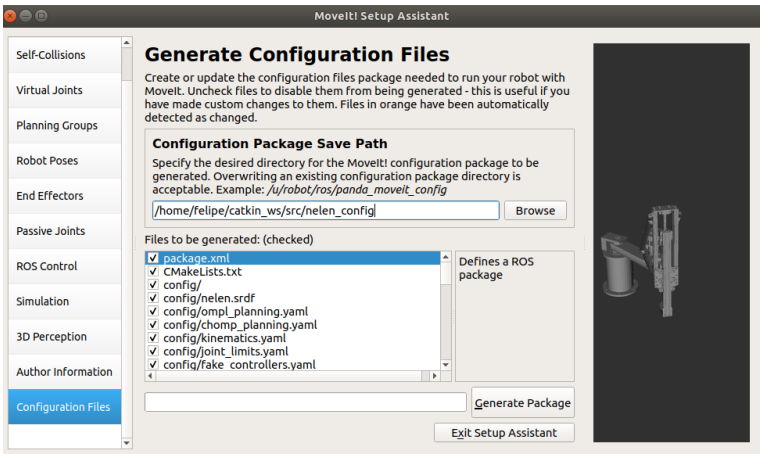


Figura 8.11: Paso número 8 del asistente de *MoveIt!*

dentro de este el joint_state_controller. Además se mueven a este tag todos los controladores que quedan al final del archivo, estos se identan añadiendo 2 espacios al inicio de todas las líneas de aquí al final del archivo. En cada controlador se cambia la palabra joints por joint y gains por pid, además los parámetros del controlador se expresan en una sola línea con paréntesis de llave, el controlador final queda definido por ejemplo:

hombro_control:

type: effort_controllers/JointPositionController

joint: hombro

pid: {p: 100.0, i: 0.01, d: 10.0}

Finalmente, en la carpeta generada por el asistente, en la subcarpeta launch, se modifica el archivo ros_controllers.launch, entre output y args se genera un nuevo parámetro llamado ns="/nelen". Se añade también el joint_state_controler en el parámetro args. Con todo esto se puede correr sin problemas las simulaciones en gazebo con la linea "`roslaunch nolen_config gazebo.launch`"

A continuación se incluye el URDF del SCARA generado con este procedimiento.

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- URDF for SCARA robot -->

<robot name="nelen">
    <link name="base_link">
        <inertial>
            <origin xyz="-0.024844 -0.012773 0.34999" rpy="0 0 0" />
            <mass value="15.652" />
            <inertia ixx="0.077426" ixy="1.6502E-15" ixz="-3.9864E-16" iyy="0.08833" iyz
                ="9.4391E-06" izz="0.07706" />
        </inertial>
        <visual>
            <geometry>
                <mesh filename="package://nelen_description/meshes/base_corrected.dae" /
                    >
            </geometry>
            <material name="">
                <color rgba="0.75294 0.75294 0.75294 1" />
            </material>
        </visual>
        <collision>
            <origin xyz="0 0 0" rpy="0 0 0" />
            <geometry>
                <mesh filename="package://nelen_description/meshes/base_corrected.dae" /
                    >
            </geometry>
        </collision>
    </link>

    <link name="humero">
        <inertial>
            <origin xyz="1.0851E-08 0.048114 0.14266" rpy="0 0 0" />
            <mass value="5" />
            <inertia ixx="0.019084" ixy="-1.4804E-10" ixz="-2.2963E-09" iyy="0.020036"
                iyz="-0.00043182" izz="0.0013447" />
        </inertial>

        <visual>
            <origin xyz="0 0 0" rpy="0 0 0" />
            <geometry>
                <mesh filename="package://nelen_description/meshes/humero_corrected.dae"
                     />
            </geometry>
            <material name="">
                <color rgba="0.75294 0.75294 0.75294 1" />
            </material>
        </visual>
        <collision>
            <origin xyz="0 0 0" rpy="0 0 0" />
            <geometry>
                <mesh filename="package://nelen_description/meshes/humero_corrected.dae"
                     />
            </geometry>
        </collision>
    </link>
```

