Universidad de la República
Facultad de Ingeniería

PicassoBotZ: Un Brazo Robótico que Realiza Retratos

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Juan Pablo de Souza, Diego Pereyra, Santiago Suárez

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutor

Pablo Monzón	Universidad de la República
Pablo Musé	Universidad de la República
Juan Pablo Oliver	Universidad de la República

Tribunal

Agustín Rodríguez	Universidad de la República
Ignacio Ramírez	Universidad de la República
Francisco Puignau	Universidad de la República

Montevideo
martes 4 octubre, 2022

PicassoBotZ: Un Brazo Robótico que Realiza Retratos, Juan Pablo de Souza, Diego Pereyra, Santiago Suárez.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 162 páginas.
Compilada el martes 4 octubre, 2022.
http://iie.fing.edu.uy/

Un pintor es un hombre que pinta lo que vende. Un artista, en cambio, es un hombre que vende lo que pinta.

Pablo Picasso

La ciencia más útil es aquella cuyo fruto es el más comunicable.

Leonardo Da Vinci

It's important to draw wisdom from many different places, if we take it from only one place it becomes rigid and stale.

Uncle Iroh

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

En primer lugar, queremos agradecer a nuestras familias y amigos, este trabajo no hubiese sido posible sin su apoyo incondicional, tanto durante el proyecto como durante toda la carrera. Ellos son los que a lo largo de este año nos han soportado en diferentes momentos, brindándonos compañía, escucha, distracción y confianza en nuestro trabajo. Siempre dándonos para adelante en todo lo que íbamos haciendo.

También queremos agradecer a todas aquellas personas con las que tuvimos el gusto de recorrer parte o la totalidad de la carrera. Con ellos hemos compartido una gran cantidad de tiempo, comidas, intereses, estudio, sufrimientos y apoyos mutuos. Por aguantarnos y por aguantarlos, es por lo que podemos estar nosotros presentando este trabajo en este momento.

A todos los maestros, profesores y docentes que se preocuparon en nuestra formación, ya sean en el aula o en la vida. Gracias a ellos es que comenzamos a recorrer este camino hace cinco años, y hoy podemos decir que su esfuerzo con nosotros ha dado sus frutos.

En particular, agradecemos a nuestros tutores, Pablo Monzón, Pablo Musé y Juan Pablo Oliver, quienes nos han guiado durante este último año para que este proyecto sea posible. Por su infinidad de consejos, compañía y apoyo es que logramos solucionar problemas que parecían imposibles fracciones de reunión.

También agradecer a Rafael Grompone por su atento oído, su interés y su aporte fundamental con el algoritmo de la detección de ridges. Sin ello no hubiese sido posible la incorporación de nuevos estilos de dibujo al sistema.

A Martín Etchart, por siempre haber mandado cualquier trabajo relacionado que encontraba y pudiera servir. En particular, los últimos dos estilos surgieron gracias a una de sus sugerencias.

Por último, queremos agradecer a los integrantes de los proyectos PicassoBot y PARRA, que comenzaron con esta antorcha de construir un brazo que dibuja. Por estar dispuestos para tener charlas formales o informales, por explicarnos tan pacientemente sus proyectos y las ideas en ellos. Por servir como inspiración para cargar la antorcha aunque sea un poco más.

A todos los aquí mencionados, y a todos quienes tenemos presentes en nuestros corazones, solo tenemos para decirles:

¡Gracias!

Esta página ha sido intencionalmente dejada en blanco.

A nuestros amigos y familias

Esta página ha sido intencionalmente dejada en blanco.

Resumen

En este documento se describe el trabajo realizado en el proyecto **Picasso-BotZ**, en el cual se propone la construcción y programación de un brazo robótico capaz de realizar retratos de personas, imitando el dibujo humano. El objetivo principal del proyecto es que el brazo robótico sea capaz de realizar los retratos en tiempo real, comandado por un dispositivo capaz de realizar procesamiento de señales y capaz de obtener la foto a dibujar en el acto.

El brazo está compuesto por cuatro motores restringidos a moverse solamente en un plano paralelo a la hoja en la cual se quiera dibujar. Tres de estos motores representan las articulaciones del brazo humano (hombro, codo y muñeca). El cuarto motor sirve para dar la posibilidad de realizar movimientos del brazo con el lápiz levantado. Además, estos motores se encuentran dispuestos sobre una instalación artística, la cual provee portabilidad y comodidad para utilizar el brazo.

Dentro de la instalación, se encuentra la electrónica suficiente para que el dispositivo que comanda al brazo se comunique con el mismo. Esta electrónica está conformada por un Arduino y un circuito impreso en PCB. Además, se tienen en la instalación dos ventiladores para mejorar la disipación de calor de los motores.

El dispositivo que comanda al brazo se puede encontrar tanto fuera como dentro de la instalación, dependiendo de cuál dispositivo de comando se elija utilizar. Este dispositivo obtiene la imagen a dibujar a través de una cámara web, realiza un procesamiento de imagen indicado por el usuario y planifica los comandos a enviar al brazo. También se encarga de controlar el flujo de comandos que se envían hacia el brazo, comunicándose de forma apropiada con el Arduino.

En este proyecto se logra resolver problemas de una amplia gama de áreas del conocimiento. Se solventa la comunicación entre una dispositivo de alta capacidad de cómputo con los motores que conforman el brazo. Esto se realiza en la implementación del firmware que se ejecuta desde el Arduino. Además, se logra, mediante técnicas de robótica moderna, la planificación adecuada de comandos a realizar por el brazo, dado un conjunto ordenado de curvas. También se logra obtener curvas representativas de una imagen de una persona, dotando a estas curvas de una naturalidad parecida a la de los humanos, mediante novedosas técnicas de visión artificial. Por último, se logra realizar una instalación que permite la portabilidad del sistema y un acondicionamiento adecuado.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Capítulo 1

Introducción

El proyecto **PicassoBotZ** propone una combinación entre una creación tecnológica y una obra artística, de forma de obtener un dibujo de carácter natural hecho por un robot. Se plantea un sistema completo capaz de adquirir una imagen de una persona y hacer un retrato de la misma con un brazo robótico. También se busca que la progresión en la ejecución del dibujo se asemeje a la forma en que los humanos dibujan.

1.1. Objetivos

Este proyecto surge a partir de dos proyectos previos de fin de carrera de Ingeniera Eléctrica, que trabajaron en distintas partes del sistema planteado y que se pretende unir. El primer proyecto es PicassoBot [1], del cual hereda el nombre este proyecto. Se trata de un robot que dibuja formas geométricas simples enviadas desde una computadora de manera cableada o inalámbrica. El otro proyecto es PARRA [2], el cual consiste en un software que dado una imagen de un retrato de una persona, extraiga un conjunto de curvas que sean una representación artística de la imagen. En este último proyecto también trabajaron en profundidad sobre cómo las curvas encontradas podrían ser trazadas por un brazo robótico. Ambos proyectos fueron pensados para luego conformar un único sistema que funcione de punta a punta, extrayendo curvas a partir de una imagen y comandando un robot para que las dibuje. La idea de este proyecto es utilizar el trabajo de ambos antecedentes y lograr tal objetivo.

Lo primero que se aborda en este proyecto es evaluar qué de lo implementado por los antecesores se puede utilizar y qué modificaciones hay que hacer para lograr compatibilidad entre los dos sistemas ya creados. Un vez que se tiene el sistema funcionando de punta a punta, se empieza a construir sobre lo que se tiene, mejorando o reformulando las distintas piezas para optimizar el sistema. Se verá que, a lo largo del proyecto, en la mayoría de las partes fueron introducidas modificaciones o mejoras. En algunos casos se replantea la metodología utilizada, mientras que en otros simplemente se incorporan mejoras sobre la base que se tiene.

El alcance de este proyecto incluye desde mecanismos de control que permiten al brazo seguir curvas específicas, hasta aspectos de procesamiento de imágenes para lograr representaciones artísticas. También contempla la construcción de un sistema final portable, de forma de crear una plataforma física, e independizarse de la computadora para el procesamiento de imágenes.

Se destaca que la idea del proyecto es un brazo robótico que simule dibujos de una persona a mano alzada. Por lo tanto, no se trabaja en profundidad en la precisión y detalle de los trazos que puede esperarse de un robot. Por el contrario, se limita incluso los movimientos que el brazo puede hacer para poder mimetizarse lo más posible con un brazo humano, en este caso izquierdo. La idea es dar un enfoque artístico al proyecto y dotar de naturalidad su obra, alejándose de la exactitud y el determinismo de una máquina.

1.2. Contexto y antecedentes

1.2.1. El arte y la robótica

La palabra *robot* proviene de la palabra checa *robota* que significa "trabajo forzado". Este término se utilizó por primera vez en el año 1920, en una obra de teatro llamada RUR (Rossum's Universal Robots) de Karel Capek. Sin embargo, el origen de la robótica se podría remontar hasta la época de Aristóteles y sus ideas sobre "herramientas automatizadas", pasando por Leonardo Da Vinci o Isaac Asimov, hasta Henry Ford.

Uno de los robots más famosos a lo largo de la historia ha sido Asimo. Este robot fue creado por Honda en el año 2000 y es un robot humanoide capaz de caminar como una persona y reconocer caras y voces. Además, Asimo es capaz de identificar objetos en movimiento, entre otras cosas. Otro robot conocido internacionalmente es el robot humanoide Nao como se puede ver en la figura 1.1a. Este robot es utilizado normalmente en competencias de fútbol de la Robocup. Nao está hecho para que ayude a desarrollar nuevos robots inspirados en él, es por eso que en muchas partes del mundo, las instituciones educativas cuentan con estos robots. Otro robot reconocido mundialmente es Spot de Boston Dynamics que se muestra en la figura 1.1b. Spot es ampliamente utilizado por diferentes empresas constructoras de forma de minimizar el riesgo de los operarios al momento de relevar un terreno. Este robot es capaz de imitar los movimientos de animales de forma natural. Por último, la robot Sophia, de la empresa Hason Robotis, es de los robots humanoides más famosos, ya que cuenta con un aspecto humano como se ve en la figura 1.1c, que otros robots humanoides no tienen. Esta robot además, tiene reconocimiento facial, puede comunicarse con las personas y tiene la capacidad de aprender.

Dentro del mundo de los robots hay una amplia rama, y que cada vez tiene más fuerza: los robots que dibujan. En ella se pueden ver diferentes estilos de robots y de dibujos creados por los mismos. Un ejemplo de esto es DrawBo. Este robot cuenta con algoritmos de aprendizaje automático que usa para enseñar a dibujar a los niños paso por paso. Para esto solo se necesita una superficie vertical y algo

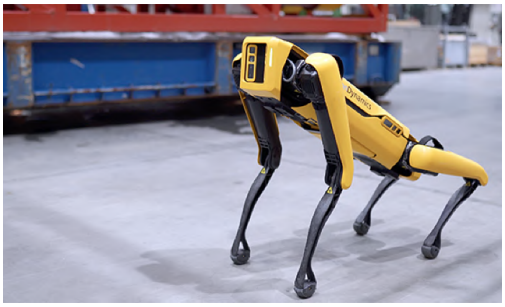
para dibujar. DrawBo tiene forma de mariposa como se ve en la figura 1.2a y utiliza su software para dividir el dibujo en pasos intermedios, de forma que los niños que están viendo el dibujo puedan copiar en sus cuadernos. Otro robot similar a este es es el robot Quincy, que se ve en la figura 1.2b. Este robot también enseña a dibujar, sin embargo este cuenta con un ojo que tiene una cámara y con ella lee diferentes códigos QR donde cada uno conlleva a un dibujo que luego realizará.

Por otra parte, el trabajo de Patrick Tresset es el antecedente principal, ya que fue disparador de los proyectos anteriores y de este. Tresset es un artista francés especializado en robótica de la Universidad de Londres, que exhibe obras de arte teniendo como pieza principal a los robot creados por él. En el transcurso de su carrera, ha trabajado junto a científicos computacionales e informáticos como Oliver Deussen de la Universidad de Konstanz en Alemania.

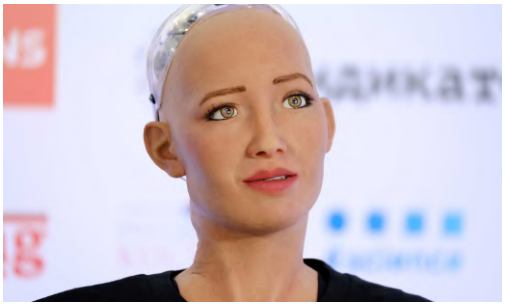
Sus obras introducen sistemas informáticos con el objetivo de alinear un carácter "artístico, expresivo y obsesivo" con el comportamiento de los robots. Sus trabajos incluyen el desarrollo de robots como *Paul the robot* [8] y *e-David* [9], [10] junto con Deussen.

El hardware de Paul the robot contiene un brazo de tres servomotores que se pueden mover para que el cuarto servomotor pueda hacer el dibujo deseado. Las obras de Paul han sido exhibidas en museos y galerías de arte a lo largo del mundo. Por su parte, e-David es un trabajo más avanzado. Este es un brazo con dimensiones mucho mayores y es capaz de agarrar diferentes brochas para trabajar con distintas pinturas de colores. Las obras que este realiza son también de mayor

(a) *Nao* [3].



(b) *Spot* [4].



(c) *Sophia* [5].

Figura 1.1: Diferentes robots famosos que son inspiración en el rubro.

(a) *DrawBo* [6]. (b) *Quincy* [7].

Figura 1.2: Robots existentes en la rama de los robots que dibujan.

tamaño.

(a) *Paul the robot* [8] retratando a Patrick Tresset. (b) Obra de *E-David* [9].

Figura 1.3: Robots dibujantes elaborados por Patrcik Tresset y Oliver Deussen.

1.2.2. Proyectos anteriores

PicassoBot

Este proyecto tuvo como objetivo la construcción y el diseño un brazo electromecánico de cuatro articulaciones compuesto por servomotores, que se inspiró en "Paul the Robot" desarrollado por el artista Patrick Tresset. Trabajaron también en desarrollar un sistema de control de movimiento para los servomotores tomando como entrada un archivo de texto que contenga los comandos a ser ejecutados en el lenguaje gráfico HPGL usado por *plotters* e impresoras.

El desarrollo de este proyecto comprendió principalmente la implementación de un sistema capaz de mapear la trayectoria del lápiz con la ejecución de movimientos coordinados de las cuatro articulaciones. Se procuró también asimilarse a un brazo humano, restringiendo los movimientos posibles del robot. Además de este procesamiento, desarrollaron herramientas visuales de simulación, que fueron útiles para evaluar las trayectorias.

Específicamente, su sistema completo queda conformado por el brazo robótico, el módulo de procesamiento y otro módulo de comunicación entre la computadora y el brazo. La información sobre los trazos que se quiere dibujar se lee y se envía al algoritmo de posicionamiento y control de trayectoria del lápiz, caracterizada por una secuencia de configuraciones de brazo. Esta se transmite por puerto serial hacia un microcontrolador que conforma los paquetes de instrucciones de los servomotores y los envía para su ejecución. Se tiene también una interfaz de usuario en la que se implementa una realimentación visual con una cámara que graba y transmite el estado del dibujo en tiempo real.

PARRA

El objetivo de este proyecto de fin de carrera fue generar algoritmos de procesamiento de imágenes capaces de transformar una fotografía en una imagen, con características similares a las de un dibujo hecho por una persona. El propósito es luego traducir la información generada en instrucciones que puedan ser interpretadas por un simulador del brazo realizado. Adicionalmente, se logró dotar de cierto carácter humano al proceso de ejecución de la obra, imponiendo las restricciones físicas propias del movimiento de un brazo humano, y considerando los procesos que ocurren usualmente al dibujar del natural. Esto lo lograron inspirándose en técnicas plásticas como la aplicación de líneas en distintos sentidos de forma de generar diferentes efectos visuales, y a su vez, determinando un orden para el trazado de las curvas. Además añadieron imperfecciones y aleatoriedades con el fin de alejar la estética resultante de la precisión robótica.

El alcance del proyecto abarcó principalmente el procesamiento de imágenes, y la traducción de sus resultados a los movimientos del brazo. No se contempló la construcción de un brazo físico ni la ejecución de los trazos resultantes de su trabajo. Se asumió simplemente la existencia de un brazo robótico funcional (el PicassoBot) y se trabajó sobre una capa superior para que ese brazo pudiera dibujar. Sí trabajaron con simulaciones del brazo para entender a grandes rasgos cómo

serían ejecutados los dibujos y cuáles serían sus resultados.

Su sistema toma como entrada una imagen y devuelve comandos o movimientos que un brazo robótico debería realizar para trazar un conjunto de curvas que son una representación artística de la imagen. Para ello se aplicaron diferentes técnicas de procesamiento de imágenes. Por un lado, algoritmos clásicos y conocidos como detección de bordes, pero por otro, también trabajaron en técnicas de sombreado con algoritmos propuestos por ellos mismos. Para la traducción de las curvas a movimientos del brazo, trabajaron con áreas de la robótica como la cinemática inversa (o *inverse kinematics*), y propusieron un algoritmo que determina qué trayectoria debe seguir el brazo para poder dibujar esas curvas. Asimismo, el proyecto PARRA también implementó la alternativa de simular el movimiento que haría el brazo en formato de vídeo.

1.3. Definición del sistema

Al ser uno de los objetivos del proyecto PicassoBotZ la continuación y unión de lo logrado en los proyectos PicassoBot y PARRA, se tuvieron que realizar una serie de cambios a lo que fueron los sistemas heredados de dichos proyectos anteriores. Los cambios en una primera instancia se debieron a un intento de compatibilizar ambos proyectos anteriores. Posteriormente se realizaron aún más cambios con los objetivos de cubrir problemas no contemplados, optimizar procesos realizados de forma ineficiente y agregar prestaciones nuevas al sistema. Con todos estos cambios y algunos más, se llegó a un sistema nuevo y propio del equipo PicassoBotZ.

En realidad, se puede decir que el sistema al que se llegó al final de este nuevo proyecto no es único. A lo largo del desarrollo del proyecto se exploraron diversas alternativas y modificaciones parciales al sistema que hacen que haya, en efecto, varios sistemas distintos que puedan ser denominados "PicassoBotZ". La mayoría de estas alternativas se debieron al desarrollo progresivo de una interfaz física más amigable para un usuario del sistema. De esta forma, se pudo progresar hacia algo similar a lo realizado por Patrick Tresset en sus exposiciones con *Paul the Robot*. Justamente por ser la interfaz física el motivo de estas variaciones, es que la mayoría de las diferencias entre los distintos sistemas posibles son a nivel de hardware. Por contrapartida, el software es esencialmente el mismo para cualquiera de las variaciones posibles al sistema.

Por esta cantidad de combinaciones de hardware posibles, es que es conveniente agruparlo en distintos bloques como se muestra en la figura 1.4. Además esta separación le confiere al sistema un nivel de abstracción. Se puede pensar cada bloque hardware como una caja negra, con determinadas entradas y salidas, independientes de sus componentes.

El **bloque de adquisición**, como su nombre lo dice, es el encargado de obtener la imagen a ser dibujada por el brazo. A efectos prácticos será algún tipo de cámara digital. Este bloque puede estar o no en el sistema final, esto se explicará más adelante en la sección 2.1. Este bloque es comandado desde el bloque de procesamiento, quien envía un pedido de toma de imagen. Luego de que la imagen es tomada, esta es enviada al bloque de procesamiento, donde comenzará a ser

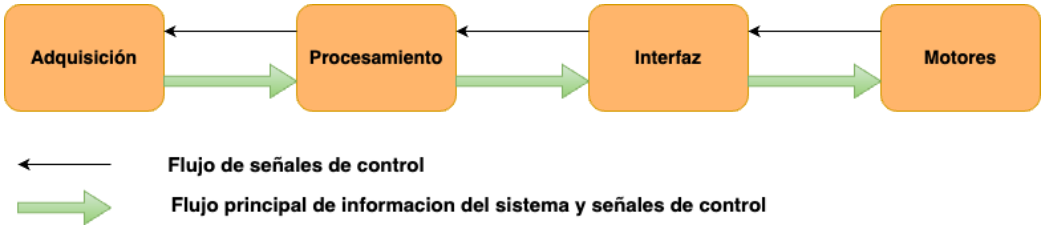


Figura 1.4: Diagrama de bloques del hardware del sistema PicassoBotZ.

utilizada.

El **bloque de procesamiento** es el encargado de orquestar el funcionamiento del sistema en su totalidad. Es quien envía las señales al bloque de adquisición (si existe) para que tome una foto. Es este bloque también quien procesa la imagen obtenida hasta generar comandos a ser enviados hacia los motores. Aquí es donde se encuentra la mayor parte del software del sistema. La salida de este bloque serán comandos interpretables por el bloque de interfaz, quién luego se encargará de que estos sean interpretables por los motores.

Es entonces el **bloque de interfaz** el encargado de traducir los comandos obtenidos por el bloque de procesamiento y enviarlos al brazo. Además es quien controla el flujo de comandos enviados al brazo, pidiendo comandos a necesidad al bloque de procesamiento y enviándolos al bloque de motores. También es capaz de recibir realimentaicón de los brazos y modificar su comportamiento de forma acorde al mismo.

Por último, el **bloque de motores** es quien se encarga efectivamente de ejecutar los comandos obtenidos en el bloque de procesamiento y traducidos por el bloque de interfaz. Este bloque está esencialmente compuesto por los motores elegidos para el brazo y la fuente necesaria para alimentarlos.

En cuanto al software, aunque no cuente con la cantidad de posibles combinaciones del hardware, también es útil separarlo en distintas etapas de un pipeline. Estas se pueden ver en la figura 1.5. El software en el sistema PicassoBotZ se encuentra repartido dentro de dos bloques hardware. Estos son el bloque de procesamiento y el de interfaz.

La **etapa de image processing** es la primera del pipeline y es la encargada de obtener las curvas que se quieren dibujar. Estas curvas se obtienen de forma de ser lo suficientemente representativas de la imagen que se desea dibujar y de cumplir con ciertos criterios estéticos de diseño.

Luego, las curvas pasan a la **etapa de path planning**. En esta etapa las curvas se procesan para obtener comandos a ser ejecutados por el brazo. Estos comandos son calculados de forma que comanden los motores, para que realicen los trazos que correspondan a cada una de las curvas obtenidas en la etapa de image processing.

Posteriormente, está la **etapa de control por software**. Aquí se toman los comandos generados en la etapa de path planning, y se van enviando a la interfaz, para que esta los envíe definitivamente a los motores. En esta etapa se realiza

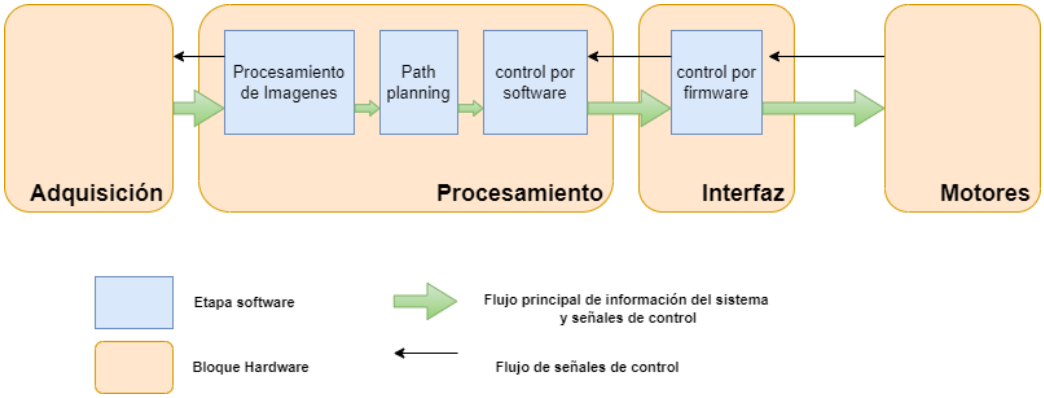


Figura 1.5: Pipeline del software del sistema PicassoBotZ.

un mínimo control del envío de comandos. Se envía de a un comando a la vez, esperando a que la interfaz pida el siguiente comando para efectivamente continuar con el envío.

Por último está la **etapa control por firmware**, que es la encargada de comunicarse directamente con los motores, enviándoles los comandos recibidos desde la etapa anterior, recibiendo las respuestas de los mismos y pidiendo comando a las etapas anteriores. En esta etapa es en donde se encuentra todo el firmware desarrollado para el sistema. También se realiza cierto control del sistema, dependiendo de la forma de pedir los comandos a la etapa anterior y de cómo se envían comandos a los motores.

1.4. Organización del documento

Los capítulos que continúan este documento están pensados de forma de que se alineen con la descripción de bloques que se hizo del sistema. De esta manera, el capítulo 2 explica inicialmente en detalle todas las partes que conforman el sistema, y se centra luego en el hardware utilizado a lo largo del proyecto. El capítulo 3 describe cómo se implementó el firmware utilizado y los distintos protocolos de comunicación que lo componen. Los capítulos 4, 5 y 6 refieren a distintas etapas del bloque de procesamiento. En particular, el capítulo 4 establece la teoría matemática y los sistemas de referencia usados para la etapa de path planning, que luego se describe en el capítulo 5. El capítulo 6 presenta las técnicas utilizadas en la etapa de image processing para la extracción de curvas en las imágenes. Luego, en el capítulo 7, se describen detalles de implementación para el sistema planteado, así como también las pruebas realizadas para validar y evaluar el funcionamiento del sistema final. Finalmente, las conclusiones y el trabajo a futuro se presentan en el capítulo 9.

Se destaca también que existe una serie de apéndices para cubrir algunos tópicos no esenciales para la comprensión del documento, pero que pueden ser de interés para el lector.

Capítulo 2

Arquitectura del Hardware

En este capítulo se describen los componentes de hardware que conforman el sistema PicassoBotZ en todas sus versiones. Se recuerda que el sistema PicassoBotZ no es único y por lo tanto cuenta con distintas combinaciones posibles de hardware. En esta línea, también se presentan las posibles combinaciones de hardware para tener un sistema PicassoBotZ funcional, y aquellas que fueron efectivamente utilizadas a lo largo del proyecto.