```
50
51      <joint name="hombro" type="revolute">
52          <origin xyz="0 0 0.262" rpy="0 0 0" />
53          <parent link="base_link" />
54          <child link="humero" />
55          <axis xyz="0 0 1" />
56          <limit lower="-1.57" upper="1.57" effort="45" velocity="8" />
57      </joint>
58
59      <link name="rc">
60          <inertial>
61              <origin xyz="6.9227E-07 0.15158 0.13901" rpy="0 0 0" />
62              <mass value="15" />
63              <inertia ixx="0.011379" ixy="1.7481E-08" ixz="-5.0618E-08" iyy="0.007139"
                    iyz="-0.0017743" izz="0.0062172" />
64          </inertial>
65
66          <visual>
67              <origin xyz="0 0 0" rpy="0 0 0" />
68              <geometry>
69                  <mesh filename="package://nelen_description/meshes/rc_corrected.dae" />
70              </geometry>
71              <material name="">
72                  <color rgba="0.75294 0.75294 0.75294 1" />
73              </material>
74          </visual>
75          <collision>
76              <origin xyz="0 0 0" rpy="0 0 0" />
77              <geometry>
78                  <mesh filename="package://nelen_description/meshes/rc_corrected.dae" />
79              </geometry>
80          </collision>
81      </link>
82
83      <joint name="codo" type="revolute">
84          <origin xyz="0 -0.33 0.052" rpy="0 0 0" />
85          <parent link="humero" />
86          <child link="rc" />
87          <axis xyz="0 0 1" />
88          <limit lower="-2.61" upper="2.61" effort="35" velocity="8" />
89      </joint>
90
91      <link name="zeta">
92          <inertial>
93              <origin xyz="-0.011881 0.12518 -0.016578" rpy="0 0 0" />
94              <mass value="2" />
95              <inertia ixx="0.0028275" ixy="3.4325E-06" ixz="2.9995E-10" iyy="0.00010967"
                    iyz="9.9633E-12" izz="0.0028174" />
96          </inertial>
97          <visual>
98              <origin xyz="0 0 0" rpy="0 0 0" />
99              <geometry>
100                 <mesh filename="package://nelen_description/meshes/z_corrected.dae" />
101             </geometry>
102             <material name="">
103                 <color rgba="0.75294 0.75294 0.75294 1" />
```

```
            </material>
        </visual>
        <collision>
            <origin xyz="0 0 0" rpy="0 0 0" />
            <geometry>
                <mesh filename="package://nelen_description/meshes/z_corrected.dae" />
            </geometry>
        </collision>
    </link>

    <joint name="pris" type="prismatic">
        <origin xyz="0 -0.3534 -0.049" rpy="0 0 0" />
        <parent link="rc" />
        <child link="zeta" />
        <axis xyz="0 0 1" />
        <limit lower="-0.268" upper="0" effort="70" velocity="5" />
    </joint>
    <joint name="suelo" type="fixed">
        <parent link="world"/>
        <child link="base_link"/>
    </joint>

    <transmission name="trans_hombro">
        <type>transmission_interface/SimpleTransmission</type>
        <joint name="hombro">
            <hardwareInterface>hardware_interface/EffortJointInterface</
                hardwareInterface> <!-- aqui poner interfaz con odrive -->
        </joint>
        <actuator name="hombro_motor">
            <hardwareInterface>hardware_interface/EffortJointInterface</
                hardwareInterface> <!-- aqui poner interfaz con odrive -->
            <mechanicalReduction>7</mechanicalReduction>
        </actuator>
    </transmission>

    <transmission name="trans_codo">
        <type>transmission_interface/SimpleTransmission</type>
        <joint name="codo">
            <hardwareInterface>hardware_interface/EffortJointInterface</
                hardwareInterface> <!-- aqui poner interfaz con odrive -->
        </joint>
        <actuator name="codo_motor">
            <hardwareInterface>hardware_interface/EffortJointInterface</
                hardwareInterface> <!-- aqui poner interfaz con odrive -->
            <mechanicalReduction>7</mechanicalReduction>
        </actuator>
    </transmission>
    <transmission name="trans_pris">
        <type>transmission_interface/SimpleTransmission</type>
        <joint name="pris">
            <hardwareInterface>hardware_interface/EffortJointInterface</
                hardwareInterface> <!-- aqui poner interfaz con odrive -->
        </joint>
        <actuator name="pris_motor">
            <hardwareInterface>hardware_interface/EffortJointInterface</
                hardwareInterface> <!-- aqui poner interfaz con odrive -->
```

```
            <mechanicalReduction>392.5</mechanicalReduction>
        </actuator>
    </transmission>
    <gazebo>
        <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
            <robotNamespace>nelen</robotNamespace>
        </plugin>
    </gazebo>

</robot>
```

codigo/nelen.urdf

Ademas se incluye el archivo de configuración de controladores `ros_controllers.yaml`.

```
# MoveIt-specific simulation settings
moveit_sim_hw_interface:
  joint_model_group: controllers_initial_group_
  joint_model_group_pose: controllers_initial_pose_
# Settings for ros_control control loop
generic_hw_control_loop:
  loop_hz: 300
  cycle_time_error_threshold: 0.01
# Settings for ros_control hardware interface
hardware_interface:
  joints:
    - hombro
    - codo
    - pris
  sim_control_mode: 1 # 0: position, 1: velocity
# Publish all joint states
# Creates the /joint_states topic necessary in ROS

controller_list:
  []
nelen:
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 250
  codo_cont:
    type: effort_controllers/JointPositionController
    joint: codo
    pid: {p: 100.0, i: 0.01, d: 10.0}
  hombro_control:
    type: effort_controllers/JointPositionController
    joint: hombro
    pid: {p: 1000000.0, i: 0.01, d: 10.0}
  prism_control:
    type: effort_controllers/JointPositionController
    joint: pris
    pid: {p: 100.0, i: 0.01, d: 10.0}
```

codigo/ros_controllers.yaml

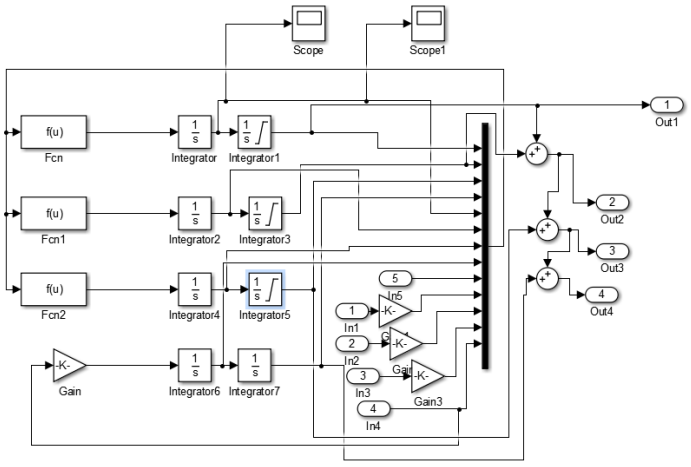


Figura 8.12: Interior del bloque simulink del modelo del SCARA

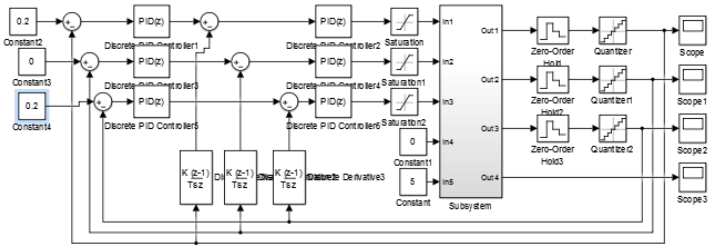


Figura 8.13: Sistema de simulink con controladores PID para control del modelo del SCARA

B . Simulaciones modelo fenomenológico

Para generar las simulaciones se utilizó Matlab Simulink. En primer lugar es necesario pasar el modelo mostrado en 2.39 a un bloque simulink, para esto se calcula una expresión para $\ddot{alpha} = \dot{x} = fx$ y se ocupa el bloque *user-difiined-function*, la salida de esta función es integrada dos veces, obteniendose $\dot{\alpha}$ y α. En la Figura 8.12. Notese que el segundo integrador cuenta con un saturador, estos son los limites físicos del brazo. Además, tiene 5 entradas, las 4 que tienen una ganancia son la corriente de los motores respectivos y la ganancia es la transformación de corriente a torque según la expresión (3.3).

Luego, este sistema se agrupa en un subsistema y se le incluyen los lazos de control. Esto se puede ver en la Figura 8.13. En la parte derecha, a la salida del modelo se incluyeron dos bloques para discretizar la salida, tanto temporalmente como en su valor, esto se hace para simular los efectos de la lectura que puede entregar el Encoder. El sistema de control consiste en lazos anidados, el lazo central corresponde a control de velocidad y el lazo mas externo corresponde a control de la posición. Este escenario es realista pues es como funciona el driver de motores *Odrive* según se discute en [56]. Notese que no se ha incluido un controlador para el efector final, esto pues en la punta tiene un motor stepper y este se controla con lazo abierto.

Luego es necesario obtener las ganancias de cada controlador, para hacer esto, se elije un punto de operación

Tabla 8.1: Constantes del PID obtenidas para el alzo de velocidad

Articulación	P	I
hombro	497.1	1132.7
codo	133.5	1469
Z	0.616	6.785

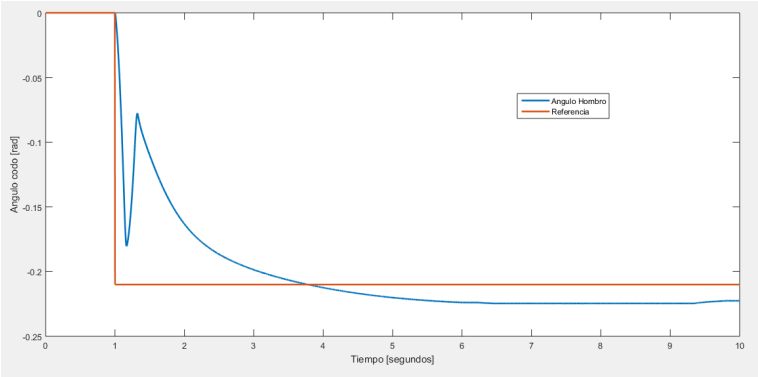


Figura 8.14: Reacción del codo para un cambio de posición de 0 a 0.2 radianes

y se linealiza en torno a el. Esto es directo en sistemas que son SISO (*single input single output*), pero en sistemas MIMO (*multiple input multiple output*) como este este procedimiento no puede hacerse directo, por el efecto que tiene en cada articulación el movimiento de las otras. Para solucionar esto, se utiliza la ecuación de cada una de los ejes y se linealiza respecto a dicho eje asumiendo que el resto de las entradas y estado, asociados a los otros ejes son 0. Luego, se utiliza el asistente para afinación de controladores de simulink y se obtienen los valores de la Tabla 8.1.