2.1. Sistema Completo

Como se explicó en detalle en la sección 1.3, hay 4 bloques de hardware, cada uno de los cuales puede estar conformado por un juego distinto de componentes indistintamente del resto (véase figura 1.4). Se explica a continuación cuáles componentes son válidos para cada uno de estos bloques.

2.1.1. Bloque de procesamiento

Este bloque puede estar conformado o bien por una PC, o bien por una Raspberry Pi comandada por una PC. En el caso de que el sistema esté conformado solamente por la PC, es esta la que comanda el bloque de adquisición la toma de la imagen, procesa la imagen para obtener una serie de comandos y envía estos comandos procesados a la siguiente etapa. La adquisición de imagen y el procesamiento se realizan ejecutando un programa en Python desde la PC, y el envío de comandos también se realiza ejecutando otro programa aparte en Python. Por otro lado, en el caso de tener además la Raspberry, es esta última la que realiza el pedido de adquisición, el procesamiento y el envío de los comandos. La diferencia con la situación anterior es que en este caso los comandos no se ejecutan directamente desde la terminal de la Raspberry, sino que se envían a la misma a través de una PC conectada a la Raspberry a través del protocolo SSH [11]. Esta separación fue realizada con el objetivo de poder comandar al sistema completo de forma remota. Como la unidad de procesamiento central del sistema tiene que estar cableada al bloque de interfaz y al de adquisición por la construcción del

sistema, este cableado se puede realizar con la Raspberry y encerrar todo esto en un bloque, separarlo y comandarlo a distancia desde una PC.

2.1.2. Bloque de adquisición

Este bloque es el más sencillo de todos. Puede estar conformado o bien por una cámara web, o bien puede directamente no estar en el sistema. Se deja la opción de no tener este bloque en el sistema porque el software en el bloque de procesamiento admite como entrada una imagen en cualquier formato que acepte openCV con su método .imread() en un directorio particular de su sistema. Se pueden ver los formatos disponibles de entrada del método .imread() en [12], entre los cuales se encuentran .png, .jpg, .jpeg, entre otros. De esta forma, la imagen puede ser obtenida de la forma que desee el usuario. Por otro lado, si se quiere tener un sistema completo y utilizarlo de forma remota, se deberá utilizar la cámara web que se conecta mediante USB a la unidad de procesamiento del bloque de procesamiento.

2.1.3. Bloque de interfaz

Este bloque solo tiene una versión posible en cuanto a hardware. Está conformado por una placa Arduino Mega, un buffer tri-state doble y la electrónica necesaria para unirlos. La placa Arduino se conecta por USB al bloque de procesamiento, y mediante uno de sus puertos seriales al buffer tri-state, que hace, a su vez, de interfaz entre el Arduino y los motores.

2.1.4. Bloque de motores

Por último, el bloque de motores está conformado por los motores elegidos y la fuente que los alimenta. Durante el proyecto se utilizaron dos modelos distintos de motores para realizar los dibujos. Al final se decantó por uno y es el que actualmente conforma este bloque, justificando esta elección en 2.3. De todas formas, el otro modelo perfectamente podría ser utilizado si se rompe un poco la separación en bloques realizada hasta ahora y se realizan modificaciones a estos. Es por esto que el modelo alternativo de motores no se incluye en el sistema final PicassoBotZ.

A modo de resumen y recordatorio, se presentan en la tabla 2.1 las posibles variantes de hardware que puede tener cada bloque. Todas las posibles combinaciones fueron probadas y se obtuvieron resultados similares.

A continuación se describe en mayor profundidad cada uno de los elementos hardware utilizados, ponderando la descripción por la relevancia que tienen en el sistema.

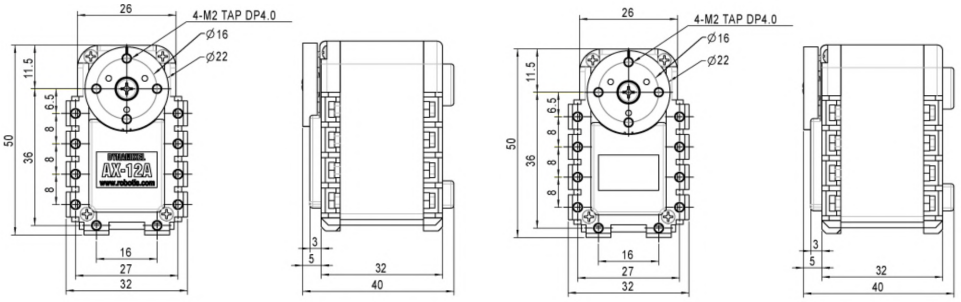
Tabla 2.1: Tabla de posibles variantes de componentes Hardware.

Bloque hardware	Conjunto de componentes	Conjunto alternativo de componentes
adquisición	Cámara web	Sin cámara
procesamiento	PC	Raspberry comandada por PC
interfaz	Arduino y buffers tri-state	Sin alternativa
motores	Motores MX-12W	Sin alternativa

2.2. Motores

Al comienzo de este proyecto se trabajó con un único modelo de motores, heredados del equipo PicassoBot. Estos motores son los AX-12A (de ahora en más, AX), de la marca Dynamixel de la compañía ROBOTIS. Pero, por sugerencia de este mismo equipo, se incorporó al proyecto un nuevo modelo de motores, los MX-12W (de ahora en más, MX), de la misma marca. De esta forma se pudo explorar una variedad de resultados más grande que solamente con los motores AX, debido a las prestaciones adicionales que los nuevos motores proporcionaron.

Ambos modelos de motores poseen la misma interfaz física, es decir que todos los accesorios vendidos por la marca y diseñados para un modelo de motores, son compatibles con el otro. Además, las conexiones necesarias para su alimentación y funcionamiento también son idénticas. (véase figuras 2.1 y 2.2).



(a) Diagrama de los motores AX (b) Diagrama de los motores MX

Figura 2.1: Imágenes descriptivas de las dimensiones de cada modelo de motor. Extraídas de [13] y [14] respectivamente.

Ambos modelos de motores también siguen el mismo principio de funcionamiento. Estos posen una tabla de control interna; al ser escrita esta tabla se le indica al motor cómo actuar, y al ser leída se obtiene información acerca del estado del motor. La tabla completa de los motores MX puede verse en la figura 2.3. Entre lo que se puede escribir en la tabla está: la posición objetivo, la velocidad de movimiento, el control de manejo de errores, entre otras. Por otro lado, lo que se puede leer de la tabla de control es, entre otras cosas, la posición actual, la velo-

Figura 2.2: Foto de un motor modelo AX (izquierda) y otro MX (derecha).

cidad de movimiento, la temperatura y el torque actual. En particular, se destaca por su amplio uso el registro **Moving**, que al leerlo indica si los motores se están moviendo o no. Para mayor referencia se puede consultar los manuales electrónicos de los modelos AX-12A [13] y MX-12W [14]. También, en ambos modelos de motores, hay una parte de la tabla de control implementada en RAM y otra en EEPROM. De esta forma los motores pueden almacenar información, aún cuando estos dejen de ser alimentados.

De las tablas mencionadas, hay dos registros que son los más importantes para efectivamente hacer funcionar el motor. El registro de "Goal Position" y el de "Moving Speed". Con estos dos registros se configura la posición a la que se quiere que el motor se mueva (Goal Position), y con qué velocidad (su Moving Speed). Aunque se tengan estos dos registros, no resulta claro en la documentación de los motores si efectivamente se puede esperar que los motores realicen sus movimientos a velocidad constante fijada por el usuario. Dicho problema se abordará en la sección 2.3 de este mismo capítulo.

Para comunicarse con los motores y así escribir o leer esta tabla de control se debe hacer uso del "DYNAMIXEL Protocol 1.0" [15], protocolo de comunicación bidireccional que funciona sobre UART[1]. Este es un protocolo basado en paquetes, en el cual primero un dispositivo adecuado envía lo que se llama un "Instruction

[1]del inglés *universal asynchronous receiver-transmitter*

Address	Size(Byte)	Data Name	Description	Access	Initial Value
0	2	Model Number	Model Number	R	360
2	1	Firmware Version	Firmware Version	R	-
3	1	ID	DYNAMIXEL ID	RW	1
4	1	Baud Rate	Communication Speed	RW	1
5	1	Return Delay Time	Response Delay Time	RW	250
6	2	CW Angle Limit	Clockwise Angle Limit	RW	0
8	2	CCW Angle Limit	Counter-Clockwise Angle Limit	RW	4,095
11	1	Temperature Limit	Maximum Internal Temperature Limit	RW	70
12	1	Min Voltage Limit	Minimum Input Voltage Limit	RW	60
13	1	Max Voltage Limit	Maximum Input Voltage Limit	RW	160
14	2	Max Torque	Maximun Torque	RW	1023
16	1	Status Return Level	Select Types of Status Return	RW	2
17	1	Alarm LED	LED for Alarm	RW	36
18	1	Shutdown	Shutdown Error Information	RW	36
20	2	Multi Turn Offset	Adjust Position with Offset	RW	0
22	1	Resolution Divider	Divider for Position Resolution	RW	1

(a) Sección de la tabla de control de los motores MX implementada en EEPROM

Address	Size(Byte)	Data Name	Description	Access	Initial Value
24	1	Torque Enable	Motor Torque On/Off	RW	0
25	1	LED	Status LED On/Off	RW	0
26	1	D Gain	Derivative Gain	RW	8
27	1	I Gain	Integral Gain	RW	0
28	1	P Gain	Proportional Gain	RW	8
30	2	Goal Position	Desired Position	RW	-
32	2	Moving Speed	Moving Speed(Moving Velocity)	RW	-
34	2	Torque Limit	Torque Limit	RW	Max Torque
36	2	Present Position	Present Position	R	-
38	2	Present Speed	Present Speed	R	-
40	2	Present Load	Present Load	R	-
42	1	Present Voltage	Present Voltage	R	-
43	1	Present Temperature	Present Temperature	R	-
44	1	Registered	If Instruction is registered	R	0
46	1	Moving	Movement Status	R	0
47	1	Lock	Locking EEPROM	RW	0
48	2	Punch	Minimum Current Threshold	RW	32
50	2	Realtime Tick	Count Time in millisecond	R	0
73	1	Goal Acceleration	Goal Acceleration	RW	0

(b) Sección de la tabla de control de los motores MX implementada en RAM

Figura 2.3: Tabla de control de los motores MX. Imagen tomada de [14].

packet" al motor y luego el motor responde con un "Status Packet". El motor nunca envía paquetes sin un envío previo hacia él. Este protocolo además admite que varios motores compartan la misma capa física, utilizando una estrategia de asignar una ID única a cada motor. De esta forma se puede comandar a varios motores compatibles con este protocolo utilizando solo un dispositivo de control (como por ejemplo, un Arduino). Lo que es más, los motores están diseñados para poder conectarse unos a otros mediante una configuración de Daisy Chain como se ve en la figura 2.4, lo que facilita que se comparta la capa física de la comunicación.

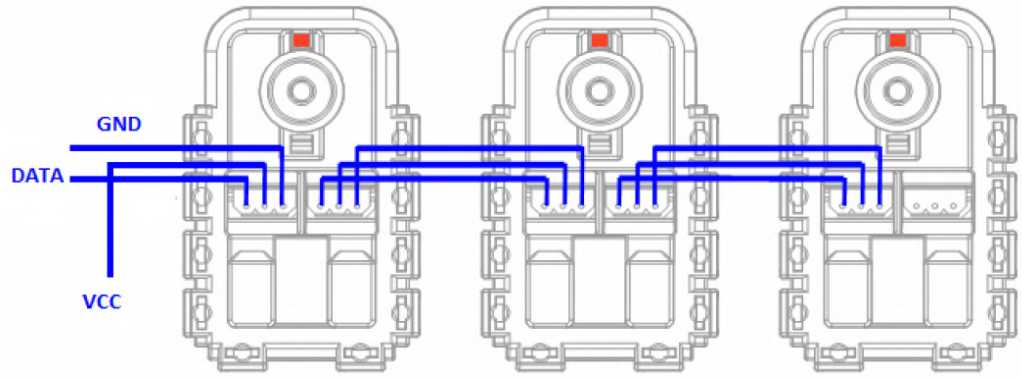
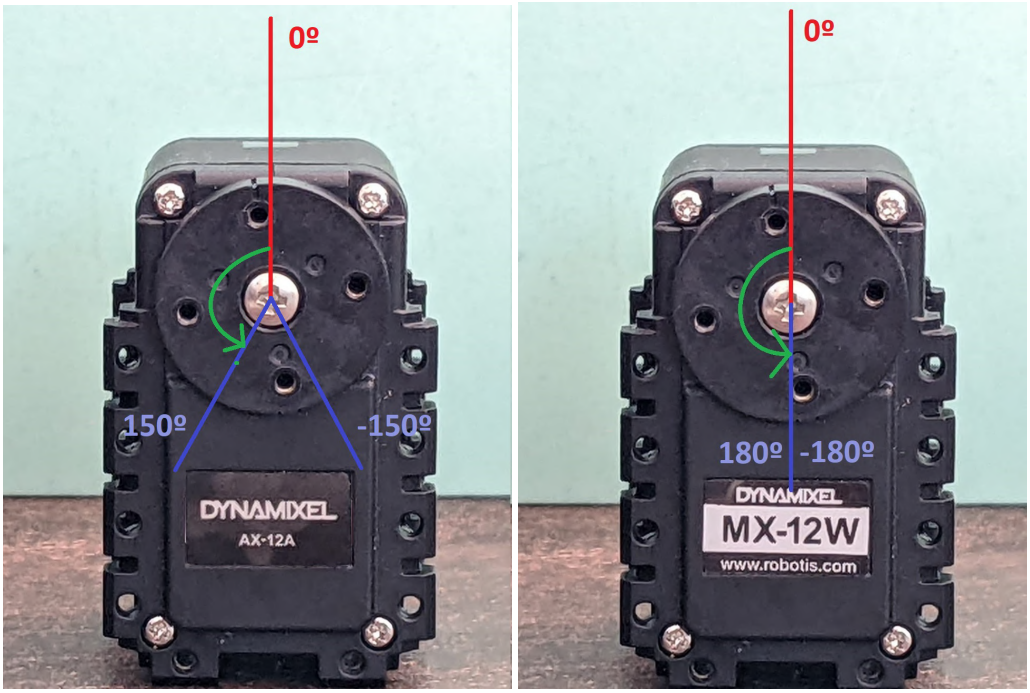


Figura 2.4: Tres motores MX conectados en daisy chain. Imagen tomada de [1].

Estas similitudes hicieron que fuese sencillo utilizar un juego de motores ya habiendo utilizado otro. Estas mismas similitudes son las que permiten utilizar

la misma interfaz física para ambos, modificando en contados detalles el firmware desarrollado para comunicarse con los mismos.

De todas formas, y es por esto que se consiguieron nuevos motores, hay muchas diferencias entre los motores a nivel de funcionalidad. La mayoría de las diferencias entre ambos modelos radica en lo que son capaces de hacer, en contraposición a la forma en que se comunica con ellos. Por ejemplo, los motores MX tienen mayor rango angular, pudiendo recorrer la totalidad de los 360 grados, mientras que los AX solo pueden moverse en un rango angular de 300 grados. Estos rangos y el sistema de referencia utilizado se pueden ver en la figura 2.5. Además, en el caso de los MX este rango está discretizado en un total de 4096 valores; mientras que en el caso de los AX la discretización se hace con 1024 valores. En consecuencia de estas dos últimas diferencias, también se puede deducir que la precisión angular es bastante más fina en los MX ($\approx 0.088°/unidad$) que en los AX ($\approx 0.29°/unidad$). Una lista de las diferencias entre ambos motores se puede ver en la tabla 2.2, entre las cuales se destaca su rango de velocidades, resolución en velocidad y su voltaje de alimentación.



(a) Sistema de referencia y límites angulares de los motores AX

(b) Sistema de referencia y límites angulares de los motores MX

Figura 2.5: Sistemas de referencia utilizados para los motores.

Además de las diferencias previamente mencionadas, hay una que fue central en la elección de los nuevos motores: el método de control que se puede aplicar sobre la posición de cada motor. Los motores de modelo AX utilizan una estrategia de control en lazo abierto basado en dos valores denominados **Compliance Margin**

Tabla 2.2: Tabla Comparativa entre MX-12W y AX-12A.

Característica	**MX-12W**	**AX-12A**
Baud Rate	8000bps - 4.5Mbps	7843bps - 1Mbps
Control Algorithm	Control PID	
Resolution	4096 pulse/rev	0.29^{o}
Operating Modes	Joint Mode (0 - 360^{o})	Joint Mode (0 - 300^{o})
Weight	54.6g	53.5g
Stall Torque	0.2Nm @ 12V; 1.4A	1.5Nm @ 12V; 1.5A
No Load Speed	470 rev/min @ 12V	59 rev/min @ 12V
Operating Temperature	-5 - 70^{o}C	-5 - 70^{o}C
Input Voltage	10.0 - 14.8V Recomended 12V	9.0 - 12.0V Recomended 11.1V
Protocol Type	Half Duplex Asynchronous Serial Communication (8bit, 1stop, No Parity)	Half Duplex Asynchronous Serial Communication (8bit, 1stop, No Parity)
Physical Connection	TTL Level Multidrop Bus (Daisy Chain Type Connector)	TTL Level Multidrop Bus
ID	253 (0 - 252)	254 (0 - 253)
Feedback	Position Temperature Load Input Voltage etc	Position Temperature Load Input Voltage etc
Standby Current	60 mA	

y **Compliance Slope**. Por otro lado, los motores modelo MX utilizan un sistema realimentado en lazo cerrado basado en PID.

La estrategia de los AX consiste en una disminución progresiva del torque ejercido sobre el eje de los motores, a medida que el eje se acerca a su posición objetivo. La forma en la que varía este torque es controlada por los dos valores mencionados previamente (Compliance Margin y Compliance Slope). Esta estrategia se puede visualizar en la figura 2.6.

En el eje vertical de esta figura se representa el torque ejercido, mientras que en el horizontal se representa la posición angular del eje del motor. La idea de esta estrategia es lograr decrementar el torque lo suficiente cuando se está cerca del objetivo para no pasarse del mismo. Se puede observar también en la figura 2.6 el significado del Compliance Margin y Compliance Slope. En un movimiento del brazo hacia una posición "Goal Position", el brazo ejercerá un torque constante

hasta encontrarse a una distancia de Compliance Margin + Compliance Slope del objetivo. A partir de esta distancia, su toque comenzará a disminuir con pendiente constante. Luego, a una distancia de Compliance Margin del objetivo, es que se dejará de ejercer torque sobre el eje del motor. De esta forma el rozamiento es quien se deberá encargar de frenar el motor en la posición adecuada.

Se observa también en la figura que existe un valor denominado **Punch**. Según la documentación de los AX este valor es la mínima corriente requerida para mover los motores. Esto resulta confuso, puesto que las unidades de corriente y de torque son claramente distintas. Lo que se puede interpretar de este valor aquí es que el torque tiene un mínimo posible definido por la mínima cantidad de corriente para mover el motor. Esto se respalda en que si se disminuye el torque ejercido, entonces debe de disminuirse la corriente absorbida de la fuente; y como hay un límite inferior para la corriente, también lo debe de haber para el torque.

También se observa en la figura 2.6 que hay dos Compliance Margin y dos Compliance Slope. Esto es porque se puede configurar estos valores de forma diferenciada para movimientos antihorarios (CCW, del inglés *counter-clockwise*) y horarios (CW, del inglés *clockwise*).

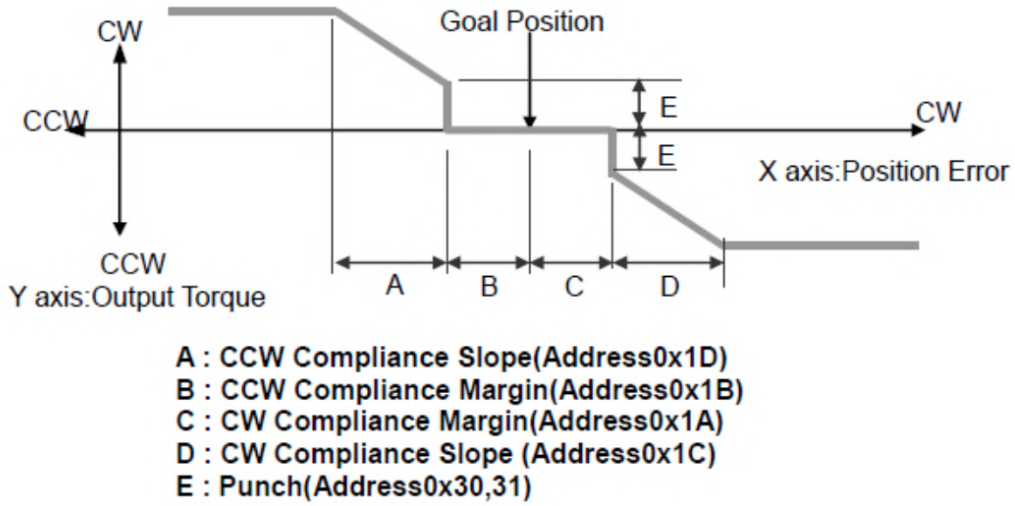


Figura 2.6: Representación gráfica del Compliance Margin, Compliance Slope and Punch. Obtenida de [13].

Por otro lado, la estrategia de los MX consiste en realimentar la posición usando un controlador PID integrado en la electrónica de los motores. El diagrama de este sistema de control realimentado se puede ver en la figura 2.7. El valor de cada una de las constantes del controlador PID es configurable desde la tabla de control de cada motor. Además de poder configurar estos valores, en el caso de los motores MX también se puede configurar la "Goal Acceleration" del motor. En el manual digital de los motores quedan dudas de cómo es que funciona este registro. De todas formas, por resultados experimentales de este proyecto, parece ser que se utiliza para configurar una aceleración objetivo a la que se quiere llegar en el movimiento

de los motores. De esta forma, el motor realiza algún tipo de control para llegar a dicha aceleración y no alejarse demasiado de esta.

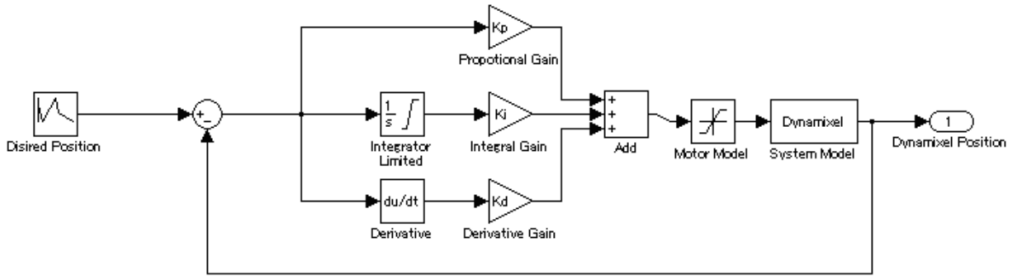


Figura 2.7: Diagrama de bloques del sistema realimentado de los MX. Obtenido de [14].

Como prestación adicional que fue relevante al proyecto, ambos modelos de motores también poseen la capacidad de realimentar el torque que ellos están experimentando en un momento dado. En particular, esta capacidad fue utilizada para regular la presión realizada con la lapicera en el extremo del brazo sobre la hoja en la que se quería dibujar. La implementación de este mecanismo se desarrolla en la sección 3.4.

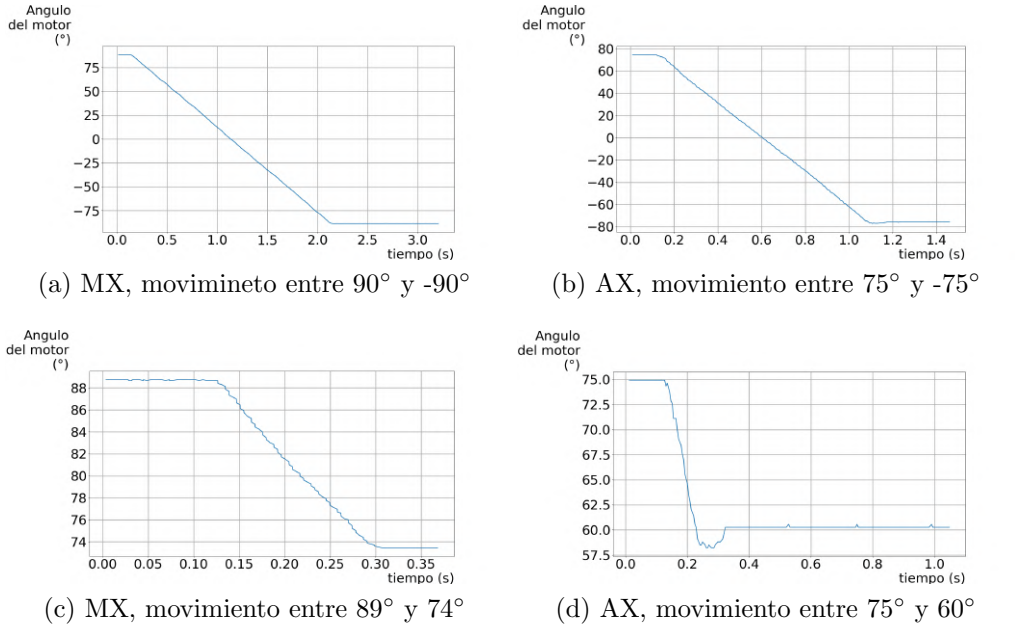
También fue relevante a este proyecto el registro **Shutdown**. Cuando el motor experimenta uno de varios errores posibles, este se detiene, realiza un parpadeo con el led (si fue así configurado), y almacena en el registro "Shutdown" información sobre dicho error. De esta forma fue que en el proyecto se observó que la temperatura de estos motores MX se elevaba con mucha mayor rapidez que en los AX. Se realizaron pruebas y optimizaciones en respuesta a esto, detalladas en la sección 7.1.