Luego se prueban estas ganancias en el modelo completo. Si bien cada uno de los controladores encontrados cumple con los parámetro de diseño, al ponerlos todos juntos el sistema se hace inestable. En la Figura 8.14. Este resultado puede mejorar significativamente si se procura que el lazo de control interno sea mas rápido que el externo, sin embargo con este modelo sucede que el lazo de control externo oscila descontroladamente cuando se busca subir su tiempo de respuesta.

También se probó el enfoque de [22], en el cual se utiliza un solo lazo de control para controlar cada eje, como se muestra en la Figura 8.15. Para elegir los parámetros de los controladores se siguió el mismo procedimiento, se linealizó la el modelo para el punto de operación estático en 0 para todos sus ejes. Las ganancias utilizadas se encuentran en la Tabla 8.2.

.

Tabla 8.2: Ganancias para sistema de control de un solo lazo

Articulación	P	I	D
hombro	149.2	28.66	97.42
codo	15.55	4.96	12.18
Z	6.28e-3	5.79e-3	1.7e-2

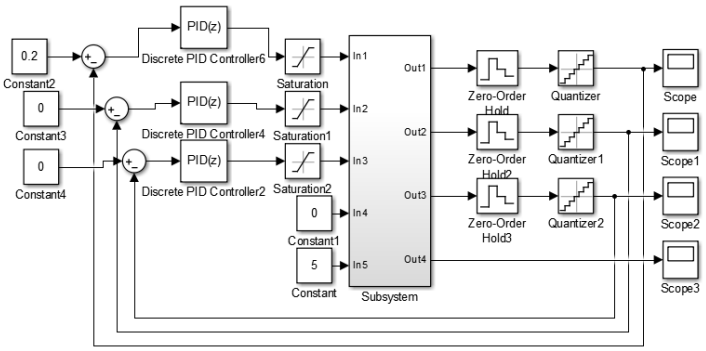


Figura 8.15: Modelo de control con solo un lazo de control.

Con este sistema si se obtienen resultados mas estables pero aun no es posible seguir una trayectoria deseada. A la fecha de esta entrega no se ha podido mejorar el desempeño del controlador ni con la topología ni con las constantes. De todas formas es muy probable a que el modelo tenga algún error en su implementación pues debería ser mucho mas estable.

Si bien se invirtió mucho tiempo en desarrollar un modelo que considerara los aspectos mas principales del robot, este proceso ha sido tedioso y difícilmente sirva para la afinación de los controladores reales. Esto principalmente porque según como esté implementado el algoritmo de control las constantes que lo describen cambian drásticamente. Por ejemplo, si la referencia se elige como una cantidad de pasos entonces al cambiar la resolución de los encoders, cambiará el modelo totalmente. Por esta razón, si bien este es uno de los objetivos específicos, se priorizó la evaluación de componentes por sobre el modelamiento y por eso los pobres resultados, sin embargo estos se mejoraran en la entrega final.

C . Protocolos de comunicación

En esta subsección se discutirán los protocolos de comunicación con dispositivos perifericos que se utilizaran y como se pueden implementar.

C .1. I2C

La información de este protocolo se basa en el sitio web 2c.info [57] que vela por entregar información detallada sobre el funcionamiento de este protocolo. I2c fue creado a finales de la decada de los 70 por Phillips. Destaca por utilizar solo 2 cables de comunicación y poder conectar múltiples dispositivos. Hoy en día la mayoría de los microcontroladores cuenta con este protocolo.

Los dos cables corresponden a SDA y SCL los cuales corresponden a *Serial data* y *Serial clock*. Ambos cables se conectan al voltaje de alimentación mediante resistencias. SCL es la señal de reloj, envía una señal periódica cuadrada que sirve para determinar el inicio y final de cada bit. SDA es el canal mediante

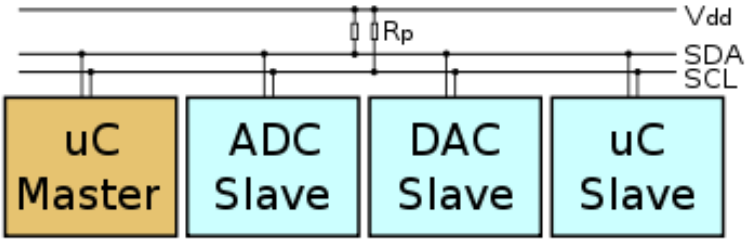


Figura 8.16: Bus de I^2C que incluye un maestro, un DAC, unADC y un microcontrolador como esclavos. SDA corresponde a *Serial data* y CLK corresponde a *Serial clock*

el cual se envía y recibe información, es *Logic-Low* lo que significa que al bajar SDA al valor bajo es el 1 lógico y al subirlo a Vcc es el 0 lógico.