2.3. Cinemática y dinámica de cada motor

Como se vio en la sección de 2.2 no queda claro qué esperar del perfil de velocidad de cada motor en función del tiempo para un movimiento dado. Lo "razonable" podría ser esperar algo parecido a un pulso en velocidad, o lo que es lo mismo, una rampa en posición. Esto debido a que para cada movimiento se puede configurar tanto la velocidad con la que se mueve como su posición objetivo.

Según lo dicho en [2], el perfil de velocidades de los motores es esencialmente trapezoidal. Pero como ya se cuenta con firmware abundante para control de los motores y obtener realimentación de estos, se decidió realizar unas pruebas sencillas en los motores para validar esta afirmación. Las pruebas fueron realizadas primero para los motores sin carga, con la configuración por defecto. Luego, para los motores sin carga, variando sus configuraciones. Y finalmente se realizaron, para los MX, pruebas con carga. Cabe aclarar que la carga de un motor sería cualquier cuerpo externo que se le acople a su parte rotativa, por ejemplo, otro motor o motores.

A continuación, en la figura 2.8 se puede ver la posición en función del tiempo para ambos modelos de motores, sin carga, con su configuración por defecto. Se realizaron movimientos con amplitud angular "grande" y "pequeña", ya que no se sabía aún qué tipos de movimientos se priorizarían a la hora de dibujar.



(a) MX, movimineto entre 90° y -90°

(b) AX, movimiento entre 75° y -75°

(c) MX, movimiento entre 89° y 74°

(d) AX, movimiento entre 75° y 60°

Figura 2.8: Posición en función del tiempo para distintos rangos angulares de los motores AX y MX sin carga. Los valores numéricos de los ángulos son aproximados.

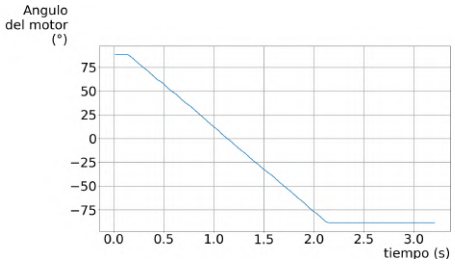
En la figura 2.8 se puede ver que para movimientos de amplitud angular "grande", ambos motores tienen un comportamiento similar y bastante parecido a una rampa. Los puntos singulares que deberían tener en el cambio de pendiente son más observables en los motores AX, pero pequeños en comparación con el movimiento total.

Por otro lado, en los movimientos pequeños sí se nota una gran diferencia entre los distintos motores. En ambas gráficas se observa un ruido mayor en la posición en función del tiempo. Pero en la gráfica de los motores AX, se observa inmediatamente que se pierde completamente la rampa en posición sobre el final de la misma. Hay un pronunciado sobretiro.

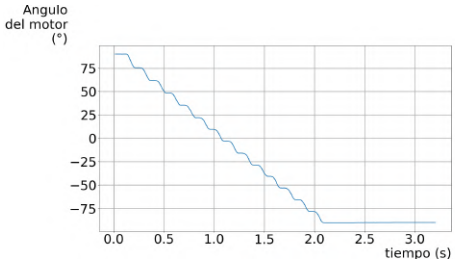
Luego de analizar estas gráficas, observar la fluidez de los MX y tener los MX más prestaciones, se decide utilizar este modelo de motor para la construcción final del brazo. A parte de las ventajas que ya ofrecen los MX, hubo otros comportamientos no deseados por parte de los motores AX[2] que apoyaron la decisión de no usarlos.

[2]Véase el video `https://www.youtube.com/shorts/xLZRPWTHlxg`. En este se muestra cómo, para un movimiento oscilatorio, los motores exhiben distintos comportamientos según la dirección en la que se mueva.

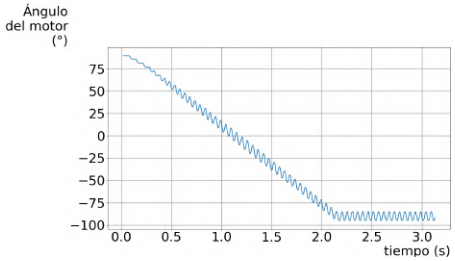
Al ser los motores MX la elección final, se realizaron pruebas para corroborar que las constantes del PID efectivamente modificaban el movimiento de los motores sin carga. De ser así, se podrá controlar el movimiento de cada motor cuando se le agregue la carga. Las gráficas obtenidas para estas pruebas se pueden ver en la figura 2.9. En esta figura se observan gráficas de posición en función del tiempo para un motor dynamixel sin carga. Para obtener cada una de ellas, se hizo que el motor realizara un mismo movimiento, desde un ángulo de $-90°$ a $90°$, pero con distintos valores de constantes PID. Cabe destacar que los efectos, para los motores sin carga, del derivador son imperceptibles, es por eso que se ve su efecto sumado al de la constante proporcional. En estas se observa que efectivamente al variar las constantes del controlador la curva obtenida varía de forma significativa. A partir de esto se puede concluir que efectivamente la cinemática de los brazos es afectada por las modificaciones en las distintas constantes del PID.



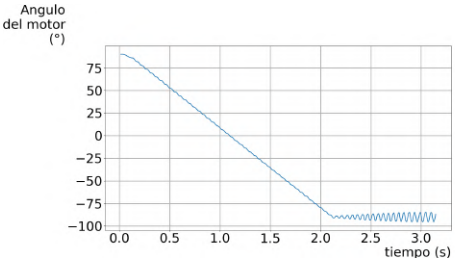
(a) MX, movimiento entre 90° y -90°



(b) MX, movimiento entre 90° y -90° con alta constante de integrador



(c) MX, movimiento entre 90° y -90° con alta constante de derivador y proporcional



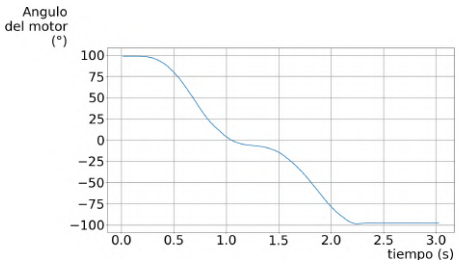
(d) MX, movimiento entre 90° y -90° con alta constante de proporcional

Figura 2.9: Efectos de las distintas combinaciones de valores para las constantes del PID de los motores MX, sin carga.

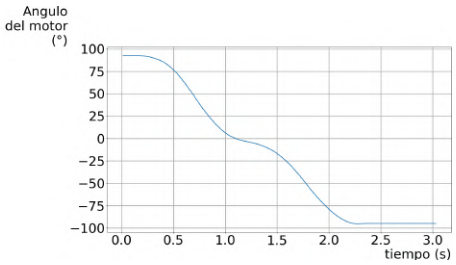
Es entonces posible (en teoría) el control de los motores para el caso con carga. De todas formas, esto también implica que un mal ajuste de las constantes de los motores puede llevar rápidamente a una inestabilidad

En la figura 2.10 se presentan los efectos de la compensación PID modificando solamente la constante proporcional. Para obtener estas, se realizó un mismo movimiento del motor correspondiente al hombro del brazo armado. Este movimiento se realizó para un brazo sin compensar, y compensado. Además, también se realizó

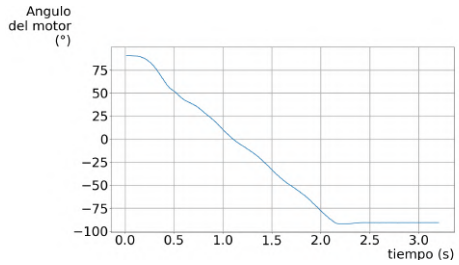
para dos configuraciones del resto del brazo, de esta forma se puede ver el efecto que tiene variar la carga del motor en su movimiento.



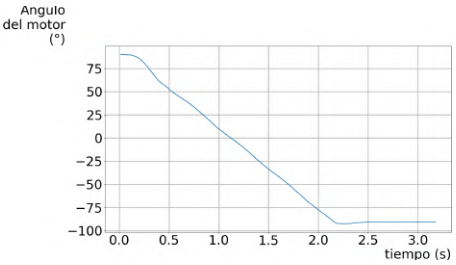
(a) MX con carga (brazo recto), movimiento entre 90° y -90°



(b) MX con carga (brazo torcido), movimiento entre 90° y -90°



(c) MX con carga (brazo recto) y compensación, movimiento entre 90° y -90°



(d) MX con carga (brazo torcido) y compensación, movimiento entre 90° y -90°

Figura 2.10: Efectos de la compensación PID en los motores MX con dos cargas distintas: brazo recto (todos los motores acoplados y rectos), y brazo torcido (todos los motores acoplados y enrollados sobre sí mismos).

2.4. Morfología del brazo

Como se mencionó en el capítulo 1, el presente proyecto de fin de carrera está basado en la obra artística *Paul the Robot* realizada por Patrick Tresset. *Paul the Robot* fue diseñado para imitar un humano al dibujar, en consecuencia, se construyó para que su forma fuese lo más antropomórfica posible. Paul está constituido por cuatro servomotores unidos secuencialmente. En los proyectos PARRA y PicassoBot se intentó reconstruir una morfología similar a la utilizada por Tresset.

En el proyecto Picassobot, al utilizar a *Paul the Robot* como referencia, se utilizaron también cuatro motores que representan el brazo, el antebrazo, la mano y la muñeca de un brazo real. Pese a utilizar motores con las mismas dimensiones que los utilizados por Patrick Tresset, no utilizaron la misma disposición física de los mismos. En su lugar, decidieron poner todos los motores orientados hacia el mismo lado como se muestra en la figura 2.11.

Las partes que constituyen un brazo real (el brazo, el antebrazo y la mano) estarían definidas por la distancia entre cada par de articulaciones, definiéndose a cada articulación como el eje de cada motor. Sin embargo, para la mano se tuvo

Figura 2.11: Forma del brazo realizado por el equipo PicassoBot.

que utilizar dos articulaciones: una que representase el movimiento horizontal, y otra que representara el movimiento vertical (apoyar o no apoyar la lapicera).

Por otro lado, en el proyecto original de Patrick Tresset, los motores no estaban acoplados con la misma disposición, sino que algunos se acoplaban por el eje con el siguiente, mientras que otros se acoplaban por la parte opuesta al eje. Este brazo se puede ver en la figura 2.12

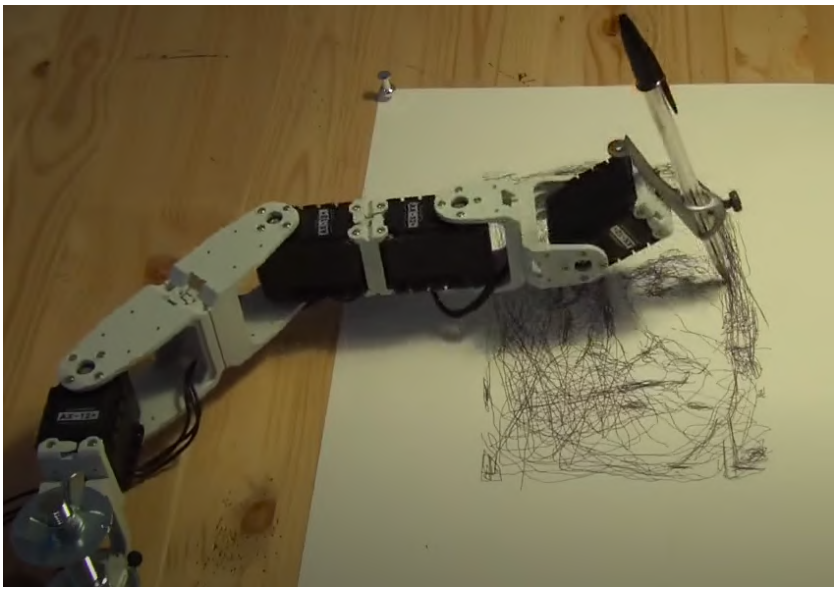


Figura 2.12: Estructura de Paul the Robot [8].

La morfología del brazo obtenida por el grupo PicassoBot facilitaba a la hora de calcular los ángulos a los cuales se movería el brazo y también los rangos de movimiento de cada servomotor. Sin embargo, es una morfología que no contempla las dimensiones de un brazo humano, ya que con esa forma, el brazo y el antebrazo permanecían de igual tamaño y más chico que la mano-muñeca, el cual es un tamaño antinatural.

De este modo, el proyecto aquí desarrollado se utilizó la morfología de Paul de Robot. Esta morfología cuenta con brackets más grandes que los que utilizaba el PicassoBot, y el motor que simula el codo se encuentra al revés. Esto provoca que se aproxime un poco más al brazo humano, si bien sigue siendo un tamaño antinatural por las proporciones que se manejan. En la figura 2.13 se puede apreciar el brazo armado en el proyecto PicassoBotZ.

Figura 2.13: Forma del brazo realizado por PicassoBotZ.

Un análisis interesante a realizar, fue el de comparar las proporciones de las distintas morfologías vistas con las proporciones entre las partes de un brazo real. En el canon de las 8 cabezas (canon que corresponde con la figura ideal de una persona y es usado por escultores y pintores) [16] se establecen las proporciones normales entre las partes de un brazo con la cabeza. Entonces se puede deducir qué proporción del total tiene cada parte. En la tabla 2.3 se puede ver una comparación entre medidas y proporciones para el canon y los brazos robóticos construidos por el proyecto PicassoBot y el actual.

Las medidas tomadas son las distancias entre las articulaciones de los motores que conforman el brazo entero. Se aprecia que en el proyecto PicassoBot el brazo y antebrazo tienen el mismo tamaño y sobre todo que la mano es casi el doble que las otras partes. Buscando asimilar las proporciones a las de un brazo real es que se alarga el brazo y antebrazo cambiando la disposición de los motores, tal como

Tabla 2.3: Tabla de medidas y proporciones para un brazo canónico, comparado con los brazos robóticos construidos por PicassoBot y el presente proyecto.

	Parte	**Medida**	**Proporción**
Canon	Brazo	1 cabeza	33 %
	Antebrazo	1.5 cabezas	50 %
	Mano	0.5 cabezas	17 %
PicassoBot	Brazo	6.8 cm	26 %
	Antebrazo	6.8 cm	26 %
	Mano	12.8 cm	48 %
PicassoBotZ	Brazo	10.7 cm	35 %
	Antebrazo	8.3 cm	27 %
	Mano	11.5 cm	38 %

lo había hecho Tresset. Si bien la mano sigue siendo la parte más grande, se logra achicar notablemente la diferencia y acercarse más a las proporciones reales. Por otro lado, el antebrazo es más chico que el brazo, lo que antes no era así, pero globalmente las nuevas proporciones del brazo reflejan mejor las de un humano.

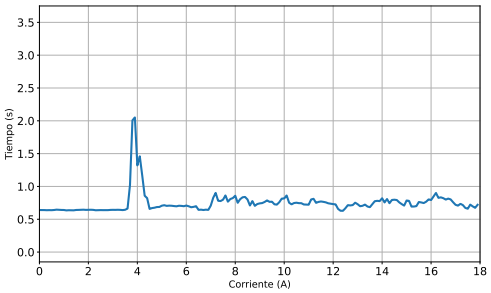
Otra modificación importante que se hizo con respecto al brazo del proyecto PicassoBot, fue cambiar la punta que lleva el elemento que dibuja. Se decidió que la lapicera estuviera sostenida por un brazo de compás, en lugar de por un bracket de Robotis, de forma que sea mas fácil intercambiar la lapicera cuando esta se termina.

Al armar el brazo se observó que el comportamiento dinámico de los motores no era el documentado en la sección 2.3. Este cambio de comportamiento se debe a que la carga que tiene cada motor es diferente entre sí y es diferente en comparación con el motor sin carga. De esta forma, se concluyó que se deben ajustar los valores del controlador PID de cada motor para disminuir el efecto generado por la carga que presenta cada uno y por la carga variable que tiene. Este cambio en la carga se debe a que, a medida que los motores se mueven, la forma del brazo cambia y por ende cambia su momento de inercia y su centro de masa. Se vio empíricamente, que este cambio de movimiento que experimentaba el brazo a lo largo de un dibujo, repercutía de manera diferente en cada motor y a lo largo del tiempo de la ejecución del dibujo. Debido a esto, sintonizar el PID de cada motor fue una tarea central para este proyecto para poder tener un comportamiento predecible de los motores.

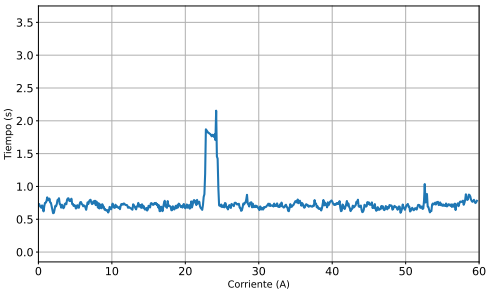
2.5. Fuente de Alimentación

Como se mencionó en la sección 2.2, según el manual de los motores MX [14], estos deben ser alimentados con un voltaje en un rango de entre 10V y 14.8V, siendo el recomendado 12V. Además, también según el manual, pueden haber condiciones en las que se requieran hasta 1.4 A de corriente para los motores. Es por esto que se optó por una fuente AC-DC (un cargador común con un transformador) de 12 V y 5A.

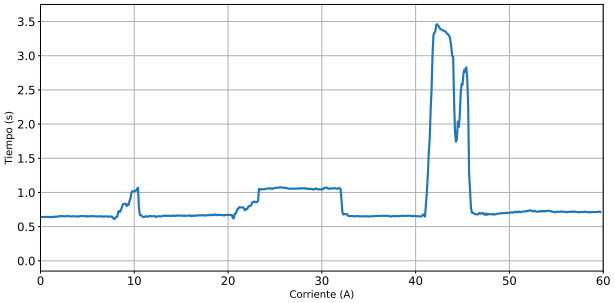
Si bien el amperaje de la fuente no cubre en totalidad el consumo máximo sumado de los cuatro motores en serie, no se creía en un principio que los motores realmente requirieran esta cantidad de corriente para realizar dibujos. Se midió entonces la corriente total que consumen los cuatro motores en movimiento. Dicha medición se realizó poniendo una resistencia Shunt (de 1.5 Ω y 1W) a la alimentación de los motores y midiendo la caída de voltaje en esta resistencia.



(a) Inicio de un dibujo.



(b) Perturbación externa mientras el brazo dibuja



(c) Distintas fuerzas aplicadas en el brazo cuando está en reposo.

Figura 2.14: Mediciones de la corriente que entrega la fuente en función del tiempo para situaciones donde hay mayor consumo. En el primero caso, los motores pasan de estar quietos a moverse todos a la vez lo que genera un pico de corriente. En los otros dos, las fuerzas aplicadas como perturbaciones externas incrementan el consumo de corriente. De igual forma, el consumo de corriente nunca se acerca a los 5 Ampere, lo que indica que la fuente utilizada es suficiente.

En la figura 2.14 se pueden ver tres comportamientos diferentes del brazo. En el primer gráfico 2.14b se observa el funcionamiento normal del motor en medio de un dibujo. Se percibe que si bien tiene un pico de corriente, este apenas alcanza los 2A. En la segunda gráfica 2.14a se puede observar el comportamiento del brazo mientras está dibujando y como se le perturba con una fuerza. En la gráfica 2.14c se ve un caso similar, solo que los motores están energizados y quietos, y se les hace fuerza para tratar de moverlos. En estos casos la corriente sube considerablemente, pero igualmente esta sigue siendo menor a la que puede suministrar la fuente de energía.

Anteriormente, el equipo PicassoBot usaba una fuente de laboratorio para ajustar el voltaje con mayor precisión y alimentar todos los motores. Esto era en parte porque usaban un voltaje más específico de 11.1 V, pero también era incómodo y poco práctico para transportar. Al usar un cargador como fuente esto deja de ser un problema y el sistema es más portable.

2.6. Interfaz de Comunicación entre el Arduino y los motores

Como se mencionó en la sección 2.2 los motores se comunican bidereccionalmente, esto significa que por el mismo cable se envían y reciben datos. Sin embargo, todos los pines del arduino son unidireccionales, aunque hay pines especiales que son los Tx y Rx que sirven para enviar o recibir datos. Por esta razón se debió comprar un circuito Tri-state para lograr enviar y recibir información entre Arduino y el motor, sin que haya conflicto.

Debido a que este es un problema recurrente en estos motores, el fabricante Robotis ofrece una solución estándar al problema que se visualiza en la figura 2.15. En esta se observa el circuito que se debe realizar para el integrado 74LVC2G241 o NC7WZ241. Estos integrados están hechos con tecnología CMOS, y soportan velocidades de transmisión de hasta 344MHz. En este circuito hay una señal de activación para enviar o para recibir datos del o hacia el motor. A su vez, se ve en la figura que se utilizan resistencias de pull-up y de pull-down de forma que las señales que entran y salen se parezcan lo más posible a 5V y 0V para no causar distorsiones.

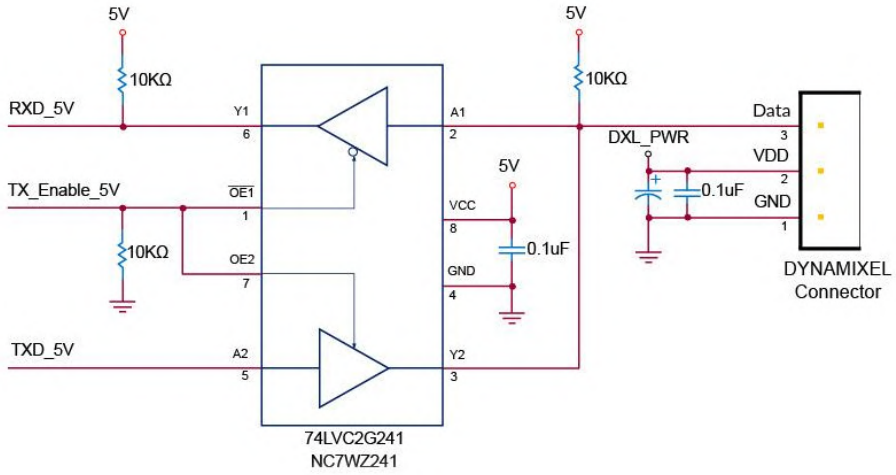


Figura 2.15: Circuito para la comunicación recomendado por Robotis en el manual de los motores MX [14].

Puesto que no se encontraron dichos integrados en plaza, se optó por hacer un circuito similar pero con el integrado 74LS241N. Este integrado, al igual que los

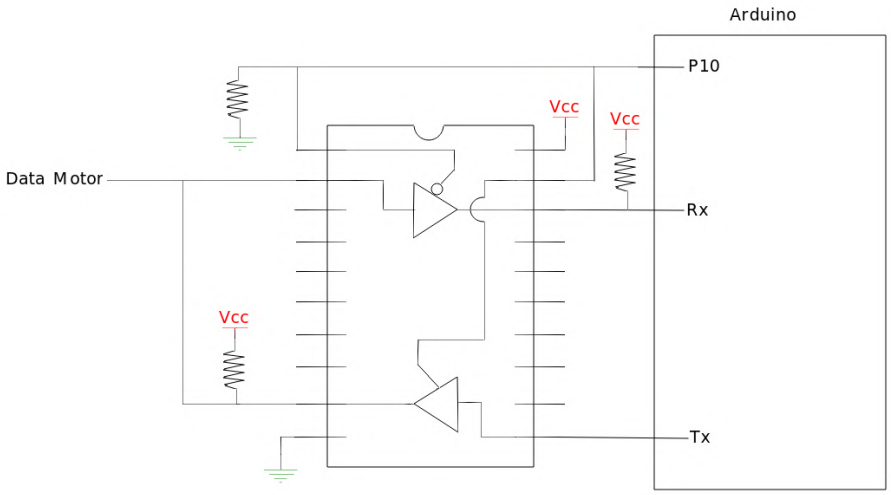


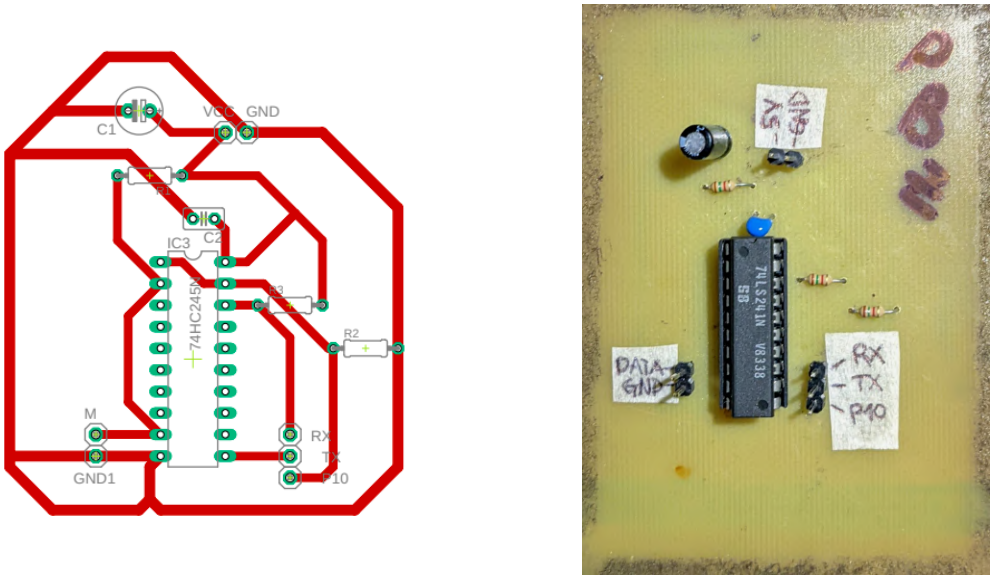
Figura 2.16: Circuito para la comunicación realizado para el presente proyecto. Se utilizó el integrado 74LS241N.

que ofrece Robotis, funciona con una fuente de alimentación de 5V (voltaje que puede otorgar el Arduino). Sin embargo, éste está hecho con tecnología BJT, lo cual a priori lo hace más lento que el integrado en comparación. Según la hoja de datos de este integrado, su frecuencia de transmisión es de 55MHz. De esta forma, si bien el integrado que se consiguió es considerablemente más lento, como la comunicación entre los motores y el Arduino se realiza a una frecuencia de 1MHz, el integrado encontrado en plaza funciona de manera adecuada.