En este tipo de protocolo, típicamente, cada dispositivo esclavo tiene una dirección única, la cual corresponde a 7 bits, por esta razón se pueden conectar directamente a este bus hasta $2^7 = 128$ dispositivos. Sin embargo se debe cumplir la condición de que la capacitancia de la linea no supere los 400 $[pf]$. Esta ultima condición difícilmente se cumple cuando se trabaja a pequeñas distancias (6m) y con pocos dispositivos. En [58] se presentan alternativas para disminuir la capacitancia de un cable, dentro de las que se encuentran; 1- utilizar un conductor mas delgado, 2-Aumentar el espesor de la pared aislante del cable, 3- utilizar un cable con con aislante de menor constante eléctrica relativa.

En su funcionamiento normal, tanto SDA como SCL se encuentran en estado alto (conectadas a Vcc). Luego, para iniciar la comunicación, el dispositivo maestro envía una *secuencia de inicio*, la cual corresponde a bajar a el voltaje de la linea SDA y luego la del reloj, continuando esta ultima con su comportamiento periódico durante lo que dure la comunicación. Para finalizar la comunicación se envía una *Señal de parada*, la cual corresponde a subir la linea SDA mientras SCL esta arriba. Durante su funcionamiento SDA cambia solo mientras SCL se encuentra abajo. Una vez iniciada la comunicación, el dispositivo maestro envía los siete primeros bits, los cuales corresponden a la dirección del esclavo al que se quiere acceder, el 8vo bit indica si se leerá o si se escribirá en el dispositivo. Posteriormente se envían 8 bits con la instrucción al dispositivo, donde el 8vo bit corresponde al igual que antes a la especificación si se escribirá o si se leerá. Por último, el dispositivo responde con la información solicitada. Cada 8 bits, el 9no bit es desde el esclavo al maestro y corresponde a una señal de *acknoledgment* o ACK, este bit es una confirmación de que el esclavo entendió el byte enviado, es 0 en caso de que hubo algún error en la comunicación y 1 si la comunicación fue exitosa. En la Figura 8.17 se muestra un diagrama de comunicación utilizando este protocolo

C .2. SPI

La información de esta subsección se obtiene de la norma de SPI de motorola [60], es en esta compañía en la que en la época

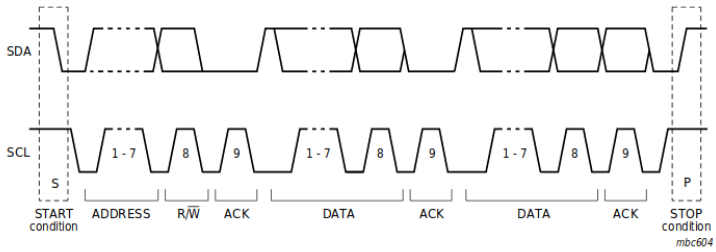


Figura 8.17: Transferencia de información mediante I^2C. Copiada desde [59]

del 1980 se desarrolló este protocolo.

Serial Peripheral Interface (SPI) es un protcolo de comunicación de corta distancia pensado principalmente para comunicar dispositivos electrónicos en placas de circuitos. Es un sistema *full-duplex*, uno en que el los dos dispositivos comunicados pueden enviar información simultáneamente. Es del tipo Master Slave, en la que un solo sistema (Master) es quien organiza las comunicaciones para todos los demás dispositivos (Slaves). Las 4 señales utilizadas para el funcionamiento de este protocolo y sus funciones son:

- *Serial Clock* (SCLK): Esta es una señal periódica utilizada para sincronizar la comunicación, además se utiliza para setear la velocidad de comunicación
- *Master Input Slave Output*: Es la salida del dispositivo maestro y la entrada del dispositivo
- *Master Output Slave Input* (MOSI): Es la salida del dispositivo esclavo y la entrada del dispositivo maestro
- *Slave Select* (SS) es una señal de on/off para especificar que dispositivo es al que se envía información. Nótese que el uso de esta señal de baja velocidad hace que no sea necesario dar direcciones a los dispositivos como en el caso de I^2C. Solo un dispositivo puede estar encendido a la vez.

En la Figura 8.18 se muestra una configuración de un sistema de comunicación SPI y 3 esclavos.