En la figura 2.16 se puede apreciar el circuito que se utilizó efectivamente. Vale aclarar que se utilizó como base el circuito sugerido por Robotis. El circuito impreso que se muestra en la figura 2.17, donde se puede ver el modelo creado en Eagle para imprimir, y que se realizó sobre una placa simple de cobre. De un lado están los componentes eléctricos y del otro lado están las pistas de cobre. También se observan capacitores entre GND y VCC. Estos son condensadores de desacople, y su propósito es el de proporcionar energía en los picos de consumo del integrado. Además se ven las conexiones que se encuentran entre el motor y el Arduino que se ven en la figura 2.16 donde a la izquierda del circuito se encontraría el motor y a la derecha el Arduino.

En una primera instancia, se utilizaron resistencias de 10kΩ como sugería Robotis para hacer los pull-up del circuito. Estas resistencias hacían que la comunicación fuera más lenta de lo deseado y de esta forma el circuito no se comportaba como se pretendía, este efecto se muestra en la figura 2.18. En esta se ve cómo varia la información debido al circuito Tri-State. Se puede observar que hay una baja en la amplitud de la señal que le llega a los motores.

Para solucionar esto se optó por bajar las resistencias, con el objetivo de confirmar que la corriente que entraría a los terminales del integrado estuviese dentro de los rangos admitidos para que no se quemara. En la figura 2.19 se puede ver cómo es la señal que le llega a los motores cuando se utiliza una resistencia de



(a) Circuito diseñado en Eagle

(b) Circuito impreso y soldado

Figura 2.17: Diseño del pcb realizado en Eagle que luego se imprimió en una placa de cobre. El circuito se conecta entre el Arduino y los motores para hacer posible la comunicación entre ellos usando el integrado con buffers tri-state.

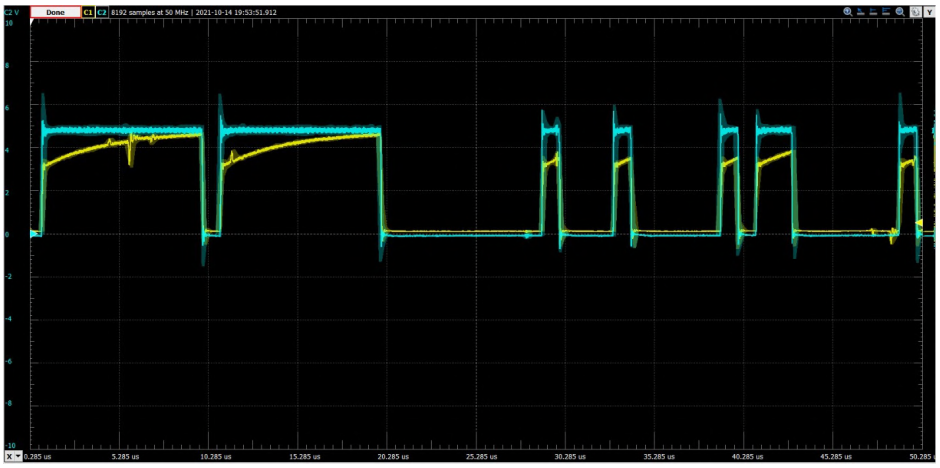


Figura 2.18: Gráfica del canal de comunicación, donde la curva cian corresponde a la salida del Arduino. La curva amarilla corresponde a la señal que le llega a los motores luego de pasar por el buffer tri-state.

1.5kΩ. Si se compara esta gráfica con la gráfica amarilla de la figura 2.18 se puede apreciar que, con una resistencia menor, el envío de datos es mejor y más parecido a la señal de salida del Arduino. Como se comprobó que se podían usar resistencias menores, se optó por usar resistencias de 1.5kΩ.

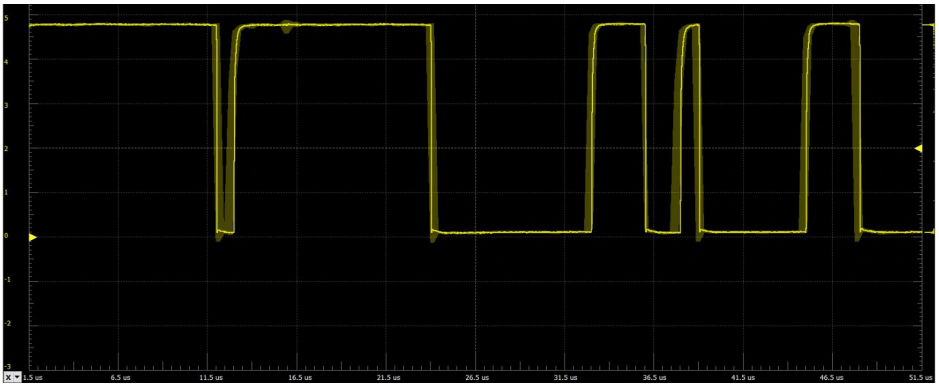


Figura 2.19: Gráfica del canal de comunicación utilizando resistencias de pull-up de 1.5kΩ.

2.7. Arduino

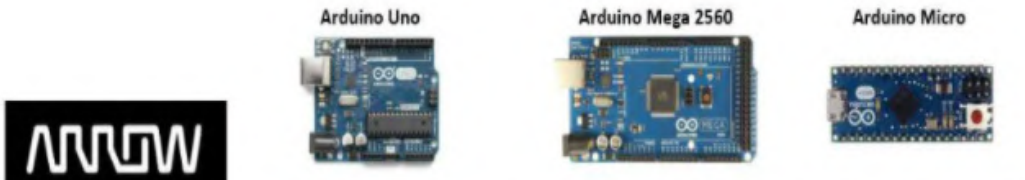
Para este proyecto fue utilizado el Arduino MEGA 2560, cuyas las principales especificaciones de este producto son:

- 54 I/O digitales
- Corriente continua para cada pin I/O 20mA
- Corriente continua para pin 3.3V 50mA
- Memoria flash 256 KB
- SRAM 8 KB
- EEPROM 4 KB
- Frecuencia de reloj de 16MHz.

Vale aclarar que cada uno de sus 54 pines digitales, se puede usar como entrada o salida según la programación que se desee. Además hay pines que tienen funciones especializadas, como por ejemplo, los pines seriales. Hay 4 pares de estos pines que se utilizan para recibir (Rx) y transmitir (Tx) datos en serie TTL. Se hace énfasis en estos pines ya que, como se explicó en la sección anterior, estos fueron utilizados para la comunicación con los motores.

Este microcontrolador se utiliza en este proyecto para la comunicación entre la computadora (o Raspberry) y los servomotores que conforman el brazo. Además de decodificar los comandos que llegan desde la computadora como se presenta en detalle en el capítulo 3, en el programa del Arduino se programan los diferentes registros de cada motor, seteando los valores de las constantes del PID, la aceleración de cada motor, y otros registros que se mencionaron en la sección 2.2.

Se consideró la posibilidad de utilizar una placa Arduino UNO en lugar del modelo MEGA, pero se eligió esta placa para el sistema, ya que es la que uso el equipo de PicassoBot. Además la placa MEGA cuenta con mayor memoria flash (8 veces mayor) y tiene más SRAM (4 veces mayor) a la UNO. En la tabla 2.20 se



	Arduino Uno	Arduino Mega 2560	Arduino Micro
Price Points	$19.99-$23.00	$36.61 - $39.00	$19.80 - $24.38
Dimension	2.7 in x 2.1 in	4 in x 2.1 in	0.7 in x 1.9 in
Processor	Atmega328P	ATmega2560	ATmega32U4
Clock Speed	16MHz	16MHz	16MHz
Flash Memory (kB)	32	256	32
EEPROM (kB)	1	4	1
SRAM (kB)	2	8	2.5
Voltage Level	5V	5V	5V
Digital I/O Pins	14	54	20
Digital I/O with PWM Pins	6	15	7
Analog Pins	6	16	12
USB Connectivity	Standard A/B USB	Standard A/B USB	Micro-USB
Shield Compatibility	Yes	Yes	No
Ethernet/Wi-FI/Bluetooth	No (a Shield/module can enable it)	No (a Shield/module can enable it)	No

Figura 2.20: Tabla de diferencias entre Arduino Mega, Arduino Nano y Arduino Uno. Tabla tomada de [17].

presenta una comparación entre los modelos de Arduino más comunes. En ella se puede observar que el Arduino MEGA es el más potente de los tres, debido a que cuenta con mayor velocidad y mejores prestaciones.

2.8. PC

Dado que el sistema final involucra el uso de muchos recursos de diversos orígenes, que el sistema haya sido multiplataforma presentó serias dificultades.

Se tuvo la necesidad de realizar el desarrollo del software a ejecutarse en el bloque de procesamiento en Ubuntu 20.04. Esto se debe a que en la etapa de image processing se utilizan dos programas (un filtro Cannydevernay y un detector de ridges[3]) cuya implementación requiere ejecutar comandos desde una termina Linux. En particular, se requiere compilar y ejecutar código C utilizando el compilador GCC desde esta terminal. Debido a la modularidad de estos programas utilizados, se decidió no modificarlos y utilizarlos directamente desde Ubuntu, en las computadoras de las que se disponía.

Por otro lado, para algunas variantes del image processing, se necesita una cantidad significativa de memoria RAM, del orden de los 7GB disponibles.

De esta forma, la PC utilizada para ejecutar el software principal del proyecto (image processing y path planning) debe de tener un sistema operativo basado en Linux, y al menos 7 GB de RAM disponibles.

[3]Se explicarán en detalle estas técnicas en el capítulo 6.

2.9. Raspberry PI

Al momento de migrar el software desde una computadora a una Raspberry, primeramente se optó por la Raspberry PI 3 B debido a que es el modelo con el que se contaba. Una práctica habitual es que las Raspberries tengan un sistema operativo *Raspbian* [18] que se encuentra basado en Debian. Este sistema operativo esta optimizado para funcionar en equipos con CPU ARM. En la figura 2.21 se puede apreciar la forma que tiene la Raspberry. Esta cuenta con:

- procesador Broadcom BCM2837, Cortex-A53 (ARMv8) de 64-bits
- frecuencia de reloj de 1.2GHz
- memoria RAM de 1GB
- conexión de red ethernet 10/100 Gbps
- cuatro puertos USB 2.0
- micro SD
- Puertos propios de periféricos de Raspberry como pantalla táctil o cámara.

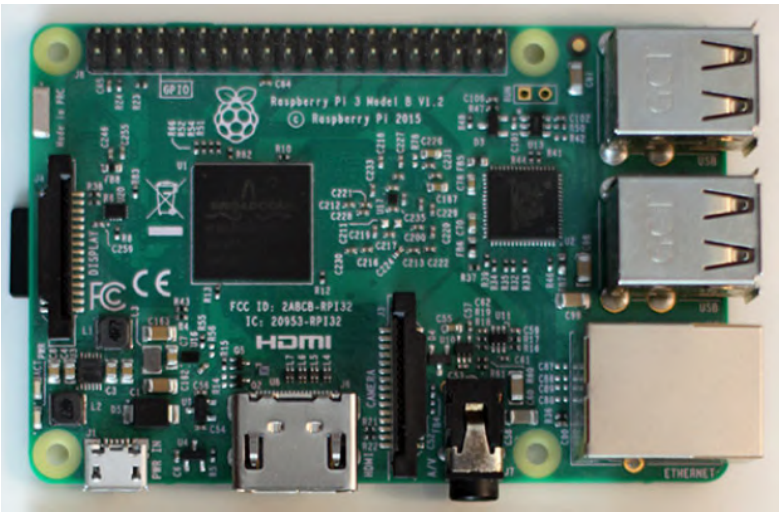


Figura 2.21: Raspberry Pi 3B.

Al probar los diferentes estilos, que se explican en la sección 6.3, se vio que la memoria RAM de la que dispone la Raspberry PI 3 no es suficiente para correr el procesamiento necesario para algunos estilos desarrollados. De esta manera se decidió migrar al siguiente modelo, la Raspberry PI 4, debido a que en este nuevo modelo cabía la posibilidad de tener hasta 8GB de memoria RAM. Esto se puede observar en la tabla 2.22. Esta placa tiene el mismo aspecto y dimensiones que su modelo antecesor y cuenta las siguientes especificaciones:

- procesador Broadcom ARM Cortex-A72 de 64-bi
- frecuencia de reloj de 1.5GHz
- memoria RAM de 1, 2, 4 o 8 GB
- conectividad bluetooth 5.0 y wifi
- conexión de red ethernet 10/100 Gbps
- 2 puertos micro HDMI
- 2 puertos USB 2.0 y 2 puertos USB 3.0
- micro SD
- Puertos propios de periféricos de Raspberry como pantalla táctil o cámara.

Si bien ambos modelos de Raspberry coinciden en la mayoría de sus características, la principal diferencia (y por la cual se decidió cambiar de modelo) es que la Raspberry PI 4 cuenta con 8 GB de RAM.

	Raspberry PI 4	Raspberry PI 3 Model B+
Procesador	Quad Core Cortex A-72 1,5 GHz	Quad Core Cortex A-53 1,4 GHz
Memoria RAM	1, 2, 4 GB LPDDR4	1 GB LPDDR2
USB	2 x USB 2.0 2 x USB 3.0	4 x USB 2.0
Alimentación	USB Tipo-C	microUSB
HDMI	2 x Micro HDMI	HDMI
Ethernet	Gigabit sin limitaciones	Gigabit hasta 300 Mbps
Wi-Fi	2,4 / 5 GHz	2,4 / 5 GHz
Bluetooth	5.0	4.2

Figura 2.22: Tabla de diferencias entre Raspberry Pi 3 y Raspberry Pi 4. Tabla tomada de [19].

Al comunicarse por SSH, se tuvo acceso a la terminal de la Raspberry (en este caso), y se pudo correr todos los scripts de forma remota. Las pruebas que se hicieron con SSH fueron conectándose ambos dispositivos a una misma LAN. Si se quisiera dejar la Raspberry para poder acceder de forma global habría que hacer una configuración más avanzada.

2.10. Cámara web

Como se mencionó anteriormente, un retrato se puede generar a partir de una imagen previamente guardada en la computadora, o mediante la adquisición con una cámara web. Para esta segunda opción, se optó por usar una cámara web modelo Logitech C270, como la que se muestra en la figura 2.23. Esta cámara ofrece una resolución máxima de 720 píxeles y corrección de iluminación automática. Se verá en el capítulo 6 que esta resolución es adecuada para el procesamiento de imágenes.

Figura 2.23: Cámara web Logitech C270 utilizada para la adquisición de fotos en tiempo real.

Capítulo 3

Arquitectura del Firmware

Como se mencionó en la sección 1.3, el objetivo del bloque hardware de interfaz es el de recibir los comandos obtenidos del bloque de procesamiento a través de una conexión física y enviarlos en tiempo real al brazo para que estos puedan ser ejecutados. Como se vio en el capítulo 2, este bloque está principalmente compuesto por un Arduino Mega y la electrónica necesaria para realizar las comunicaciones tanto con el bloque de procesamiento (cable USB - USB-B) como con el bloque de motores (circuito con buffers tri-state presentado en la sección 2.6). En este capítulo, se presenta el firmware desarrollado para que este Arduino pueda cumplir con los objetivos del bloque. Dentro del pipeline del sistema, este firmware se encuentra dentro de la etapa de control por firmware. Lo que es más, este conforma la totalidad de esta etapa.

A simple vista, puede parecer que el diseño de este firmware es sencillo, solo se precisa un intermediario que tome los mensajes recibidos desde el bloque de procesamiento, y los envíe al bloque de motores. Pero, al comenzar a implementar el mismo, es que surge una gran cantidad de decisiones a tomar. Entre otras, hay que definir: ¿Qué formato tendrán los mensajes recibidos desde el bloque de procesamiento? ¿Cómo se recibirán estos mensajes?, ¿Habrá comunicación desde el Arduino hacia el bloque de procesamiento?, ¿Cómo se procesarán los mensajes recibidos?, ¿Cuáles serán los posibles mensajes a recibir?, entre otras. En este capítulo también se abordarán todas las decisiones tomadas para implementar de forma satisfactoria el firmware de esta etapa.

3.1. Características generales

En primera instancia se debe definir con precisión cuál es el comportamiento que se quiere del firmware en cuestión. Con este comportamiento es con lo que luego se pueden responder el resto de las preguntas planteadas con anterioridad. En particular, el comportamiento más relevante es cómo se comunicará el bloque de procesamiento con el firmware.

Dado que se quiere que el firmware únicamente lleve a cabo comandos obtenidos en el bloque de procesamiento, se decide en una comunicación del estilo

maestro-esclavo. En particular, se toma la decisión de recibir de a un comando por vez en el Arduino. Cada comando recibido será posteriormente ejecutado y, cuando termine esta ejecución, el Arduino enviará un mensaje al bloque de procesamiento informando que terminó. Luego se quedará en estado de espera, hasta que el bloque de procesamiento envíe el siguiente comando. Esta lógica se puede ver en el diagrama de flujo de la figura 3.1.

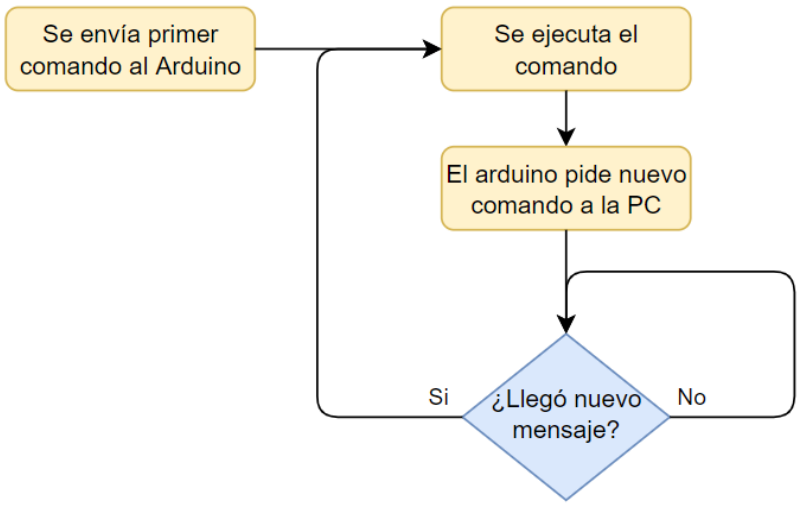


Figura 3.1: Diagrama de flujo inicial del firmware a desarrollar.

Se observa que aún no se ha restringido el tipo de comandos que se pueden llegar a ejecutar. En principio, el comando no tiene por qué ser envíado directamente al brazo, sino que también puede ser un comando de control interno hacia el Arduino. De esta forma se puede desarrollar una infinidad de algoritmos de control de bajo nivel del movimiento del brazo realizando configuraciones tanto a los motores, como al Arduino.

Considerando la cantidad de variables que se le puede, en principio, configurar a cada motor, y todos los algoritmos de control posibles a realizar, es que se precisó un firmware capaz de escalar en caso de querer agregar o quitar comandos para el desarrollo de nuevos algoritmos, o para la modificación de distintos parámetros de los motores. Es por esto que desde el principio del proyecto se desarrolló el firmware con una arquitectura de máquina de estados. De esta forma se tendría una versatilidad importante a la hora de elegir cuáles variables configurar y qué algoritmos de control implementar.

Por la cantidad de posibilidades que hay para realizar un control sobre el brazo, se decidió realizar en primera instancia un control sencillo desde el firmware. Luego se agregarían prestaciones adicionales si fuese necesario. Este control consistía en acotar los comandos a realizar a solamente dos posibilidades.

- Mover el brazo con la lapicera levantada.
- Mover el brazo con la lapicera apoyada, de forma de dejar un trazo en la hoja.

De esta forma se pueden realizar las curvas obtenidas en las etapas superiores como la concatenación de una cierta cantidad de movimientos con la lapicera apoyada. Además, al poder levantar la lapicera, también se deja la posibilidad de moverse de un punto a otro de la hoja sin realizar ningún trazo, pudiendo entonces realizar curvas disjuntas. El formato específico de estos mensajes se desarrolla en 3.2.

Se quiere además que, una vez el brazo termine de ejecutar el movimiento comandado, el Arduino sea capaz de pedir un nuevo comando al bloque de procesamiento. Para esto, una vez que estén en movimiento los motores, se consulta constantemente su registro **Moving**, el cual indica si el motor se sigue moviendo o no. Cuando los tres motores indiquen que dejaron de moverse, el Arduino es entonces capaz de pedir un nuevo comando al bloque de procesamiento.

Entonces, para la técnica de control utilizada, se puede cambiar el diagrama 3.1 por el nuevo diagrama de flujo 3.2.

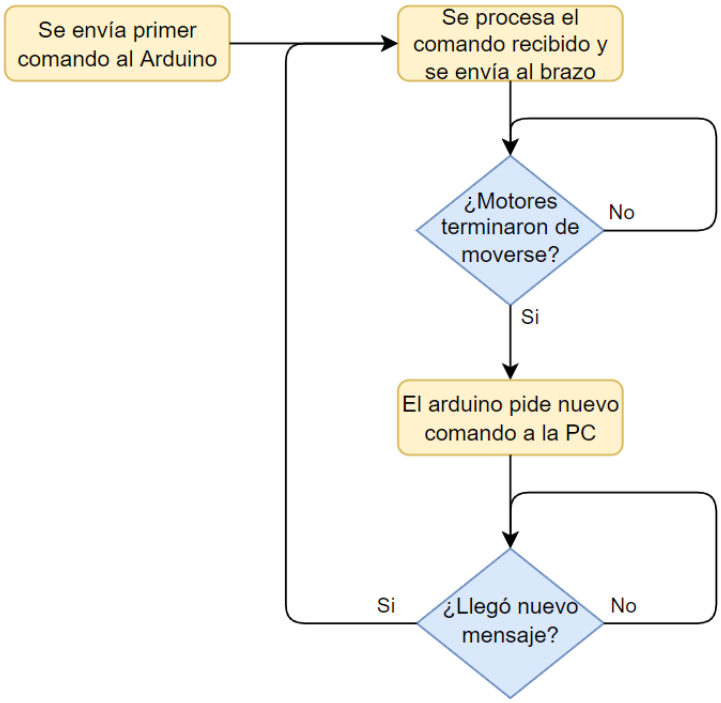


Figura 3.2: Diagrama de flujo del firmware a desarrollar, tomando en cuenta el control de envío de mensajes elegido.

Cabe mencionar que por limitar los comandos al motor solamente a movimientos, no se tiene un control en tiempo real de la configuración de los motores. No se puede ajustar el PID ni la **goal acceleration** en tiempo real. Estos valores simplemente se definen con experimentación (desarrollada en las pruebas del capítulo 7) y se configuran los motores en la inicialización del firmware.

Una particularidad de la implementación del firmware realizada es que esta no utiliza interrupciones. Esto se debe a que las interrupciones en Arduino no tienen manejo de prioridades, lo que complicaría y limitaría el uso de múltiples rutinas de atención a las interrupciones. De todas formas, la ventaja que tiene el trabajar con una placa Arduino es que hay una comunidad de gran porte y se puede encontrar mucho código sobre el cual apoyarse. En particular, los drivers para la comunicación entre el Arduino y los motores fueron encontrados en [20] por el equipo PicassoBot. Originalmente esta librería de drivers fue implementada para comunicarse con los motores modelos AX, pero se pudo usar sin mayores problemas para los motores modelo MX. La única modificación que se tuvo que hacer fue agregarle ciertos métodos adicionales para que contemplara registros que no existían (o que requerían otra manipulación) en el modelo AX.

3.2. Formato de mensaje desde la PC

La primera etapa de comunicación se da entre la PC y el Arduino, donde la PC envía los comandos que se procesaron para poder describir las curvas a dibujar. Para mover los motores se debe escribir en los registros de los mismos una posición objetivo y una velocidad a la cual moverse. En una secuencia de movimientos la posición inicial también es conocida, ya que es la posición final del movimiento anterior. De esta manera, un movimiento del brazo queda definido por una terna de ángulos y otra terna de velocidades. Son ternas porque de las cuatro articulaciones que tiene el brazo, son tres las que definen el movimiento en el plano (hombro, codo y muñeca). La articulación restante controla la punta, es decir la altura del lápiz. Esto también es importante al definir el movimiento, ya que se pueden realizar con el lápiz apoyado para dibujar o también con el lápiz levantado para trasladarse sin hacer trazos. Esta información también debe enviarse al Arduino y para ello se adopta la nomenclatura usada por el equipo PicassoBot: “PU” (Pen Up) para usar el lápiz levantado y “PD” (Pen Down) para usar el lápiz apoyado.

Se crea entonces un formato de mensaje entre la PC y el Arduino para enviar cada movimiento representado por un comando. Estos comandos llevan la siguiente información: si es lápiz levantado o apoyado, las ternas de ángulos con las posiciones finales y la terna de velocidades angulares de los motores. El proyecto PARRA también usaba esta misma información para describir los movimientos y generaba un archivo de comandos con un formato determinado. Se usó este formato para guardar los dibujos procesados con sus movimientos como una secuencia de comandos. Siendo así, el dibujo se puede volver a realizar enviando esa secuencia de comandos al Arduino. Para esto se hizo un script en Python que lee este tipo de archivos y codifica la información para enviar la secuencia hacia el Arduino. Este script también se encarga de establecer y encontrar el puerto serial para comunicarse con el Arduino. Una vez que el canal serial está activo se deben enviar los comandos en orden. En suma, se define el siguiente formato de comando para enviar en bytes:

$$PX \quad \omega_0\, \omega_1\, \omega_2 \quad \theta_0\, \theta_1\, \theta_2$$

donde:

- **PX**: Indica la posición del lápiz, usando X=U para indicar PenUp y X=D para indicar PenDown.
- ω_i: Indica la velocidad angular del motor i en las unidades del motor.
- θ_i: Indica la posición angular final del motor i en las unidades del motor.

Cada comando está conformado por catorce bytes. Los dos primeros caracteres se representan con su código ASCII por lo que usan dos bytes. El resto son valores numéricos a escribir en los registros de los motores. Estos valores son enteros ya que se convierten a las unidades que utilizan los motores. La velocidad puede ir de 0 a 1023 y la posición angular entre 0 y 4095 (en el caso de los MX). Entonces basta representar estos valores con enteros sin signo de 16 bits. Cada uno de los seis valores numéricos se envía mandando primero el byte alto y luego el bajo. Estos doce bytes conforman el resto del mensaje.

Cada vez que el Arduino recibe un comando, lo decodifica y envía la información a los motores para que ejecuten el movimiento. Una vez realizado el movimiento se envía a la PC una confirmación por puerto serial de que está listo para recibir el siguiente comando. De esta manera la secuencia de comandos se va ejecutando hasta completar el dibujo.

3.3. Implementación del Firmware

Ya se tiene entonces los drivers necesarios para la comunicación con los motores y el formato de mensaje que recibirá el Arduino desde la PC. Es entonces posible desarrollar un firmware que reciba los mensajes desde la PC, los traduzca al formato que reciben los drivers y los use para enviar el mensaje a los motores. Como se vio en la sección 3.1, se desarrolló un firmware con arquitectura de máquina de estados, en donde se espera el comportamiento explicitado en la figura 3.2. Puede verse el diagrama de estados de la implementación del firmware en la figura 3.3.

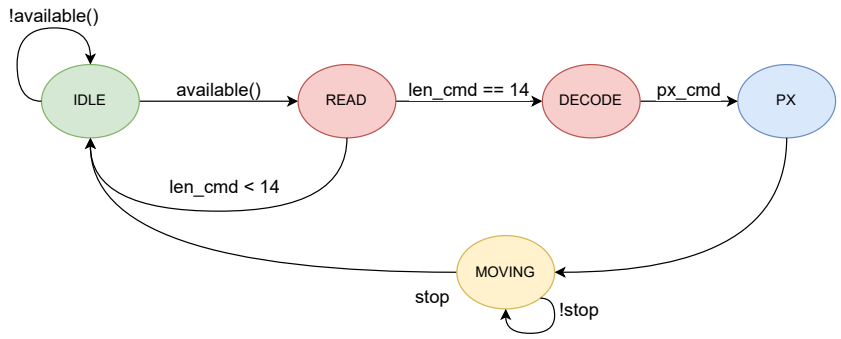


Figura 3.3: Diagrama de estados de la arquitectura firmware desarrollada.

En esta arquitectura se tiene un estado **IDLE** en el que se espera que llegue un caracter. La llegada de este caracter se verifica constantemente con la función *available()* del driver serial de Arduino.

En el estado **READ**, se almacena el caracter recibido, y se realiza el conteo de caracteres recibidos. Como los comandos tienen un largo fijo de 14 caracteres, si se reciben menos de esta cantidad, se vuelve al estado **IDLE** a esperar el resto. Cuando se reciben los 14 caracteres, se transiciona al estado **DECODE**.

En este estado de **DECODE**, como su nombre lo indica, se decodifica el comando recibido. Originalmente este estado estaba pensado para poder recibir comandos de PU, PD y además comandos de configuración para los motores. Pero como se decidió configurar los motores en la inicialización del firmware, esta característica quedó inutilizada. Simplemente se chequea que el primer caracter recibido sea una "P" para PU o PD, se define el próximo estado en consecuencia y se realiza una decodificación del comando recibido.

La decodificación consiste en traducir un valor numérico de dos bytes recibido (el valor de una posición o velocidad) a una sola variable de dos bytes. De esta forma dicho valor puede ser entendido por el driver utilizado.

Posteriormente, en el estado **PX**, se realiza el envío del comando recibido a los motores. Primero a cada uno de los motores se le envía su posición y velocidad objetivo y luego se les envía el comando de moverse a dicha posición con dicha velocidad. Además, es aquí que se realiza el algoritmo de sensado del torque explicado en 3.4.

Luego de enviar el comando de ejecución a los motores, se transiciona al estado **MOVING**, en donde se espera a que los motores dejen de moverse. Una vez todos frenan, se envía un comando al PC, indicando este fin del movimiento, y se vuelve al estado **IDLE**.

3.4. Algoritmo de sensado del torque

Una parte importante de la ejecución del dibujo es el momento de bajar el lápiz. La punta del brazo posee un motor que gira en un plano perpendicular al plano de dibujo para subir o bajar el lápiz. El equipo PicassoBot había fijado una fibra en la punta de modo que se giraba el motor de la punta hasta una posición conocida donde se sabía que la fibra quedaba apoyada. En este proyecto se decidió tener la posibilidad de cambiar la lapicera (o el elemento que se utilice para dibujar) colocando en la punta una abrazadera de compás, lo cual resulta muy práctico. El problema con esto es que cada vez que se coloca o cambia la lapicera, la posición de apoyo cambia. El motor de la punta debe tomar una posición angular diferente para poder apoyar.

Para solucionar esto se utilizó una de las prestaciones de los motores que es la medida del toque instantáneo ejercido, que se puede leer de los registros. Al ser comandado por el Arduino, cuando se envía la instrucción de bajar se empieza a leer también el torque. Cuando la punta está bajando el torque oscila en valores pequeños, pero cuando finalmente apoya y el motor sigue tratando de bajar, el

valor del torque empieza a crecer, tal como se ve en la figura 3.4. Entonces se detiene el motor cuando el torque supera un cierto valor umbral establecido.

De este modo, se puede fijar una posición de PenUp donde se asegure que el lápiz no va a estar apoyado e ir a esta posición cada vez que se quiere subir. Cuando se vaya a bajar el lápiz se baja utilizando este algoritmo de sensado del torque y no es necesario fijar la posición de PenDown para el motor de la punta.

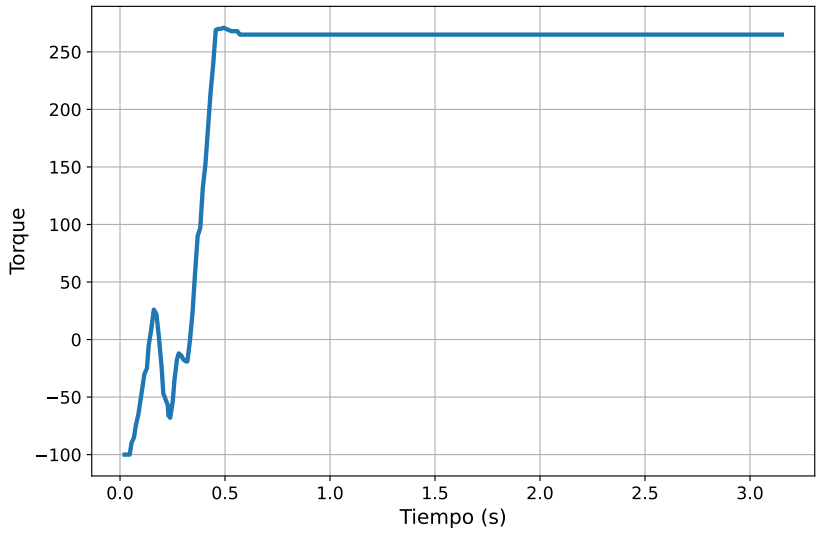


Figura 3.4: Gráfica del torque (en unidades discretas de los MX) en función del tiempo para el motor de la punta. Cuando se mueve en el aire el torque oscila en valores menores, nótese que al principio incluso es de signo opuesto porque se hace fuerza para mantener el motor en el aire. Cuando el lápiz apoya en la mesa de dibujo el torque crece considerablemente hasta que se detiene el motor para que no presione más.

3.5. Arquitectura alternativa: encolado de comandos

La comunicación entre la PC y el Arduino explicada en la sección 3.1 ejecuta el envío de comandos de a uno, es decir que para enviar un comando tiene que haberse terminado de ejecutar el anterior. Esto crea tiempos de manejo de datos entre cada movimiento y reduce la fluidez de los mismos. Para que los movimientos sean más continuos y los motores no se detengan entre cada comando se consideró un cambio en la arquitectura implementada, haciendo encolado de comandos y ejecutándolos sin tiempos de espera.

Para un movimiento dado, el lapso de tiempo en el que se mueven los tres motores es el mismo. Esto se verá en detalle en el capítulo 4. Este tiempo se puede calcular usando la diferencia entre posición final e inicial y la velocidad. La idea es que desde el Arduino, en vez de esperar a que los motores se detengan, se espere el tiempo calculado para el movimiento. La ventaja de esto es que la velocidad

es más continua porque no vuelve a cero cada vez que los motores se detienen. Una curva se define por un movimiento de PenUp hasta la posición inicial de la misma, seguido de una serie de movimientos PenDown donde se dibuja la curva con diferentes trazos continuos. Estos comandos de movimientos PenDown son los que deben encolarse en el Arduino para que sean ejecutados de corrido y ganar fluidez en el trazado de la curva. Entonces, para una curva, se envía un comando PenUp que posiciona la punta en el inicio de la curva, se envían todos los comandos PenDown para que el Arduino los encole y finalmente un comando que indica que se pueden empezar a ejecutar los comandos encolados.

Otro detalle que se tuvo en cuenta es que si una curva es muy larga se deben encolar muchos comandos, pudiendo llenarse la RAM del Arduino. Para evitar este problema se establece un tamaño máximo de cola. Si los comandos enviados alcanzan la cantidad máxima, se envía la señal de que se empiece a dibujar y el resto de los comandos se envían cuando se termine con la primera tanda.

Para enviar el tiempo, entonces, se debe cambiar el formato de mensaje para agregarlo. Además, con el fin cumplir con todos los cambios de esta arquitectura alternativa, se decide reformular el mensaje para que sea de largo variable agregando un carácter de fin de línea ('\r'). También se cambian los primeros dos caracteres por uno solo que indique el tipo de comando:

- **U**: Movimiento PenUp (mover el lápiz levantado). Este comando contiene la información de una terna de velocidades y posiciones angulares objetivo. No se envía el tiempo porque no importa el recorrido sino simplemente posicionar el lápiz en la posición inicial de la curva par empezar a dibujar.
- **Q**: Movimiento PenDown para encolar. En estos comandos se envían las posiciones finales, las velocidades y el tiempo del movimiento, previamente calculado en la PC. Cuando el Arduino recibe un comando de este tipo lo encola y espera el siguiente.
- **D**: Señal de bajar el lápiz y empezar a ejecutar los comandos de la cola (empezar a dibujar la curva). Este comando no tiene más información ya que es una señal de que no hay más comandos para encolar y deben empezar a ejecutarse.
- **E**: Señal de seguir ejecutando comandos en la cola (seguir con el dibujo). Este comando atiende el caso particular de que se haya alcanzado la cantidad máxima de comandos en una primera tanda. En ese caso se habría ya dibujado la primera tanda y luego de haber encolado la segunda tanda el lápiz ya está apoyado y lo único que se debe hacer es seguir ejecutando los comandos nuevos. Esto sirve para no enviar innecesariamente la instrucción de bajar el lápiz.

En la figura 3.5 se muestra un diagrama de la máquina de estados implementada en Arduino. Resumiendo, cuando se envía una curva para dibujar, el Arduino recibe un comando 'U' para moverse hacia el lugar indicado. Luego se reciben los comandos 'Q' que se deben encolar para dibujar la curva cuando llega el comando

'D' que indica bajar el lápiz. Cuando se apoya se empiezan a desencolar los comandos: el Arduino mueve los motores y espera el tiempo de cada comando para desencolar el siguiente y ejecutarlo. En el caso de que la curva hubiera pasado la cantidad máxima de comandos, otra tanda de comandos 'Q' llegaría seguida de un comando final 'E' que indica el final de la tanda y que se debe seguir con el dibujo.

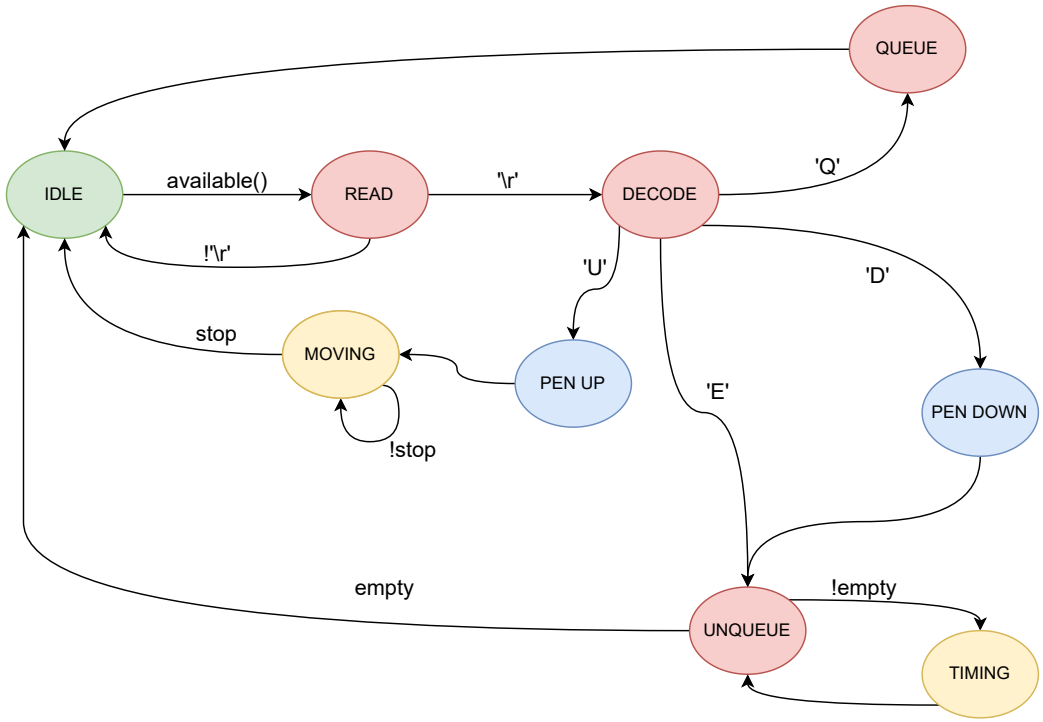


Figura 3.5: Diagrama de estados para la arquitectura alternativa. En esta arquitectura, primero se reciben todos los comandos de una curva y luego se ejecutan todos juntos midiendo el tiempo de cada movimiento.

Las pruebas con esta arquitectura implementada mostraron una mayor fluidez en el trazo pero también mucho menor precisión. Esto es así porque los motores no se detienen en cada posición, sino que van midiendo el tiempo de cada movimiento. Los trazos realizados no tienen puntos de quiebre, a diferencia de la otra arquitectura que paraba algunos instantes al final de cada movimiento. Pero el hecho de no exigir que pasara por los puntos de la curva hace también que haya mucha diferencia con el trazo esperado. Por estas razones, se decidió utilizar esta arquitectura solo como alternativa.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Modelado del brazo y el c-space

El tener un modelo matemático de un sistema físico es un elemento indispensable para lograr controlar este sistema. En este capítulo se presenta en detalle el modelo físico utilizado para el movimiento de los motores, junto con las herramientas matemáticas utilizadas para controlar este movimiento.

Este modelo y herramientas fueron posteriormente utilizadas para el desarrollo de los algoritmos contenidos en la etapa software de *path planning*. Es por eso que es útil construir intuición al respecto de dichas herramientas previo a entender la etapa de path planning. Asimismo, al ser la herramienta utilizada de carácter tan general, se obtuvieron a lo largo del proyecto resultados sobre esta herramienta más allá de lo realmente utilizado. Estos resultados pueden llegar a ser útiles en algún proyecto futuro de similares características a este. Por consiguiente, se añaden en el apéndice B.

4.1. Modelado del brazo

Como se mencionó en la sección 2.4, el brazo real está conformado por cuatro motores, acoplados secuencialmente como se puede ver en la figura 4.1. El extremo del brazo sostiene una lapicera, que al estar apoyada sobre la hoja y moviéndose, realizará los trazos que luego se convertirán en el dibujo. De esta forma, se puede abstraer el brazo real a una figura geométrica planar sencilla (de ahora en más, brazo), también mostrada en 4.1. Esta figura está conformada por tres articulaciones (con una de ellas fija), unidas entre sí por dos segmentos. También se tiene un tercer segmento que une la última articulación con lo que representa la punta de la lapicera. Las curvas que pueda realizar la abstracción de este extremo (o *tooltip*[1].) son las que luego se traducirán a comandos y serán trazadas por el brazo real.

[1] *tooltip* se puede entender como punta que sostiene la herramienta, término proveniente de la literatura robótica

(a) Brazo real. (b) Brazo real con sus articulaciones y segmentos. (c) Articulaciones y segmentos del brazo.

Figura 4.1: Imagen del brazo real, y su modelo geométrico.

4.1.1. Modelado geométrico del brazo

Luego de realizar esta abstracción del brazo, es conveniente dar nombres y definir un sistema de coordenadas sobre el cual trabajar, como se muestra en la figura 4.2. En primer lugar, se nombra la posición cada una de las articulaciones. Estas se llamarán $\mathbf{r}_0$, $\mathbf{r_1}$ y $\mathbf{r}_2$. Siendo $\mathbf{r}_0$ la posición de la articulación fija, e incrementando el valor del índice a medida que nos alejamos de esta primera articulación. Además, se le da el nombre $\mathbf{r}_3$ a la posición del extremo del brazo. Por otro lado, L_i será la longitud del segmento que une la articulación en $\mathbf{r}_{i-1}$ con la articulación en $\mathbf{r}_i$.

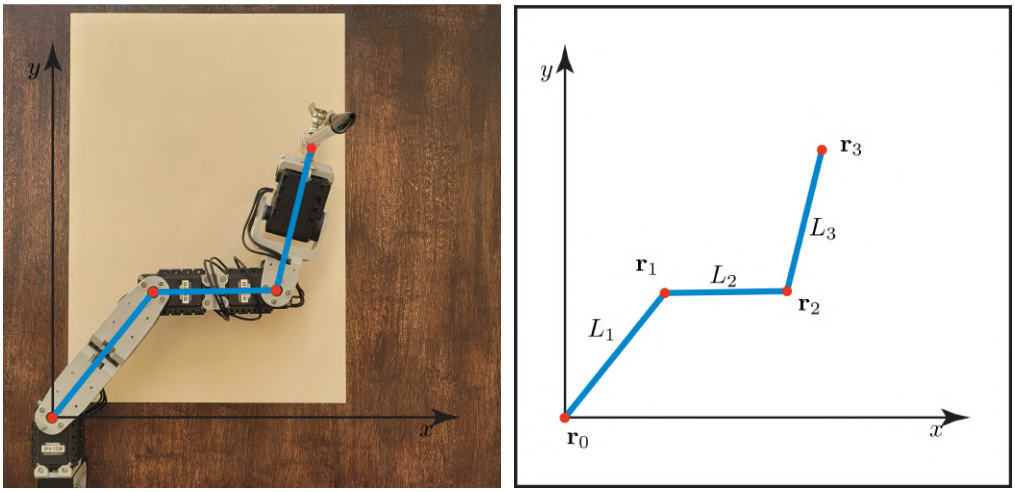
En segundo lugar, se define el sistema de coordenadas. Se considera un sistema cartesiano de coordenadas en el plano, de tal forma que:

- La articulación fija del brazo coincide con el origen de coordenadas O (es decir $\mathbf{r}_0 = (0, 0)$).
- El eje $\vec{Oy}$ coincide con el ángulo central del rango angular de la articulación fija.
- El eje $\vec{Ox}$ es perpendicular al $\vec{Oy}$ de forma tal que el producto vectorial $\vec{Ox} \times \vec{Oy}$ sea saliente al plano en la figura 4.2.

Con este plano definido, cada posición $\mathbf{r_i}$ se puede expresar de la forma: $\mathbf{r_i} = (x_i, y_i)$

A continuación se definen los ángulos θ, φ y ψ de forma tal que:

- Todos los ángulos se miden de forma antihoraria.
- θ es el ángulo entre el eje $\vec{Ox}$ y el segmento de longitud L_1.
- φ es el ángulo entre la prolongación del segmento de longitud L_1 y el segmento de longitud L_2.
- ψ es el ángulo entre la prolongación del segmento de longitud L_2 y el segmento de longitud L_3.



(a) Ejes definidos en referencia a la posición del brazo.

(b) Ejes definidos en el brazo geométrico.

Figura 4.2: Imagen de los ejes definidos para parametrizar el plano y la posición de cada elemento del brazo.

Es conveniente definir estos ángulos de esta forma, puesto que cada ángulo (θ, φ y ψ) es solidario al ángulo del motor que le corresponde (los motores en $\mathbf{r}_0$, $\mathbf{r_1}$ y $\mathbf{r}_2$ respectivamente). Para visualizar mejor estos ángulos, ver la figura 4.3.

De esta forma, se puede expresar la posición del *tooltip* en función de los ángulos de los motores. Esta fue obtenida en [2] y su expresión con el sistema de referencia aquí descrito es

$$\begin{cases} x_3 = L_1 \cos(\theta) + L_2 \cos(\theta + \varphi) + L_3 \cos(\theta + \varphi + \psi), \\ y_3 = L_1 \sin(\theta) + L_2 \sin(\theta + \varphi) + L_3 \sin(\theta + \varphi + \psi). \end{cases} \tag{4.1}$$

La expresión 4.1 se conoce en el mundo de la robótica como cinemática directa (del inglés *forward kinematics*). Esta se refiere a la expresión para obtener la posición del tooltip a partir de las posiciones (en este caso posiciones angulares) de los motores utilizados[2].

Para terminar con el modelado geométrico del brazo queda responder dos preguntas: ¿cuáles puntos del plano son alcanzables por el extremo del brazo? y ¿se puede trazar cualquier curva formada por puntos alcanzables de forma continua en los ángulos?

En referencia a la primera preguntas, para el brazo utilizado en este proyecto ($L_1 = 10.7$ cm, $L_2 = 8.3$ cm y $L_3 = 11.5$ cm), todos los puntos (x, y) del plano cartesiano definido con $\|(x, y)\| \leq L_1 + L_2 + L_3$ son efectivamente alcanzables. La prueba de esta afirmación puede verse en el apéndice B.

[2]Si a esta expresión aquí expuesta se le cambia el nombre de los ángulos de forma que $\theta_1 = \theta$, $\theta_2 = \varphi$ y $\theta_3 = \psi$, se puede llegar a expresiones más compactas y generales de *forward kinematics* para un brazo planar genérico.

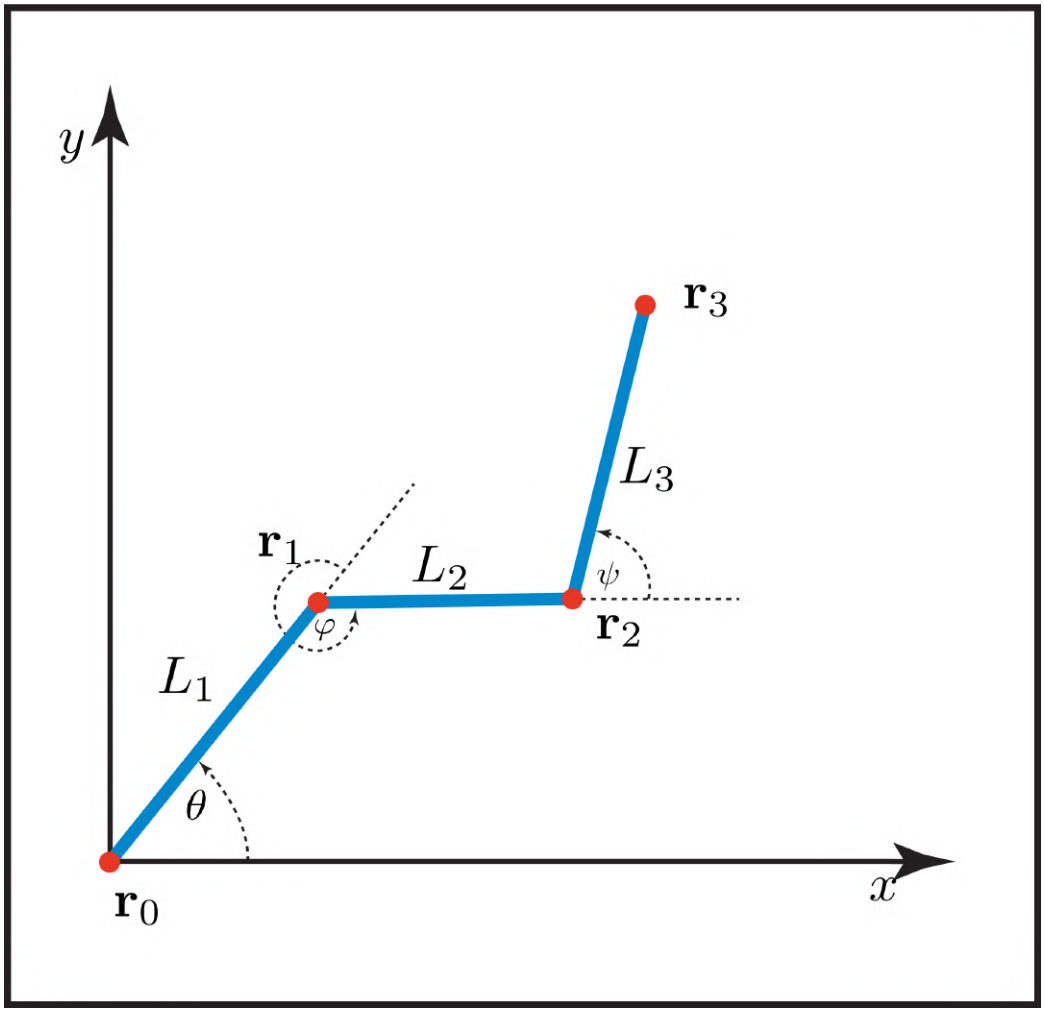


Figura 4.3: Brazo geométrico y su parametrización completa en el plano.