Por último, para utilizar este protocolo es necesario elegir correctamente los parámetros de coordinación de las señales MISO y MOSI respecto al reloj (CLK), esto se conoce como los modos de funcionamiento y son indicados en las hojas de datos de los dispositivos esclavos, por ende, la selección de estos parámetros depende del dispositivo esclavo y no del maestro. Los parámetros a modificar con *Clock Polarity* (CPOL) y *Clock Phase* (CPHA). CPOL es la polaridad del reloj, esto se refiere a si el reloj está normalmente en un nivel de voltaje bajo y cada ciclo corresponde a una subida de voltaje (CPOL=0) o si el reloj está normalmente en un nivel de voltaje alto y cada ciclo del reloj corresponde a una bajada (CPOL=1). Por su parte CPHA se refiere al desface del reloj respecto a las señales de transmisión (MOSI y MISO), si el dispositivo conectado necesita esperar solo la subida del reloj para empezar la transmisión entonces es CPHA=0, en la duración de un bit la primera mitad del tiempo el reloj está apagado y la otra mitad prendido. Si el dispositivo necesita esperar una subida y una bajada de reloj para empezar la comunicación

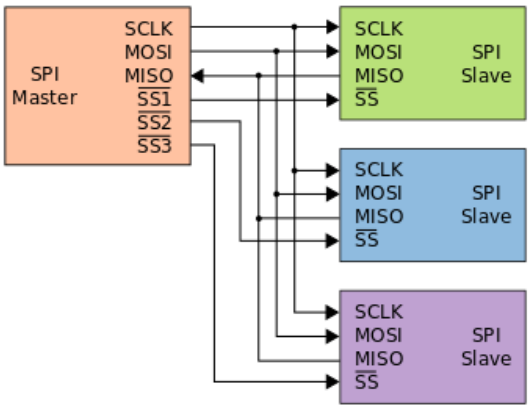


Figura 8.18: Sistema de comunicación SPI con 3 esclavos y un maestro. Imagen copiada desde [61]

es CPHA=1, en la duración de un bit la primera mitad el reloj está apagad y la otra mitad prendido. En la Figura 8.19 se ejemplifica los efectos de las variables CPOL y CPHA. Además en la tabla 8.3 se muestra el resumen de los modos de funcionamiento, dado que hay 4 combinaciones posibles entre CPOL y CPHA se definen 4 modos según muestra la tabla.

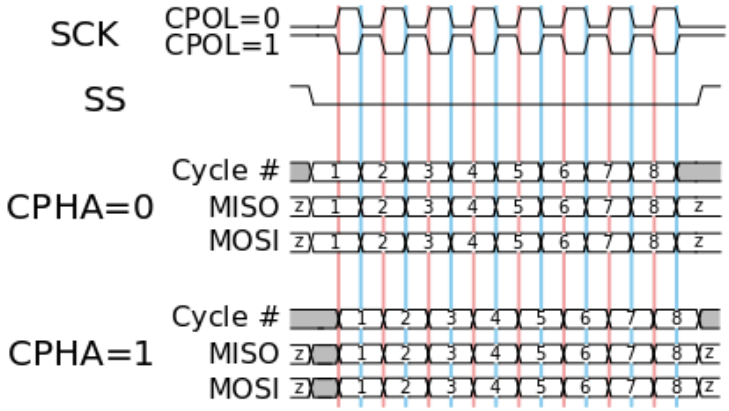


Figura 8.19: Funcionamiento del protocolo SPI para distintos parámetros de configuración de reloj. Imagen copiada desde [61]

Tabla 8.3: Modos de funcionamiento SPI

Modo	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Bibliografía

[1] P. Ovalle, «Chile lidera desigualdad en países de la OCDE», *El Dínamo*, 8 de jul. de 2017.

[2] J. Yutronic, «Los desafíos de la manufactura avanzada en Chile», *Electroindustria*, dic. de 2017.

[3] M. Meier, *Robots*, Disponible en https://mkmra2.blogspot.com/2016/01/robots.html, 2016.

[4] N. Smolenski, *Kinematic diagram of SCARA robot*, Disponible en https://commons.wikimedia.org/wiki/File:SCARA_configuration.png, 2007.

[5] *How to know when a SCARA Robot is the right choice for your application*, https://www.fanuc.eu/de/en/robots/robot-filter-page/scara-series/selection-support.

[6] C. Tech, *When to use a SCARA Robot*, Disponible en https://cyan-tec.com/laser-systems/when-to-use-a-scara-robot, 2019.

[7] E. Tsirogiannis, «Reverse Engineering, Redesign and Topology Optimization for Additive Manufacturing of an Industrial Robot Arm Link», Tesis doct., oct. de 2017.

[8] M. Shariatee, A. Akbarzadeh, A. Mousavi y S. Alimardani, «Design of an economical SCARA robot for industrial applications», en *2014 Second RSI/ISM International Conference on Robotics and Mechatronics (ICRoM)*, IEEE, 2014, págs. 534-539.

[9] R. C. Dorf y S. Y. Nof, *Concise international encyclopedia of robotics: applications and automation*. John Wiley & Sons, Inc., 1990.

[10] M. T. Das y L. C. Dülger, «Mathematical modelling, simulation and experimental verification of a scara robot», *Simulation Modelling Practice and Theory*, vol. 13, n.º 3, págs. 257-271, 2005.

[11] M. A. Al-Khedher y M. S. Alshamasin, «SCARA robot control using neural networks», en *2012 4th International Conference on Intelligent and Advanced Systems (ICIAS2012)*, vol. 1, 2012, págs. 126-130. DOI: 10.1109/ICIAS.2012.6306173.

[12] S. Suvilath, K. Khongsomboun, T. Benjanarasuth y N. Komine, «IMC-based PID controllers design for a two-links SCARA robot», en *TENCON 2011 - 2011 IEEE Region 10 Conference*, 2011, págs. 1030-1034. DOI: 10.1109/TENCON.2011.6129267.