Además, la respuesta a la segunda pregunta es también afirmativa. Deriva de forma directa de la respuesta de la primera pregunta y también puede verse en el apéndice B.

4.1.2. Modelado cinemático del brazo

Por lo visto en la sección 2.3, se va a asumir que los ángulos de los motores se mueven únicamente a velocidad angular constante para simplificar el trabajo con el modelo de los motores.

Las desviaciones de este modelo se corrigen mediante la estrategia de control de los motores, en las etapas sucesivas del pipeline de este proyecto. De esta manera, la forma en la que varían los ángulos en función del tiempo es la expresada en (4.2).

A partir de esta ecuación, se puede ver que el movimiento de los ángulos

tiene cierta independencia con respecto a las velocidades angulares de los motores. Supóngase que los motores se mueven durante un intervalo de tiempo tal que $t \in [0, T]$ con $T > 0$. Si se toma el cambio de variable $s = \lambda t$ con $\lambda > 0$, entonces para un intervalo de tiempo $s \in [0, \frac{T}{\lambda}]$ se verifica (4.3). De esta forma, se pueden escalar las velocidades a valores que se considere adecuados, modificando la escala de tiempo a utilizar, y obtener el mismo trazo descrito por el tooltip. Esto es conveniente debido a que se puede ajustar las velocidades según las necesidades de los motores reales utilizados:

$$\begin{cases} \theta(t) = \omega_\theta t + \theta_0 \\ \varphi(t) = \omega_\varphi t + \varphi_0 \\ \psi(t) = \omega_\psi t + \psi_0, \end{cases} \tag{4.2}$$

$$\begin{cases} \theta(s) = \frac{\omega_\theta}{\lambda} s + \theta_0 = \omega'_\theta s + \theta_0 \\ \varphi(s) = \frac{\omega_\varphi}{\lambda} s + \varphi_0 = \omega'_\varphi s + \varphi_0 \\ \psi(s) = \frac{\omega_\psi}{\lambda} s + \psi_0 = \omega'_\psi s + \psi_0. \end{cases} \tag{4.3}$$

También a partir de (4.2), se puede observar que las curvas que son realizables por el brazo tienen una forma funcional cerrada, sustituyendo las expresiones de 4.2 en 4.1. Se obtiene entonces

$$\begin{cases} x_3 = L_1 \cos(\omega_\theta t + \theta_0) + L_2 \cos((\omega_\theta + \omega_\varphi)t + \theta_0 + \varphi_0) + \\ \qquad L_3 \cos((\omega_\theta + \omega_\varphi + \omega_\psi)t + \theta_0 + \varphi_0 + \psi_0), \\ y_3 = L_1 \sin(\omega_\theta t + \theta_0) + L_2 \sin((\omega_\theta + \omega_\varphi)t + \theta_0 + \varphi_0) + \\ \qquad L_3 \sin((\omega_\theta + \omega_\varphi + \omega_\psi)t + \theta_0 + \varphi_0 + \psi_0). \end{cases} \tag{4.4}$$

4.1.3. Restricciones en los ángulos

Hasta ahora, en el modelado, no se ha tenido en cuenta las limitaciones angulares de los motores. Cada uno de los motores del brazo real posee un rango de ángulos posibles en los cuales moverse. Este rango se debe a diversos factores. Entre ellos se encuentran:

- Los motores no pueden girar indefinidamente en cualquier dirección.
- Los brackets que unen dos motores limitan el movimiento de estos.
- La morfología total del brazo también puede hacer que al enrollarse, se disminuya el rango de ángulos posibles para algún motor (ver figura 4.4).

Esto ya limita fuertemente los posibles posiciones del brazo. Pero además, como es uno de los objetivos del proyecto que el brazo adquiera cierta naturalidad propia del humano, se limitan más aún los rangos de los motores. Esta limitación se realiza de forma que los ángulos de cada uno de las articulaciones ($\mathbf{r}_0$, $\mathbf{r_1}$ y $\mathbf{r}_2$) se correspondan de forma aproximada a los ángulos que pueden tomar las articulaciones reales de un brazo humano (hombro, codo y muñeca respectivamente). Modificando levemente los valores utilizados por el mismo motivo en [2], y

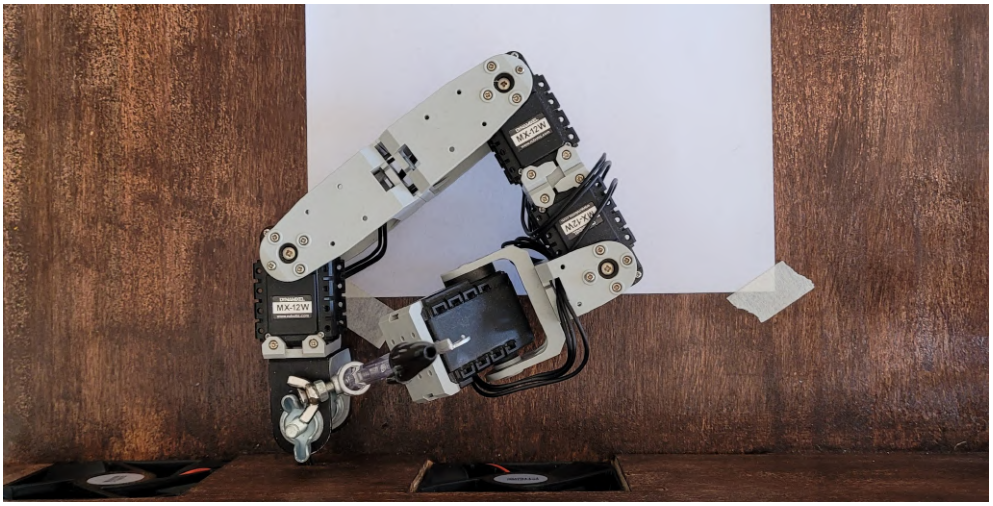


Figura 4.4: Foto del brazo enrollándose y limitando su propio movimiento.

adaptándolos a este proyecto, se toman los rangos: $\theta \in [60°, 180°]$, $\varphi \in [-150°, 0°]$ y $\psi \in [-90°, 30°]$.

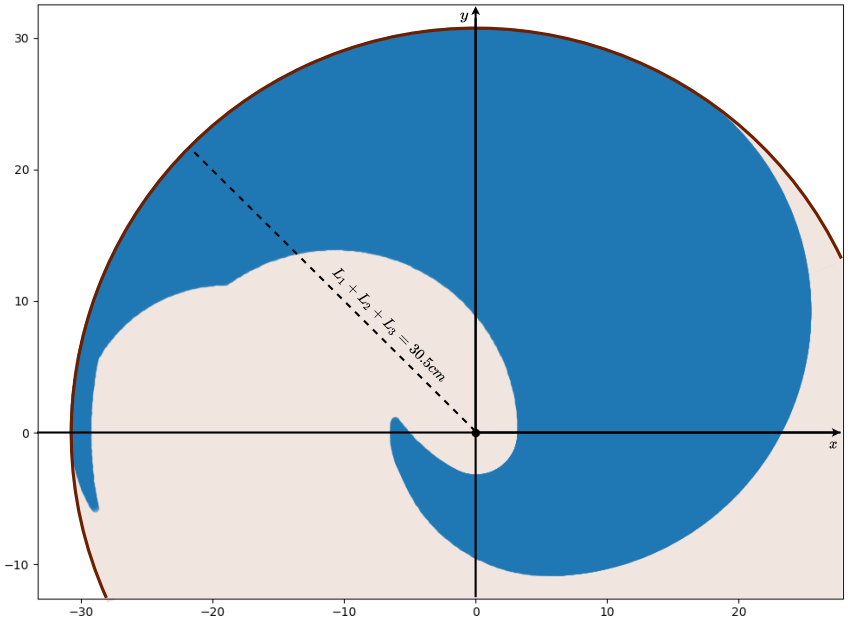


Figura 4.5: Región de posiciones alcanzables dadas las restricciones impuestas en los ángulos. Se muestra que esta región está contenida dentro de un círculo de radio $L_1 + L_2 + L_3$.

Cabe destacar que esta restricción en los ángulos corresponde a un brazo izquierdo, que es lo opuesto a lo ideado en [2]. Esta elección se tomó por varios motivos. El primero es que la disposición de las herramientas heredadas del equipo PicassoBot [1] es incompatible con un brazo derecho. En segundo lugar, un brazo

izquierdo es muy compatible con todo el pipeline del proyecto, que requiere de la definición de diversas referencias, tanto para el tratamiento de imágenes como para el path planning. Gracias a esta compatibilidad, es que se puede realizar el dibujo en el primer cuadrante del plano cartesiano previamente definido en este capítulo.

4.1.4. Discretización de los ángulos y velocidades

Por último, faltaría incluir en el modelo realizado el hecho de que la posición y la velocidad de cada motor está discretizada. En este proyecto se tomó la decisión de no modelar esto en los algoritmos de path planning, y en lugar de esto, realizar todos los cálculos asumiendo valores continuos para posiciones y velocidades de cada motor, y luego redondear los valores obtenidos para enviarlos a los motores.

De todas formas, se observó que ciertos efectos en los dibujos realizados por el brazo son debidos justamente a esta discretización, tanto en ángulos como en velocidades. Estos efectos son discutidos en la sección 5.5

4.2. El c-space

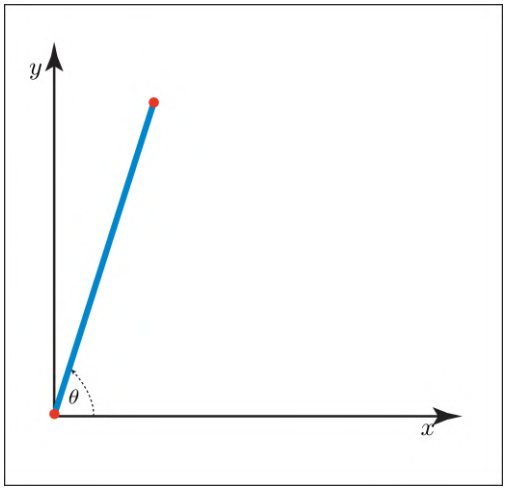
El c-space (del inglés, *configuration space*) es un concepto matemático ampliamente utilizado en la mecánica lagrangiana, termodinámica y, previsiblemente, en la robótica. Es un espacio abstracto en el que cada punto de dicho espacio se corresponde a una sola configuración del sistema. Más formalmente, es el espacio definido por los valores que pueden tomar las coordenadas generalizadas que definen el estado (o configuración) actual del sistema.

Las coordenadas generalizadas de un sistema se denotan usualmente como $(q_1, q_2, ..., q_n)$, donde n se relaciona con los grados de libertad de movimiento del sistema. Suelen expresarse cómo un único vector de la forma: $\mathbf{q} = (q_1, q_2, ..., q_n)$. Cada valor que pueda tomar el vector $\mathbf{q}$ es lo que se entiende como el *estado* o *configuración* del sistema.

Para concretar este concepto, se puede ver a continuación, en la figura 4.6, una cantidad de sistemas físicos, junto con su espacio de configuración (de ahora en más, c-space) a su lado.

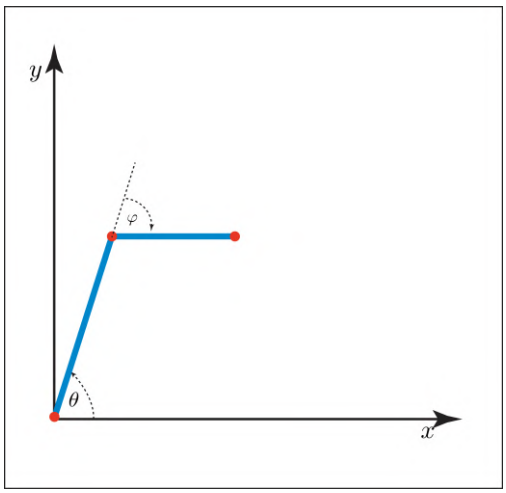
En particular, el c-space se puede utilizar para ver la dinámica de un sistema desde una nueva perspectiva. En lugar de pensar el movimiento de un sistema físico como una secuencia de "fotos" instantáneas del mismo, se puede visualizar como una única curva en el c-space. Esto se debe a que, como cada configuración del sistema es representable en el c-space, cada una de las posiciones que adopta en el tiempo también es representable en el mismo. De esta forma, uniendo los puntos correspondientes a esta secuencia de configuraciones en el tiempo, se obtiene finalmente una curva $\mathbf{q}(t)$ en el c-space[3].

[3]Aunque se utilice la palabra "unir" aquí, en principio la curva obtenida en el c-space puede ser discontinua. La topología de la curva obtenida en el c-space depende fuertemente del sistema a analizar y el movimiento que este esté realizando

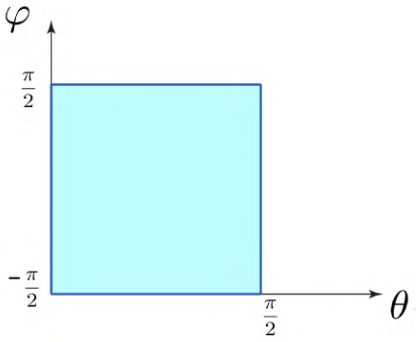


(a) Sistema con un grado de libertad angular.

(b) c-space correspondiente al sistema de un grado de libertad angular.



(c) Sistema con dos grados de libertad angulares.



(d) c-space correspondiente al sistema de dos grados de libertad angulares.

Figura 4.6: Imagen ilustrativa de varios sistemas junto con su c-space correspondiente.

De esta forma es que se tiene una correspondencia biyectiva entre curvas en el c-space y movimientos en el espacio real en donde existe el sistema. Entonces, en el caso de querer realizar un control sobre un sistema, se puede realizar el mismo pensando en los movimientos que realiza el sistema en su c-space en lugar de en su espacio físico real. Esto último fue lo que se realizó en este proyecto.

Sin embargo, como la topología de un c-space depende del sistema al cual representa, es conveniente desarrollar cierta intuición sobre cómo es esta topología antes de realizar un control del sistema en cuestión. Esto es lo que se realizará en las siguientes secciones.

4.3. El c-space del brazo planar de tres juntas

De acuerdo al marco de referencia mostrado en la figura 4.3, se puede ver que una configuración del sistema está completamente definida por el valor de los tres ángulos θ, φ y ψ. De esta forma, las coordenadas generalizadas que se toman para describir el sistema son: $\mathbf{q} = (\theta, \varphi, \psi)$. Y por lo tanto el c-space será (sin agregar aún las restricciones en los ángulos) el producto cartesiano: $[-\pi, \pi) \times [-\pi, \pi) \times [-\pi, \pi)$ [4]. Es decir, un espacio de tres dimensiones, donde cada coordenada puede tomar valores en $[-\pi, \pi)$. Véase la figura 4.7. Es por esto que por momentos en el proyecto se denominó este c-space particular como *espacio de ángulos*. En contraste con el plano en el cual dibuja el brazo, que se denominó *espacio x-y*.

Recuérdese que el c-space es conveniente a los efectos de este proyecto ya que los ángulos θ, φ y ψ definidos son solidarios a las posiciones de los ángulos de los motores.

Ahora que se tiene el espacio completo donde representar las configuraciones del sistema, es necesario ver cuáles configuraciones son las que se quiere traducir a este espacio. De esta forma se podrá realizar la traducción y controlar efectivamente los movimientos del sistema.

En el caso de este proyecto, es de interés obtener los ángulos θ, φ y ψ con los que el tooltip del brazo está en una posición (x, y) determinada. De esta forma se puede efectivamente comandar al brazo para que tenga su tooltip en la posición (x, y) deseada.

4.3.1. Cinemática inversa de un punto (x, y)

En el área de la robótica es un problema recurrente el tener un brazo robótico, formado por un cierto número de actuadores, y querer obtener la posición de los actuadores necesaria para que la punta de dicho brazo esté en una posición determinada. Este tipo de problemas se le denomina problemas de **cinemática inversa** [5]. Este es el proceso inverso a obtener la posición de la punta del brazo a partir de las posiciones de cada actuador presentado en la sección 4.1.2, o problema de **cinemática directa**[6].

En el caso de este proyecto, la cinemática directa del brazo planar queda definida por (4.1). De todas formas, la cinemática inversa es en general (y el brazo planar no es una excepción) un problema complejo, y el operador directo no es inyectivo. Es decir, hay muchos valores de ángulos que pueden producir que el tooltip esté en una misma posición (x, y). Esto se puede apreciar en la figura 4.8.

Sea Φ el ángulo total, definido como se muestra en la figura 4.9. Este se corresponde con el ángulo entre la horizontal y el último segmento del brazo. En [2] se dedujo el valor de los ángulos θ, φ y ψ del brazo a partir de la posición del tooltip (x_3, y_3) y el ángulo Φ. En este proyecto, se tomaron los procedimientos realiza-

[4] En realidad, el espacio de configuración en este caso es el conjunto: $T^3 = S^1 \times S^1 \times S^1$, siendo S^1 un círculo. Ya que cada coordenada es para el sistema 2π periódica

[5] del inglés, *inverse kinematics*

[6] del inglés *forward kinematics*

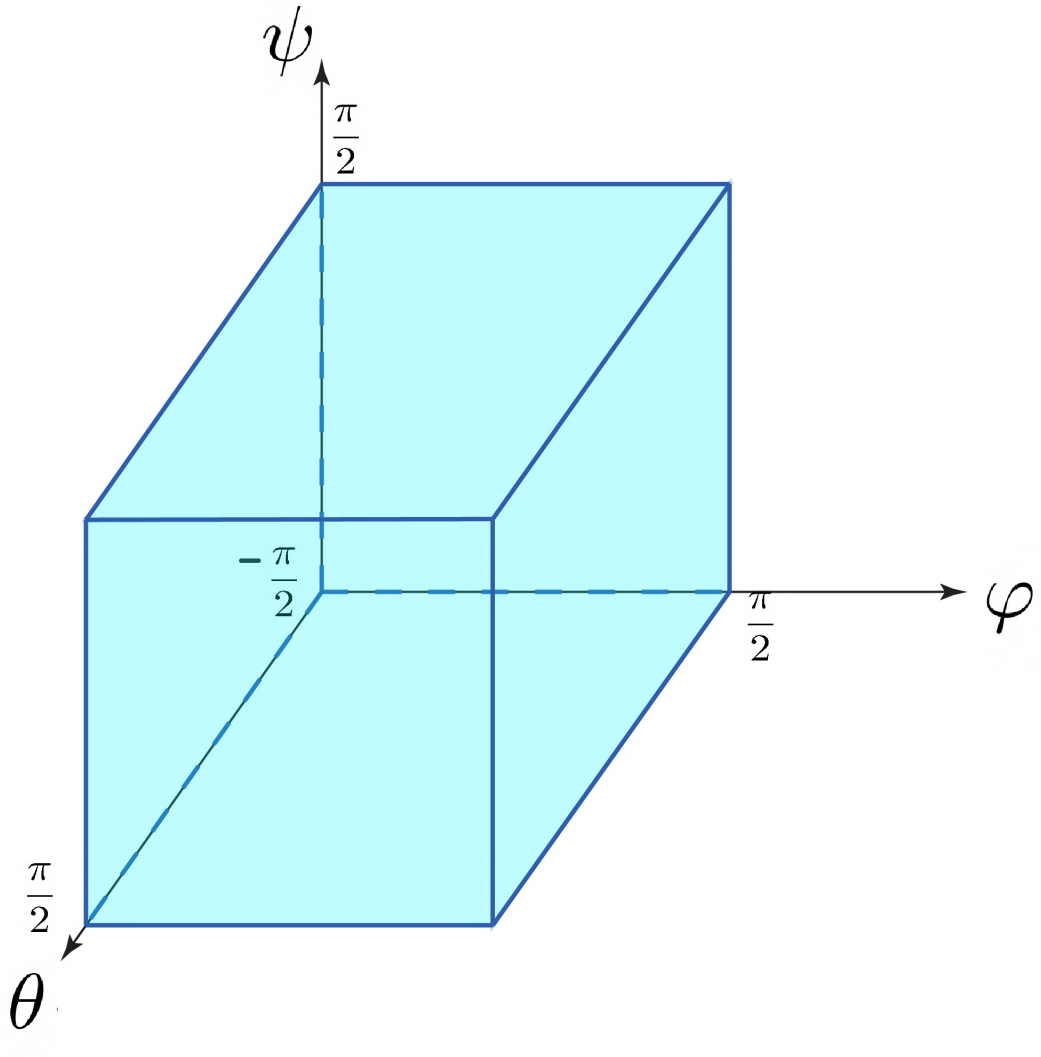


Figura 4.7: Imagen del c-space completo del brazo.

dos en [2] y se adaptaron para la nueva parametrización utilizada. Cabe destacar que, en principio, para cada (x_3, y_3) y ángulo Φ existen dos ternas de soluciones, $(\theta_d, \varphi_d, \psi_d)$ y $(\theta_l, \varphi_l, \psi_l)$. Estas ecuaciones vienen dadas por:

$$\begin{cases} \psi_l = \Phi - \theta_l - \varphi_l \\ \varphi_l = \alpha - \pi \\ \theta_l = \beta + \gamma, \end{cases} \tag{4.5}$$

$$\begin{cases} \psi_d = \Phi - \theta_d - \varphi_d \\ \varphi_d = \pi - \alpha \\ \theta_d = \beta - \gamma, \end{cases} \tag{4.6}$$

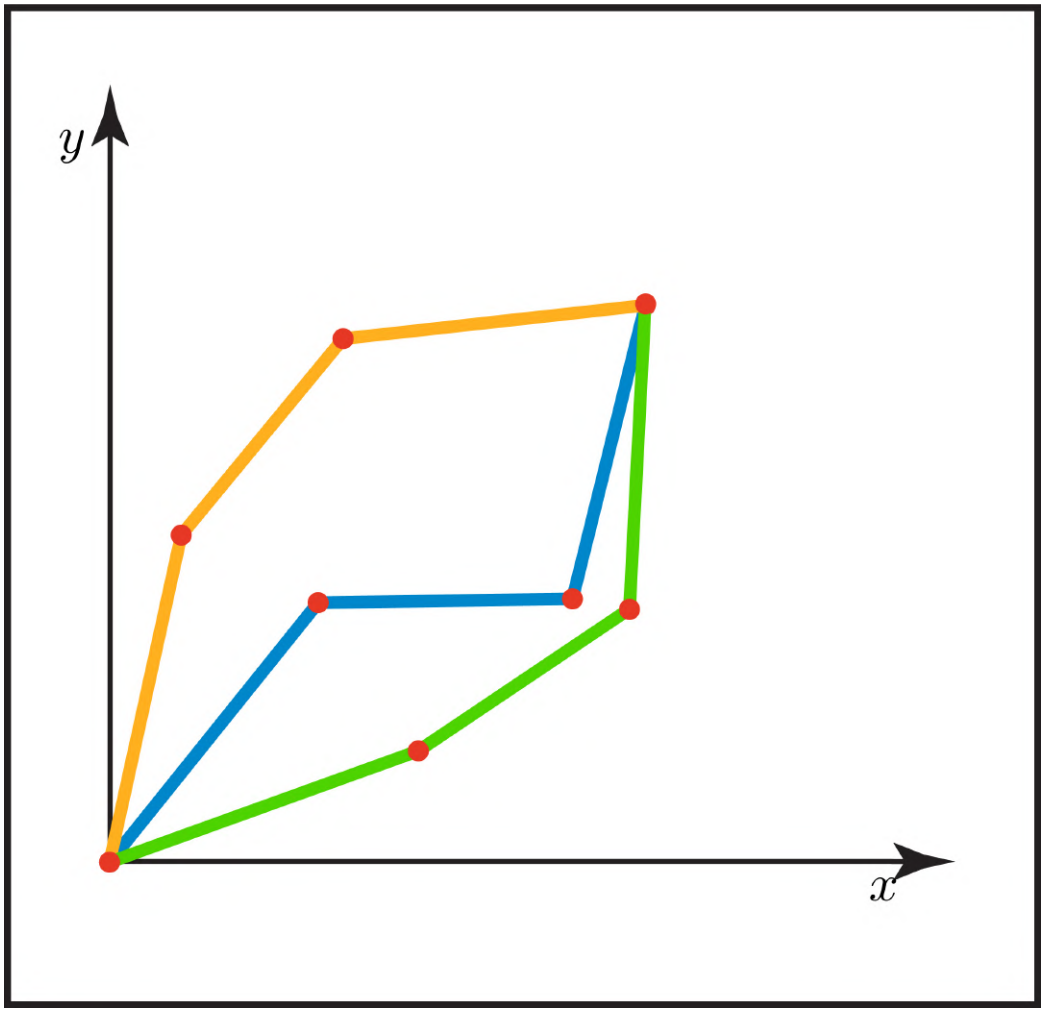


Figura 4.8: Configuraciones distintas del brazo para las cuales el tooltip queda en una misma posición.

donde se cumple:

$$\begin{cases} x_2 = x_3 - L_3 \cos\Phi \\ y_2 = y_3 - L_3 \sin\Phi \\ \alpha = \arccos\left(\frac{L_1^2+L_2^2-x_2^2-x_1^2}{2L_1L_2}\right) \\ \beta = \arctan\left(\frac{y_2}{x_2}\right) \qquad \text{(en el cuadrante apropiado)} \\ \gamma = \arccos\left(\frac{L_1^2-L_2^2+x_2^2+x_1^2}{2L_1\sqrt{x_2^2+y_2^2}}\right). \end{cases} \tag{4.7}$$

A partir de estas ecuaciones se pueden hacer tres observaciones. La primera es que el valor de los ángulos θ, φ y ψ quedan dependiendo de Φ, lo que significa que si se modifica Φ se obtienen distintas soluciones a un mismo problema de *inverse kinematics*.

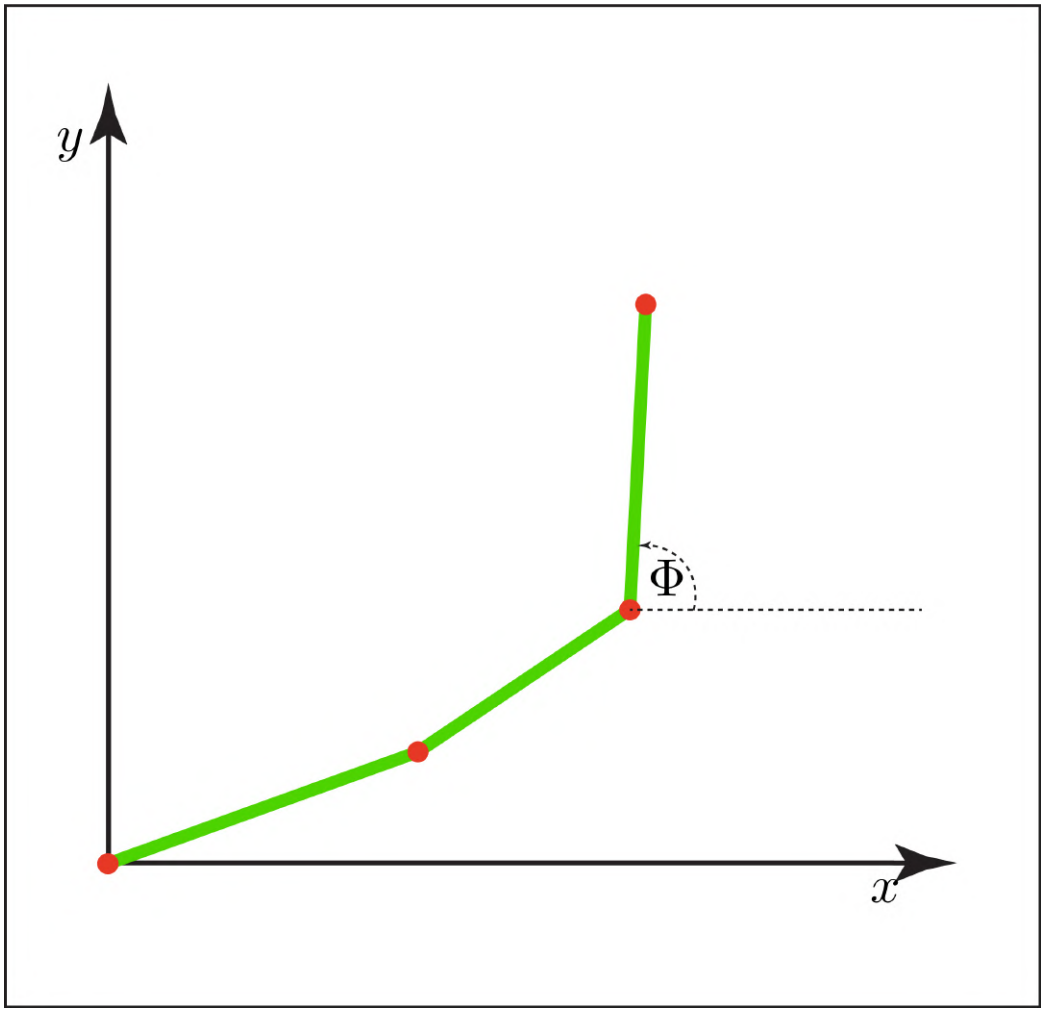


Figura 4.9: Definición del ángulo total Φ.

La segunda observación es que a partir del punto (x, y) y Φ, en general, van a surgir dos soluciones al problema de cinemática inversa del brazo. Estas dos soluciones para un mismo Φ son distinguibles por el signo de φ (si consideramos $\varphi \in [-\pi, \pi)$) y se llamarán *solución brazo izquierdo* y *solución brazo derecho* para $\varphi < 0$ y $\varphi > 0$ respectivamente. Este nombre se debe a la correspondencia entre el valor del ángulo y la posición natural del codo en caso de ser derecho o izquierdo. Esta correspondencia se ilustra en la figura 4.10

La tercera y última observación, es que estas ecuaciones no siempre tienen solución. Para que exista solución se deben de cumplir tres condiciones. Estas se deducen en el apéndice B y son las siguientes:

1. Que el punto (x, y) esté a una distancia del origen menor a $L_1 + L_2 + L_3$.

2. Que el ángulo Φ elegido haga que (x_2, y_2) esté a una distancia del origen menor a $L_1 + L_2$.

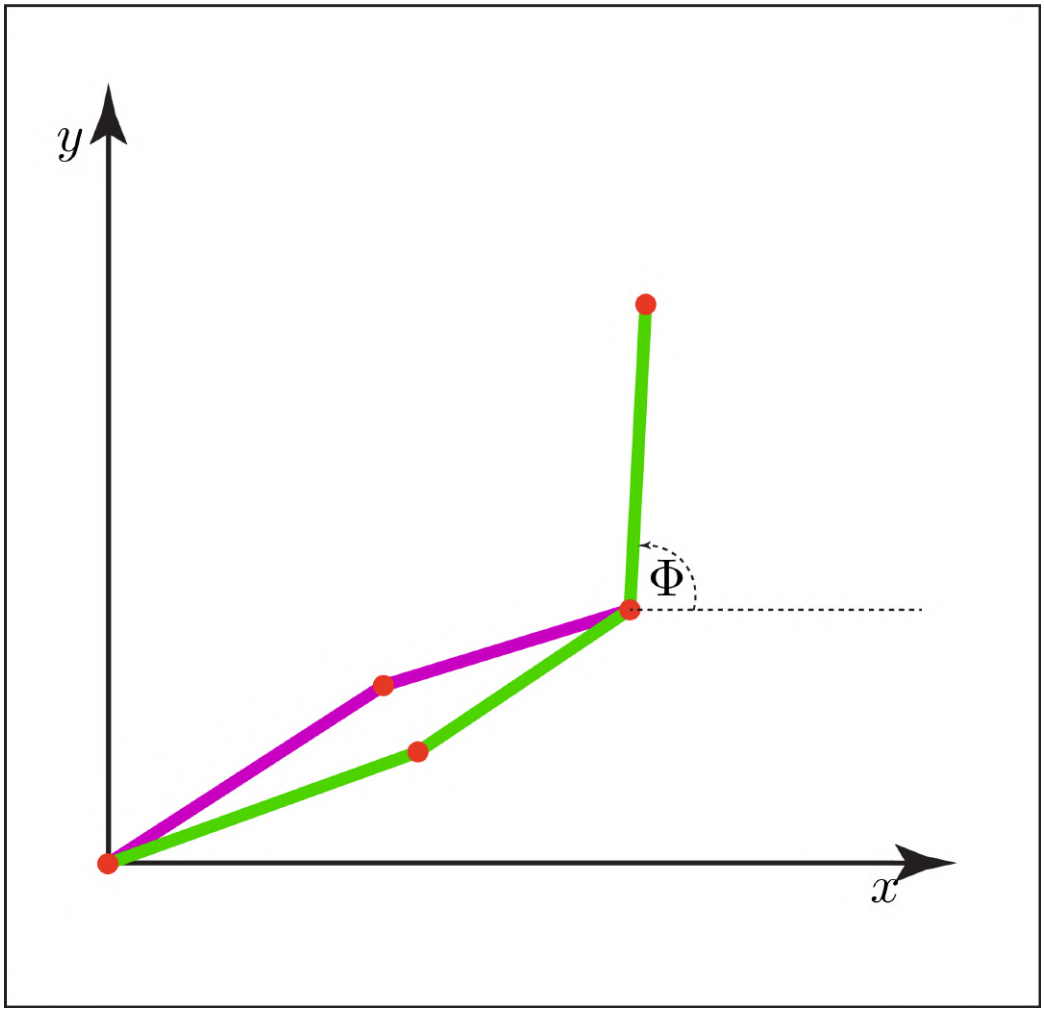


Figura 4.10: Ilustración de una *solución brazo izquierdo*(en violeta) y una *solución brazo derecho*(en verde) para un mismo valor de Φ.

3. Que el ángulo Φ elegido haga que (x_2, y_2) esté a una distancia del origen mayor a $|L_1 - L_2|$.

Como la solución del *inverse kinematics* queda dependiente de un parámetro, es intuitivo pensar que hay por lo menos una curva en el c-space (parametrizada en Φ) que da como resultado que el tooltip quede en la posición (x, y) deseada. Esto es, en efecto, como lo predice la intuición, y una justificación a esto se da en el apéndice B.

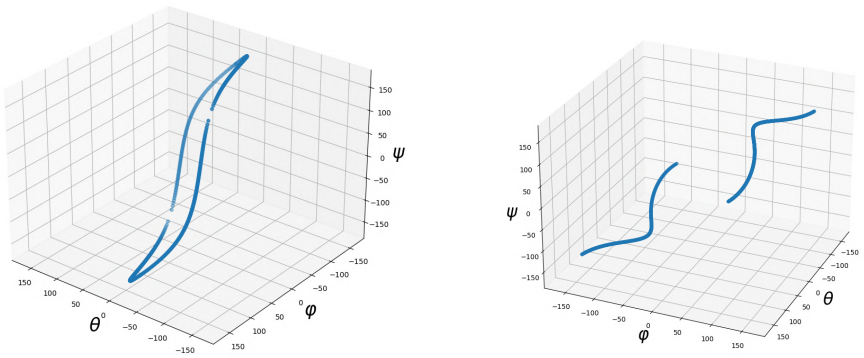
Lo que es más, la topología de dicha curva puede ser muy bien caracterizada. La prueba de esta caracterización se deja también para el apéndice B. Los posibles tipos de curva correspondientes a la cinemática inversa (para el caso de las medidas L_1, L_2 y L_3 utilizadas en este proyecto) son:

1. Una única curva cerrada,

2. Dos curvas cerradas.

Se recuerda que las curvas en el c-space, asociadas a una punto (x, y), son cerradas considerando que se encuentran en un espacio $2\pi-$periódico

Cada uno de este tipo de curvas puede verse en la figura 4.11.



(a) Inverse kinematics cuyo resultado es una curva cerrada simple.

(b) Inverse kinematics cuyo resultado son dos curvas cerradas simples.

Figura 4.11: Figura con las distintas topologías que puede adquirir una curva de inverse kinematics en el c-space para un punto (x, y).

4.3.2. Cinemática inversa de una curva $(x(t), y(t))$

Todo lo visto en la sección anterior es el desarrollo de a qué se corresponde un punto del espacio (x, y) en el c-space. Pero el objetivo del proyecto es trazar una curva en el espacio (x, y). Entonces, ¿Cómo se traduce una curva $(x(t), y(t))$ al c-space?

De forma poco rigurosa, se puede decir que la curva $(x(t), y(t))$ se traduce en una superficie en el c-space. Como las funciones que llevan del espacio (x, y) al c-space son todas continuas y quedan parametrizadas en Φ, entonces, al agregar otro parámetro t, se obtiene que el conjunto de ecuaciones 4.5 y 4.5 dependen ahora de dos parámetros. Lo que en definitiva es la expresión algebraica de una superficie. Un ejemplo de esto puede verse en la figura 4.12. La topología de dichas superficies dependerá de qué tipo de las curvas ilustradas en la figura 4.11 componen dicha superficie.

4.4. Trayectorias factibles del brazo en el c-space

Luego de todo lo anterior, puede parecer que involucrar al c-space en el cálculo de trayectorias es más un problema que una solución. En efecto, para ver cómo realizar un trazo en el c-space se tiene que tener un método para elegir el punto correspondiente en el c-space para cada una de las *inverse kinematics* de cada

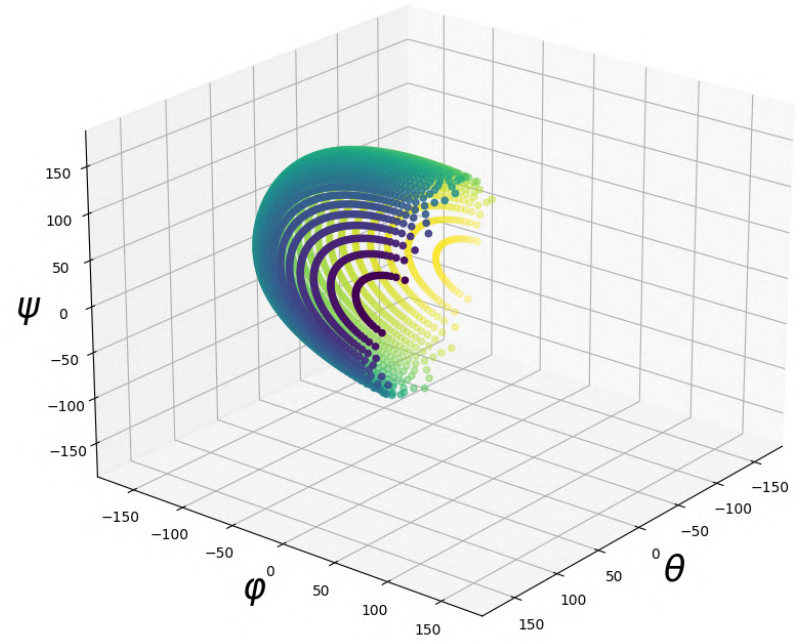


Figura 4.12: Superficie de ejemplo en el c-space que puede obtenerse a partir de una curva en (x, y). Cada punto en (x, y) es una de las curvas de nivel de la superficie en el dibujo. La superficie mostrada es solo la correspondiente a las soluciones *brazo izquierdo* del inverse kinematics.

uno de los puntos de la curva a seguir en el espacio (x, y). Pero, el siguiente es el motivo por el cual pensar el problema en el c-space ayuda a la resolución y a un buen modelado.

Hasta ahora solo se observó cómo traducir las curvas que se quieren seguir al c-space, pero falta ver cuáles son las curvas que el brazo efectivamente puede realizar en el c-space. Para ver cómo son estas curvas, basta con observar (4.2). Ahora, escribiendo esta ecuación en forma vectorial se tiene:

$$\begin{bmatrix} \theta(t) \\ \varphi(t) \\ \psi(t) \end{bmatrix} = \begin{bmatrix} \omega_\theta \\ \omega_\varphi \\ \omega_\psi \end{bmatrix} t + \begin{bmatrix} \theta_0 \\ \varphi_0 \\ \psi_0 \end{bmatrix}. \tag{4.8}$$

Lo que, de forma muy conveniente, es la parametrización de una recta en el espacio de ángulos (θ, φ, ψ). Este hecho se puede usar de forma muy provechosa para el cálculo de la trayectoria a seguir por los motores en el c-space. Sin importar cuál de todas las curvas posibles en el c-space se elija para representar una curva en el espacio (x, y), esta curva en el c-space se puede aproximar tanto como se quiera por rectas. De esta forma, el movimiento de los motores puede seguir a la curva original en el espacio (x, y) tan precisamente como se quiera.

Al tener un modelo de cómo es que se mueve el brazo en el espacio de configuración, hay muchas más libertades que se puede tomar al pensar algoritmos. Además, también los algoritmos que se pueden encontrar son computacionalmente más eficientes si se quiere aproximar curvas por rectas.

Como ejemplo de qué tanto más computacionalmente eficiente es, se puede comparar la simpleza de (4.8) con la complejidad de (4.4).

4.5. Restricciones adicionales traducidas al c-space

Lo favorable que tiene también el uso del c-space como herramienta de control, es que es sencillo traducir a este espacio las restricciones sobre el brazo. Se recuerda que las restricciones adicionales sobre el brazo son: una limitación en el rango de los ángulos de los motores y que las variables de posición y velocidad en realidad son discretas.

4.5.1. Restricciones angulares en el c-space

Se recuerda que las restricciones angulares en este sistema son: $\theta \in [60°, 180°]$, $\varphi \in [-150°, 0°]$ y $\psi \in [-90°, 30°]$. Esto implica que, en lugar de tener todo el espacio $[-\pi, \pi) \times [-\pi, \pi) \times [-\pi, \pi)$ de posibles configuraciones, se tiene que las configuraciones deben de pertenecer a $[60°, 180°] \times [-150°, 0°] \times [-90°, 30°]$. Gráficamente, equivale a un paralelepípedo embebido en el c-space. De esta forma, se pierde la periodicidad del c-space, es decir, que no se puede mover de forma continua de configuraciones con algún ángulo α a una con un ángulo $\alpha + 2\pi$.

4.5.2. Discretización en el c-space

La discretización de las posiciones se traducen simplemente a que en lugar de tener un espacio continuo, se tienen posiciones discretas en las cuales el brazo puede estar. De todas formas, el brazo si se mueve en el continuo del espacio. La discretización de los ángulos implica que el movimiento del brazo va a tener como objetivo de su movimiento uno de estos puntos. Por otro lado, la discretización de las velocidades lo que implica es que las pendientes de las rectas con las que se mueve el brazo en el c-space es un conjunto discreto. Por lo que nunca se podrá ir exactamente por la recta que se quiere, sino que se ira por una aproximación de esta.

Se recuerda que la discretización no se modeló explícitamente para resolver el cálculo de trayectorias, pero la discretización vista en el c-space sirve para analizar los efectos desarrollados en 5.5.

Capítulo 5

Path planning

En este capítulo se describen los algoritmos y estrategias utilizados para implementar la etapa software de *path planning*. Se recuerda que en dicha etapa se deben obtener los comandos a ser ejecutados por el brazo para dibujar las curvas obtenidas por la etapa de *image processing*.

El término de *path planning* procede del inglés y se puede traducir de forma literal como: "planificación de camino". Este término hace referencia al problema computacional de encontrar una secuencia de configuraciones válidas para mover un objeto de una posición inicial, a una posición de destino. Este problema es encontrado usualmente en las áreas de la geometría computacional, animaciones por computadora, en el desarrollo de videojuegos y, cómo no, en el área de la robótica.

En la etapa de *path planning* aquí desarrollada, se realiza un poco más que solamente el encontrar una secuencia de configuraciones válidas. Se elige aplicando un cierto criterio, de entre las infinitas trayectorias válidas, alguna que sea conveniente para seguir. Luego, realizando la traducción de esta trayectoria abstracta a una secuencia de comandos, para que estos sean ejecutados por el brazo real. Se buscaron estrategias para resolver este tipo de problema en la literatura, pero no se encontró ninguna propuesta lo suficientemente adecuada a las necesidades de este proyecto. Es por esto que algunas estrategias aquí utilizadas fueron propuestas originalmente en este proyecto.

5.1. Estrategia general

Se recuerda que la entrada a esta etapa está definida por la salida de la etapa software anterior de *image processing*. Esta salida está conformada por un conjunto ordenado de curvas $\mathcal{C} = \{\mathbf{C}_1, \mathbf{C}_2, ..., \mathbf{C}_n\}$, embebidas en el espacio (x, y). A su vez, cada curva $\mathbf{C}_i$ está conformada por un conjunto ordenado de puntos $\mathbf{C}_i = \left\{\left(x_1^{(i)}, y_1^{(i)}\right), \left(x_2^{(i)}, y_2^{(i)}\right), ..., \left(x_m^{(i)}, y_m^{(i)}\right)\right\}$.

Sobre esta entrada a la etapa de *path planning*, se asumen las siguientes hipótesis:

- El conjunto de curvas $\mathcal{C}$ ingresa con el orden[1] con el que se quiere dibujar las curvas.
- El conjunto de puntos que conforma cada curva $\mathbf{C}_i$ ingresa con el orden con el que se quiere recorrer dichos puntos en el dibujo. Además, no hay puntos duplicados.
- Todos los puntos correspondientes a cada una de las curvas son alcanzables por el brazo.

Teniendo un conjunto de curvas cuyos puntos cumplen con las condiciones previamente mencionadas, y utilizando las herramientas de modelado desarrolladas en el capítulo 4, se decidió separar la tarea de generar los comandos para cada una de las curvas $\mathbf{C}_i$ en tres etapas.

1. Traducir la curva $\mathbf{C}_i$ a una única curva $\mathbf{C}'_i$ en el c-space
2. Aproximar la curva $\mathbf{C}'_i$ en el c-space por las curvas realizables por el brazo en el c-space, es decir, por curvas de la forma expresada en (4.8).
3. Traducir cada una de estas rectas a comandos que puedan ser ejecutados por los motores.

Cada una de estas tres etapas será desarrollada en una sección del presente capítulo.

5.2. Encontrar la curva en el c-space

La estrategia general de este algoritmo implica transformar toda la información del plano al espacio de configuración. En otras palabras, las curvas del plano, que representan los trazos, se convierten a curvas en el c-space, que representan los movimientos del brazo. Se vio en el capítulo 4 que este es un problema sobredimensionado y que existen infinitas soluciones en el espacio de configuración para representar una curva del plano. Se quieren soluciones que aseguren la continuidad de los movimientos del brazo, de modo que al dibujar se tenga la mayor fluidez posible. Esto se logra haciendo que las posiciones angulares de los motores no tengan saltos o interrupciones, que en términos matemáticos es pedir que las curvas en el c-space sean continuas.

En formulaciones generales del problema de *inverse kinematics* (o también nombrado por su sigla, IK) se plantea como un problema de optimización. Mediante métodos iterativos se hallan configuraciones de ángulos que dejen el *tooltip* en la posición deseada. En la siguiente sección, se plantea cómo hallar estas soluciones minimizando una función de distancia.

[1] Distintos órdenes para las curvas fueron explorados en 6.4.

5.2.1. Técnica de optimización

Se toma el caso particular del brazo planar de tres juntas de este proyecto, para ver cómo se resuelve la cinemática inversa y cómo se hizo en este proyecto para llevar las curvas del plano al c-space.

Sea $\mathbf{r_O} = (x_O, y_O) \in \mathbb{R}^2$ la posición objetivo del *tooltip*, y sea la función de *forward kinematics* del brazo $FK : T^3 \to \mathbb{R}^2 \;/\; FK(\theta, \varphi, \psi) = (x_3, y_3)$ siendo (x_3, y_3) las coordenadas del *tooltip* dados los ángulos de las articulaciones, tal como fueron definidos en la ecuación (4.1). Lo que se busca son θ', φ' y ψ' tal que $FK(\theta', \varphi', \psi') = (x_O, y_O)$.

Se puede plantear entonces el problema de *inverse kinematics* como un problema de minimización con las restricciones impuestas en la sección 4.1.3:

$$\begin{aligned} \min_{\theta,\varphi,\psi} \quad & \|FK(\theta, \varphi, \psi) - \mathbf{r_O}\| \\ \text{r.a.} \quad & \theta \in [\pi/3, \pi] \\ & \varphi \in [-5\pi/6, 0] \\ & \psi \in [-\pi/2, \pi/6]. \end{aligned} \tag{5.1}$$

Se minimiza la diferencia del *tooltip* con su posición objetivo usando la norma euclídea en $\mathbb{R}^2$. Esta norma debería dar cero en caso de que la solución encontrada sea válida, pues esto implica que esa configuración de ángulos efectivamente deja el *tooltip* en la posición deseada.

Esta minimización se resuelve con métodos iterativos que usan el gradiente o estimaciones del mismo, tales como [21], [22], [23]. Para empezar a iterar se debe escoger un condición inicial. Se usa como condición inicial una configuración de ángulos que sea natural para el brazo, ya que se verá que no para cualquier configuración la iteración converge a una solución dentro de las restricciones. Pero lo importante es que al hacer la cinemática inversa de una curva completa la solución de cada punto pueda ser la condición inicial del siguiente. Esto es lo que va a garantizar que las soluciones para los puntos de las curvas en el plano sean continuas en el c-space. En palabras más simples, si se tiene una configuración que es solución para un punto en el plano, y el siguiente punto a resolver está cerca del anterior, entonces la configuración encontrada también está cerca de la siguiente solución.

Debe notarse que la solución hallada no es única sino que depende de las condiciones iniciales que se tomen. Lo que si se verifica es que la solución encontrada sea válida y no un mínimo local de la función. Esto implica evaluar la función que se minimiza en (5.1) con la configuración encontrada y verificar que la distancia de la punta del lápiz al punto objetivo sea muy cercana a cero (se define un umbral de 0.1 mm).

5.2.2. Regularización en los movimientos

Se vio que la solución encontrada para una curva (incluso un punto) no es única, sino que las hay infinitas. A la hora de optimizar se podría encontrar diferentes

soluciones que cumplan diferentes objetivos. Se consideró la idea de limitar los movimientos de las juntas, es decir, asignar un costo a mover ciertas articulaciones y fomentar el movimiento de las otras. Esto puede ser útil si se quiere imitar los movimientos humanos, donde el hombro suele moverse menos que el codo y la muñeca al dibujar. Otro objetivo podría ser mejorar los trazos, si se sabe que algunos movimientos son más imprecisos o ruidosos, se los puede limitar.