[13] M. S. Alshamasin, F. Ionescu y R. T. Al-Kasasbeh, «Modelling and simulation of a SCARA robot using solid dynamics and verification by MATLAB/Simulink», *International Journal of Modelling, Identification and Control*, vol. 15, n.º 1, págs. 28-38, 2011.

[14] C. Urrea y J. Kern, «Modeling, simulation and control of a redundant SCARA-type manipulator robot», *International Journal of Advanced Robotic Systems*, vol. 9, n.º 2, pág. 58, 2012.

[15] M. H. Liyanage, N. Krouglicof y R. Gosine, «Development and testing of a novel high speed SCARA type manipulator for robotic applications», en *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, págs. 3236-3242.

[16] *Lagrange equations (in mechanics)*, Disponible en https://www.encyclopediaofmath.org/index.php/Lagrange_equations_(in_mechanics), 2011.

[17] R. M. Murray, *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[18] N. Vuong, A. H. Marcelo Jr, Y. Li y S. Lim, «Improved dynamic identification of robotic manipulators in the linear region of dynamic friction», *IFAC Proceedings Volumes*, vol. 42, n.º 16, págs. 167-172, 2009.

[19] V. De-León-Gómez, V. Santibañez y J. Moreno-Valenzuela, «A procedure to find equivalences among dynamic models of planar biped robots», *Simulation Modelling Practice and Theory*, vol. 75, págs. 48-66, 2017.

[20] J. K. Ioan Sucan, «Xml robot description format (urdf)», *ROS wiki*, 2019.

[21] T. B. Willy K. Wojsznis, «Evolving PID tuning rules», *Control engineering*, 2013.

[22] B. Ibrahim y A. M. Zargoun, «Modelling and Control of SCARA manipulator», *Procedia Computer Science*, vol. 42, págs. 106-113, 2014.

[23] M. A. Al-Khedher y M. S. Alshamasin, «SCARA robot control using neural networks», en *2012 4th International Conference on Intelligent and Advanced Systems (ICIAS2012)*, IEEE, vol. 1, 2012, págs. 126-130.

[24] S. M. Swadi, A. I. Majeed, A. A. Ugla y col., «Design and Simulation of Robotic Arm PD Controller Based on PSO», *University of Thi-Qar Journal for Engineering Sciences*, vol. 10, n.º 1, págs. 18-24, 2019.

[25] J. Kennedy y R. Eberhart, «Particle swarm optimization», en *Proceedings of ICNN'95-International Conference on Neural Networks*, IEEE, vol. 4, 1995, págs. 1942-1948.

[26] P. Besset y R. Béarée, «FIR filter-based online jerk-constrained trajectory generation», *Control Engineering Practice*, vol. 66, págs. 169-180, 2017.

[27] Y. Fang, J. Hu, W. Liu, Q. Shao, J. Qi e Y. Peng, «Smooth and time-optimal S-curve trajectory planning for automated robots and machines», *Mechanism and Machine Theory*, vol. 137, págs. 127-153, 2019.

[28] S. R. Jape y A. Thosar, «Comparison of electric motors for electric vehicle application», *IJRET: International Journal of Research in Engineering and Technology*, vol. 6, n.º 09, pág. 2, 2017.

[29] P. Aree, «Analytical approach to determine speed-torque curve of induction motor from manufacturer data», *Procedia Computer Science*, vol. 86, págs. 293-296, 2016.

[30] P. Yedamale, «Brushless DC (BLDC) motor fundamentals», *Microchip Technology Inc*, vol. 20, págs. 3-15, 2003.

[31] R. Parson, *How to estimate the torque of a BLDC (PMSM) electric motor using only its Kv and current draw*, Disponible en https://things-in-motion.blogspot.com/, 2018.

[32] ——, *Understanding BLDC (PMSM) electric motors: Base speed, no load speed and torque vs speed*, Disponible en https://things-in-motion.blogspot.com/, 2019.

[33] N. Kau, A. Schultz, N. Ferrante y P. Slade, «Stanford doggo: An open-source, quasi-direct-drive quadruped», *arXiv preprint arXiv:1905.04254*, 2019.

[34] B. Earl, *All About Stepper Motors*, Disponible en https://learn.adafruit.com/all-about-stepper-motors, 2014.

[35] Dejan, *How a stepper motor works*, Disponible en https://howtomechatronics.com/how-it-works/electrical-engineering/stepper-motor/, 2015.

[36] G. Levine, *Open Torque*, Disponible en https://www.hackaday.io/project/159404-opentorque-actuator.

[37] Odrive, *Encoder Guide*, Disponible en https://docs.odriverobotics.com/.

[38] *14-Bit On-Axis Magnetic Rotary Position Sensor with 12-Bit Decimal and Binary Incremental Pulse Count for 28krpm High Speed Capability*, AS5047p, Rev. 1.01, AMS, abr. de 2016.

[39] T. Labs, *Calibration Routine*, Disponible en https://hackaday.io/project/11224-mechaduino/log/38559-calibration-routine.

[40] *Datasheet Single and Dual Zone Infra Red Thermometer in TO-39*, MLX90614 family, Rev 13, Melexis, sep. de 2019.

[41] Omzlo, *Adding a DC power jack to the Raspberry-Pi*, Disponible en https://omzlo.com/articles/adding-a-dc-power-jack-to-the-raspberry-pi.

[42] R. Parson, *How to select the right power source for a hobby BLDC (PMSM) motor*, Disponible en https://things-in-motion.blogspot.com/, 2018.

[43] Milliways, *What are the Electrical Specifications of GPIO pins?*, Disponible en https://raspberrypi.stackexchange.com/questions/60218/what-are-the-electrical-specifications-of-gpio-pins, 2017.

[44] *DIP 4pin General PurposePhotocoupler*, PC817X Series, Rev 1, Sharp, sep. de 2003.

[45] P. Gray, P. Hurst, S. Lewis y R. Meyer, *Analysis and Design of Analog Integrated Circuits*. Wiley, 2001, ISBN: 9780471321682.

[46] T. Thiele, *Understanding Electrical Wire Labeling*, Disponible en https://www.thespruce.com/understanding-electrical-wire-lettering-1152874, 2019.

[47] Powerstream, *Wire Gauge and Current Limits Including Skin Depth and Strength*, Disponible en https://www.powerstream.com/Wire_Size.htm.

[48] E. Toolbox, *AWG Wire Gauges Current Ratings*, Disponible en https://www.engineeringtoolbox.com/wire-gauges-d_419.html.

[49] O. cable, *NEC Ampacity Data*, Disponible en https://www.omnicable.com/technical-resources/nec-ampacity-data.

[50] A. U. Tech), *Wires, connectors and current - what you need to know as a drone builder*, Disponible en https://www.dronetrest.com/t/wires-connectors-and-current-what-you-need-to-know-as-a-drone-builder/1342, Drone Trest.

[51] NEPSI, *Short-time current rating of cables, conductors, and bus bar*, Disponible en http://nepsi.com/resources/calculators/short-time-current-rating-of-conductor.htm.

[52] *How to install ROS on raspberry pi 3*, Disponible en https://www.intorobotics.com/how-to-install-ros-kinetic-on-raspberry-pi-3-ubuntu-mate/, 2017.

[53] *Arduino - Leer multiples MLX90614*, Disponible en https://ofimatica-programacion.blogspot.com/2017/05/arduino-mega-leer-dos-sensores-mlx90614.html.

[54] *16-Bit I/O Expander with Serial Interface*, MCP23017, Rev. C, Microchip, jul. de 2016.

[55] *CircuitPython module for the Melexis MLX90614*, Disponible en https://circuitpython.readthedocs.io/projects/mlx90614/en/latest/, 2018.

[56] Odrive, *Control*, Disponible en https://docs.odriverobotics.com/control, 2016.

[57] *I2C Info – I2C Bus, Interface and Protocol*, Disponible en https://i2c.info/.

[58] *Why is Cable Capacitance Important for Electronic Applications?*, Disponible en https://www.quabbin.com/tech-briefs/why-cable-capacitance-important-electronic-applications.

[59] N. Semiconductors, «UM10204 I2C-bus specification and user manual», *User Manual*, vol. 4, 2014.

[60] S. B. Guide, «V03. 06.: Motorola», *Inc, February*, 2003.

[61] C. M. Burnett, *Serial Peripheral Interface (SPI)*, Disponible en https://en.wikipedia.org/wiki/Serial_Peripheral_Interface.