Para lograr limitar los movimientos, se pensó en el caso de los puntos sucesivos de las curvas, donde se usa como condición inicial la solución del punto anterior. Si se quiere penalizar el movimiento angular de alguna de las juntas hay que tratar de no cambiar ese ángulo al hallar la solución. Entonces se agrega un término de regularización a la función que se minimiza. Para cada uno de los ángulos se suma la distancia con respecto a la condición inicial, ponderada cada una por un peso. Se modifica el problema planteado en la ecuación (5.1) y se agrega esta regularización:

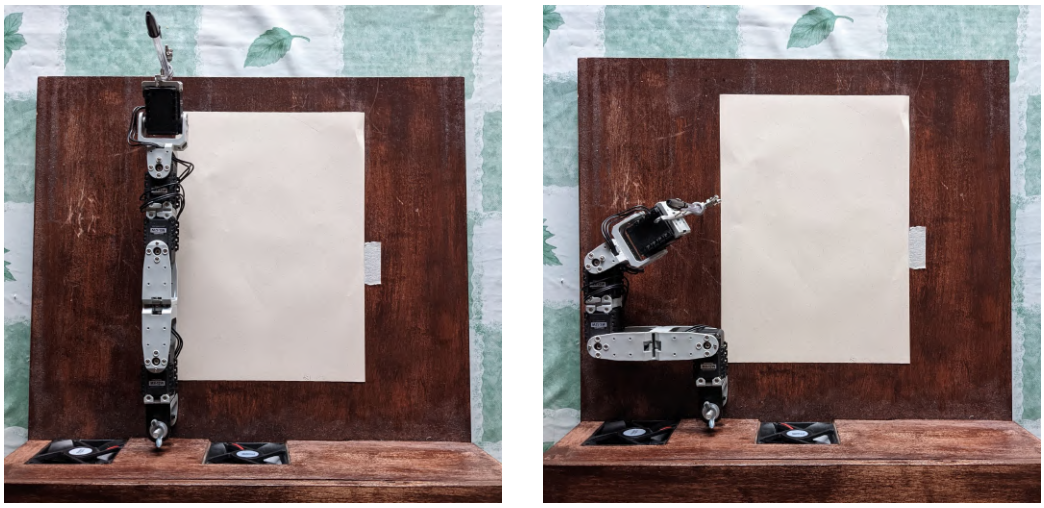
$$\begin{aligned} \min_{\theta,\varphi,\psi} \quad & \|FK\,(\theta,\varphi,\psi) - \mathbf{r_O}\| + \lambda_\theta(\theta-\theta_0)^2 + \lambda_\varphi(\varphi-\varphi_0)^2 + \lambda_\psi(\psi-\psi_0)^2 \\ \text{r.a.} \quad & \theta \in [\pi/3, \pi], \quad \varphi \in [-5\pi/6, 0], \quad \psi \in [-\pi/2, \pi/6] \end{aligned} \tag{5.2}$$

donde λ_θ, λ_φ y λ_ψ son los pesos para cada ángulo respectivamente y θ_0, φ_0 y ψ_0 son sus condiciones iniciales. Con los distintos valores de λ se estaría imponiendo cuáles juntas deben moverse más y cuáles menos. Mover de la posición actual las juntas con mayor peso asignado implicará más costo, por lo que se buscarán distintas alternativas al minimizar la función.

5.2.3. Convergencia de las condiciones iniciales

En el caso de la mayoría de los puntos, las condiciones iniciales para resolver IK estarán dadas por el punto anterior en la curva. Pero para los puntos iniciales de cada curva esto no es así. Se debe tomar una condición inicial que probablemente va a estar lejos de la solución. Lo cierto es que cualquier punto elegido dentro del radio máximo ($L_1 + L_2 + L_3$) tiene solución en espacio de configuración, tal como se explica en el capitulo 4. La condición inicial puede hacer que la optimización converja dentro de la región válida de ángulos, pero también puede terminar en la frontera tratando de llegar a una solución fuera de las restricciones.

La región escogida como válida en el espacio de ángulos representa posiciones que un humano puede lograr. Sin embargo, algunas posiciones no son naturales, al menos para dibujar. Inicialmente se utilizó el brazo estirado como condición inicial ($\varphi = 0$, $\psi = 0$), como se muestra en la figura 5.1a. En algunos casos pasaba que al resolver el problema de minimización se hallaba un mínimo en la frontera cuando había soluciones dentro de la región. Se buscó entonces usar inicializaciones más naturales o más similares a un brazo dibujando. Se estableció como condición inicial el hombro volcado hacia la izquierda y el codo y la muñeca hacia la hoja de dibujo, por tanto hacia la derecha (nótese que el brazo es zurdo), como se muestra en la figura 5.1b. En particular se usaron $\theta_0 = 180°$, $\varphi_0 = -90°$ y $\psi_0 = -60°$ como condiciones iniciales. Esto hizo que al solicitar resolver la cinemática inversa para un punto dentro de la hoja de dibujo siempre se hallara una solución válida.



(a) Brazo estirado (b) Configuración natural

Figura 5.1: Distintas configuraciones iniciales para resolver la optimización de IK. Una configuración natural conduce a los resultados que se esperan, mientras que iniciar con el brazo estirado puede llevar a resultados que no convergen dentro de las restricciones.

5.3. Aproximación por la poligonal

Como ya se vio en la sección 4.4, se decidió tomar un modelo del brazo en el que solo se puede mover por rectas a través del c-space. Es por esto que conviene aproximar la curva obtenida en la sección 5.2 directamente por rectas. De esta forma, se puede saber exactamente por cuáles configuraciones intermedias pasa el brazo al moverse de un punto origen a uno destino.

Es en este marco que surge la idea de pensar la curva $\mathbf{C}'_i$ como una poligonal (vease figura 5.2). Es decir, una curva formada por los segmentos de recta que unen los puntos en posiciones consecutivas de $\mathbf{C}'_i$. Llámese $\mathbf{P}'_i$ dicha curva poligonal y $\mathbf{P}_i$ a la representación de dicha poligonal en el espacio (x, y).

El plantear esta aproximación da una serie de importantes ventajas. Entre ellas está que los movimientos a realizar por los motores serán continuos para cada curva, por lo que no habrá que hacer un cambio brusco en los ángulos de los motores para unir dos puntos sucesivos. Esto aporta mucho a la naturalidad del trazado del dibujo. Otra ventaja es que las poligonales son objetos profundamente estudiados en el área de la computación gráfica, por lo que existe una gran cantidad de algoritmos que se pueden utilizar a posteriori para adaptar la poligonal a las necesidades del proyecto . Pero, por sobre todas las cosas, esta aproximación ya consiste de un conjunto de rectas las cuales pueden seguir exactamente los motores (al menos según el modelo obtenido en la sección 2.3).

De todas formas, cada recta que se utiliza para aproximar $\mathbf{C}'_i$ corresponde a un comando a enviar a los brazos. Lo que implica que si se utilizan muchas curvas para la aproximación, se deberá realizar una mayor cantidad de movimientos. A su vez, una mayor cantidad de movimientos implica un mayor tiempo de dibujado

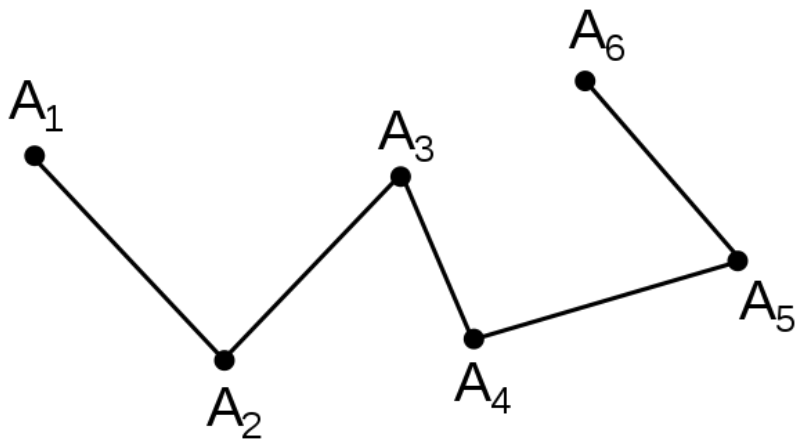


Figura 5.2: Ejemplo de un conjunto de puntos y la poligonal que estos definen. Imagen obtenida de [24].

de la imagen.

Por este motivo surge la necesidad de reducir la cantidad de rectas mediante las cuales $\mathbf{P}'_i$ aproxima la curva $\mathbf{C}'_i$ original. Pero, a su vez, no se debe de perder significativamente la precisión de $\mathbf{P}'_i$, ya que esto haría el dibujo final irreconciliable con el dibujo planeado.

5.3.1. Algoritmo de Ramer–Douglas–Peucker

Durante la realización de esta etapa, se encontró un algoritmo muy conveniente para la aproximación poligonal: el algoritmo de Ramer–Douglas–Peucker [25] (de ahora en más, algoritmo RDP o simplemente RDP). Este algoritmo, a grandes rasgos, lo que hace es, dada una tolerancia $\varepsilon \geq 0$, aproximar una poligonal genérica $\mathbf{P}$ por otra poligonal $\mathbf{p}$ de forma que:

- El conjunto de puntos de $\mathbf{p}$ es un subconjunto de los puntos de $\mathbf{P}$.
- La nueva poligonal $\mathbf{p}$ solo contiene los puntos más representativos de $\mathbf{P}$.
- Mientras menor sea la tolerancia ε, más se parecerá la poligonal muestreada $\mathbf{p}$ a $\mathbf{P}$. Además, si $\varepsilon = 0$ entonces $\mathbf{p}' = \mathbf{P}$

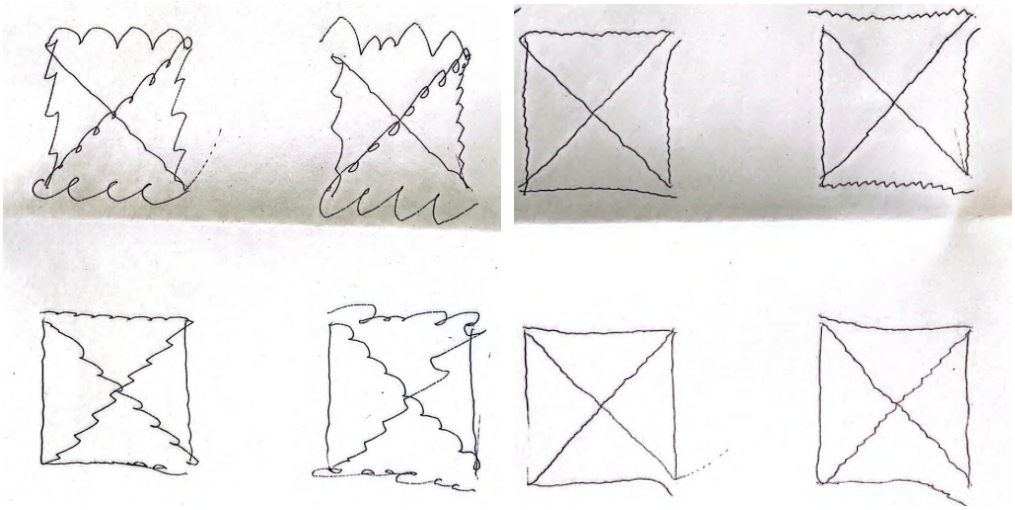
Además, se encontró una implementación en Python del mismo. Su documentación se puede ver en [26]. En el apéndice C se presenta un pseudocódigo de este algoritmo.

Se demuestra en [27] que este algoritmo puede ser implementado en $O(n \log n)$, siendo n la cantidad de puntos que conforma la poligonal original. Además, en [26] hay una visualización muy descriptiva del algoritmo.

5.3.2. Ajuste del parámetro ε

Como se mencionó previamente, el algoritmo de RDP depende de una tolerancia ε para realizar su aproximación. Esta tolerancia define qué tanto se parece la poligonal submuestreada a la original, y es por esto que debe de ser ajustada dependiendo de su aplicación. Mientras mayor sea ε, más se parecerá el submuestreo a la poligonal original y viceversa.

Para el caso de este proyecto, se vio que un solo valor de ε no era adecuado para realizar una aproximación correcta de la poligonal $\mathbf{P}'$ correspondiente a $\mathbf{C}'$. Para valores de ε grandes, la curva correspondiente a $\mathbf{p}'$ en el espacio real ($\mathbf{p}$) quedaba bien aproximada en algunas regiones, pero en otras no. Mientras que para ε pequeños, la curva $\mathbf{p}'$ quedaba bien representada en todas las regiones, pero los motores demoraban mucho en trazarla. Esto se puede ver en la figura 5.3[2].



(a) Dibujo realizado con un $\varepsilon = 0.05$. El dibujo se realizó en $150s$.

(b) Dibujo realizado con un $\varepsilon = 0.0025$.El dibujo se realizó en $270s$.

Figura 5.3: Un mismo dibujo de cuatro cuadrados con diagonales realizado con distintos valores de ε.

Esta diferencia se debe a la naturaleza no lineal de la función de *forward kinematics*. Esta hace que varíe mucho la precisión de las aproximaciones realizadas, dependiendo de la ubicación de las aproximaciones en el c-space. Si una curva es lo suficientemente larga en el c-space, esta puede requerir distintos valores de ε para ser bien representada y dibujada en un tiempo razonable.

Por esta razón surge la idea de realizar una implementación adaptiva del algoritmo RDP. Esta versión fue desarrollada por el equipo de este proyecto y se nombró *RDP adaptivo*.

[2]Los tiempos aquí expuestos y los del resto del capítulo no son los tiempos que demora el sistema en su versión final. Muchas optimizaciones sobre los mismos se hicieron luego de las pruebas aquí presentadas.

5.3.3. RDP adaptivo

La idea de la versión adaptiva del RDP es la de incorporar información del espacio real (x, y) en la realización del submuestreo, ya que esta no es incorporada en el algoritmo real. De esta forma se puede realizar una aproximación más fina en regiones donde así se requiera. Una descripción del algoritmo se puede ver en el pseudocódigo 1.

Algoritmo 1: Pseudocódigo del algoritmo adaptive-rpd.

Input : **P**: Poligonal en el c-space.
ε: ε del algoritmo RDP.
ε_{min}: mínimo ε a utilizar en el algortimo RDP.
d_{max}: máxima distancia en el espacio (x, y) a la que se quiere que estén los extremos de cada segmento de la poligonal.

Output: **p**: submuestreo de la poligonal original

```
adaptive-rdp(P, ε, ε_min, d_max)
    if (len(P) == 2) or (ε < ε_min) then
    |  return P
    end
    p = [p[0], p[1], ..., p[k]] = rdp(P, ε)
    result = [ ]
    for j = 1, ..., k − 1 do
    |  if dist(FK(p[j]), FK(p[j + 1])) > d_max then
    |  |  new_polyg = adaptive-rdp(P[p[j] : p[j + 1]], ε/2, ε_min, d_max)
    |  |  result.append(new_polyg[:,-1])
    |  end
    |  else
    |  |  result.append(p[j])
    |  end
    end
    result.append(p[k])
    return result
```

El algoritmo 1 consiste en realizar una primera aproximación **p** de la poligonal inicial **P** con un $\varepsilon = \varepsilon_0$. Luego, para cada segmento $\mathbf{p}[j], \mathbf{p}[j+1]$ de la poligonal muestreada **p**, se corrobora que sus extremos estén a una distancia menor a d_{max} en el espacio (x, y). Si estos puntos verifican esta condición, entonces formarán parte del submuestro final. Si no, el algoritmo 1 es aplicado recursivamente al tramo de la poligonal original cuyos extremos coinciden con los extremos del segmento $\mathbf{p}[j], \mathbf{p}[j+1]$ con un valor de $\varepsilon = \frac{\varepsilon_0}{2}$. De esta forma, se obtiene una poligonal muestreada con mayor resolución con respecto a la curva original.

Para la convergencia del algoritmo se agregan dos casos base. El primero es que para una poligonal formada por dos puntos (un solo segmento), el algoritmo termina directamente devolviendo dicha poligonal. Esto se debe a que, aunque estén a una distancia mayor a la deseada (d_{max}) no hay puntos de la curva original que estén entre éstos dos. El segundo caso involucra al ε utilizado. El algoritmo original RDP tiene la convergencia asegurada para cualquier ε, pero al realizar la adaptabilidad, esta cualidad se pierde para algunos casos. Si se tiene una poligonal formada por n puntos pertenecientes a una misma recta, para cualquier ε, el algoritmo original RDP dará como resultado la poligonal formada solamente por los extremos de la original. Luego, si estos extremos están a una distancia mayor a d_{max}, el algoritmo adaptivo se llamará recursivamente con la mitad del ε original, pero como para cualquier epsilon el algoritmo RDP devuelvo los extremos de esta poligonal, se llamará de nuevo recursivamente al algoritmo adaptivo y así ad infinitum. Es por este motivo que se decide pragmáticamente terminar el algoritmo, devolviendo toda la poligonal. Esto tiene el mismo efecto que, en este punto, llamar al algoritmo RDP con $\varepsilon = 0$.

Se pueden ver los resultados obtenidos con este algoritmo comparado con el muestreo RDP original en la figura 5.4

(a) Dibujo realizado solamente utilizando RDP con un $\varepsilon = 0.05$. El dibujo tardó en realizarse 17:00 min.

(b) Dibujo realizado con el algoritmo RDP adaptivo, con un $\varepsilon_0 = 0.05$. El dibujo tardó en realizarse 11:20 min.

Figura 5.4: Comparación del algoritmo RDP contra su versión adaptiva.

5.4. Interfaz con la siguiente etapa

El último paso del path planning es traducir toda la información sobre la trayectoria a comandos y valores que los motores puedan leer. El sistema de referencia establecido en la sección 4.1.1 no coincide con el de los motores. Siendo θ_M, φ_M y ψ_M los ángulos en los sistemas de referencia de cada motor, estos se pueden obtener a partir de la conversión siguiente:

$$\begin{cases} \theta_M = 90° + \theta \\ \varphi_M = 180° - \varphi \\ \psi_M = 180° + \psi. \end{cases} \tag{5.3}$$

En la figura 5.5 se ilustra cómo se ubican estos ángulos en la geometría del brazo.

Además de hacer este cambio de referencia en los ángulos, también se debe escalarlos a las unidades de los motores. El rango angular de los MX es de 360° y está dividido en 4096 valores. Entonces cada ángulo en radianes se debe multiplicar por $4096/2\pi$ y redondear para usar las unidades de los motores.

La otra información a traducir son las velocidades de los motores. Dada una poligonal que aproxima una curva en el c-space, es decir un conjunto de segmentos de recta continuos, se debe encontrar una parametrización de cada segmento. Sea un segmento de la poligonal definido por sus puntos de quiebre $\mathbf{p}[j], \mathbf{p}[j+1]$, usando (4.8) se puede parametrizarlo con $t \in [0, 1]$ como se muestra en (5.4).

$$\begin{bmatrix} \theta(t) \\ \varphi(t) \\ \psi(t) \end{bmatrix} = (\mathbf{p}[j+1] - \mathbf{p}[j])\, t + \mathbf{p}[j]. \tag{5.4}$$

Se asigna $\mathbf{p}[j] = \begin{bmatrix} \theta_0 & \varphi_0 & \psi_0 \end{bmatrix}$, el punto inicial del segmento. Luego se tiene que la parametrización evaluada en $t = 1$ es igual al otro punto de quiebre que define el segmento. Aquí se puede despejar el vector de velocidades, que resulta ser la resta entre el punto final e inicial del segmento de recta cuando se parametriza con $t \in [0, 1]$.

Luego se pueden reescalar las velocidades, pues solo importa la relación entre las mismas. Como se explicó en la sección 4.1.2, lo que pasa de fondo es que se reparametriza la curva pero el trayecto recorrido es el mismo, solo que en otro intervalo de tiempo.

Las velocidades halladas se escalan de modo que la mayor de ellas tome un valor máximo establecido para la velocidad de los motores. Las otras dos se escalan con el mismo factor manteniendo la relación. En este proyecto se utilizó 16 como velocidad máxima de los motores (en la unidades discretas de los motores, esto corresponde a $14.66 rad/s$). Esta decisión se explica en la sección 7.2.3. Cabe destacar que las velocidades calculadas son en rad/s y los motores tienen sus unidades discretas, pero no se hace una conversión de unidades. Esto es porque, como se explicó, no importa el valor mismo de velocidad si no la relación entre las mismas, en las unidades que sea. Por supuesto, esto también es válido porque la conversión de

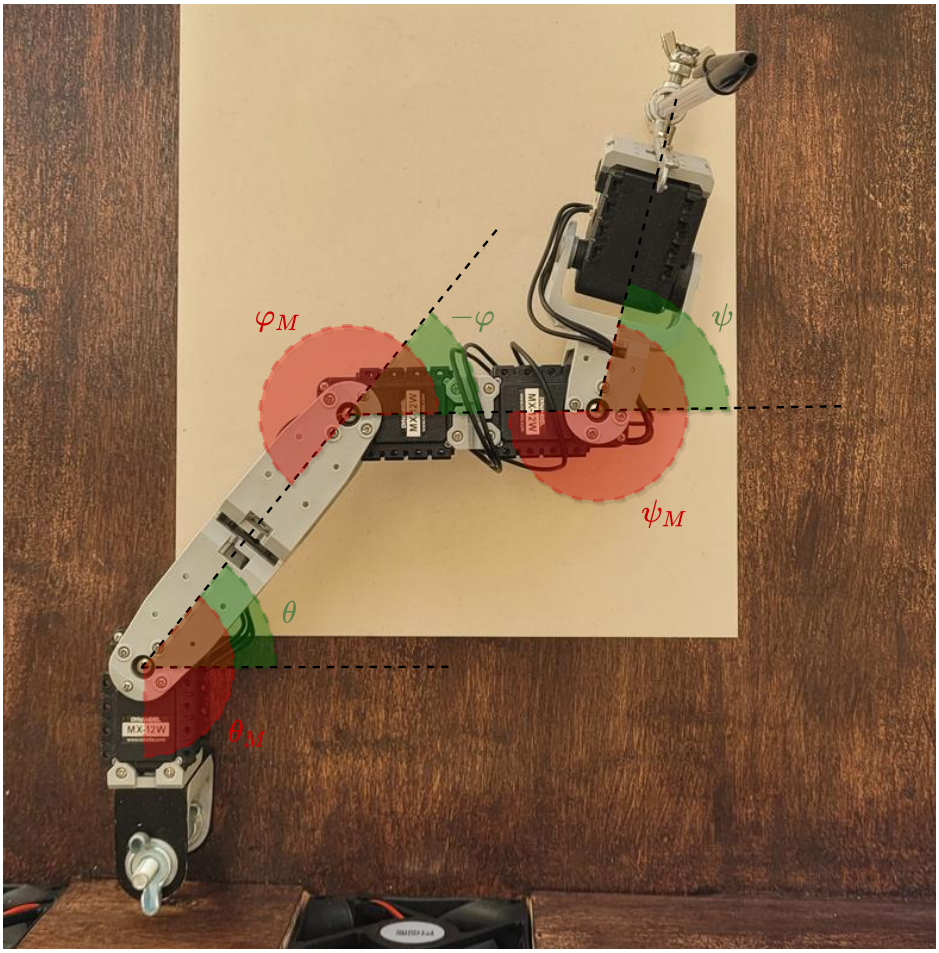


Figura 5.5: Representación en verde de los ángulos θ, φ y ψ del sistema de referencia usado en los cálculos, tal como se muestran en la figura 4.3. En rojo se muestran θ_M, φ_M y ψ_M que son los ángulos en la referencia de los motores, los cuales miden desde abajo en su vertical en sentido antihorario. En (5.3) se muestran las expresiones para convertir los ángulos a la referencia de los motores. Nótese que se muestra $-\varphi$ ya que, dadas sus restricciones, es más conveniente medir el ángulo en sentido horario y cambiarle el signo.

rad/s a los motores es lineal (cada unidad de velocidad de los motores equivale a $0.096 rad/s = 5.5°/s$).

Toda esta información recabada se utiliza para describir las curvas en comandos. Para una curva aproximada por una poligonal se tiene entonces lo siguiente:

- Una instrucción inicial de moverse con el lápiz levantado hasta la configuración de ángulos inicial de la curva. Este sería el primer punto de la poligonal. Las velocidades no importan para este movimiento, pues no importa la trayectoria.

- Una vez posicionado al principio de la curva se debe dar la instrucción al brazo de bajar el lápiz y empezar a dibujar. Se le indica que se mueva hacia

el siguiente punto de la poligonal con los motores moviéndose cada uno a las velocidades calculadas para ese segmento.

- Se continúa haciendo este proceso, manteniendo el lápiz apoyado, hasta llegar al final de la curva, pasando por todos los segmentos de la poligonal.

De esta manera, siguiendo el formato de comandos definido en la sección 3.2, una curva estaría conformada por una serie de comandos donde el primero es PenUp con posición destino el inicio de la poligonal y velocidad arbitraria (esta velocidad no se calcula en ningún momento, simplemente se definió una velocidad a la cual se mueven los motores cuando el lápiz va levantado). Luego un comando PenDown por cada segmento de la poligonal, con posición destino el punto final del segmento y sus velocidades calculadas. Escribiendo de esta forma los comandos se puede indicar a los motores que dibujen cada curva.

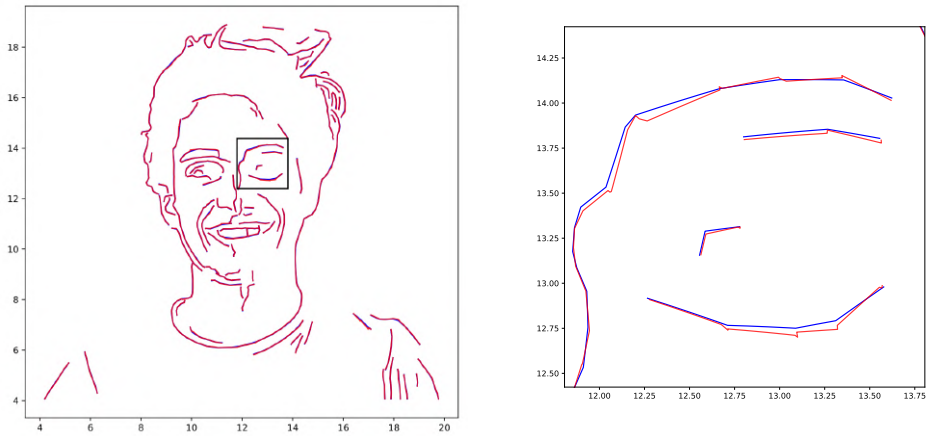
5.5. Efectos de la discretización

Hasta ahora se hizo una descripción teórica del path planning, pero no se tuvo en cuenta que los motores trabajan con un conjunto de valores discretos para representar las posiciones y velocidades angulares. Lo que se hace es hallarlas de igual manera y luego tomar el valor discreto más cercano. Esto conlleva un error de cuantización.

En el caso de la posición angular se tiene una apreciación de 0.088°, pues los MX discretizan los 360 grados en 4096 valores. El brazo estirado tiene un largo de 31.5 cm, por lo que con esta apreciación angular en el peor de los casos (el brazo completamente estirado) un error de 1 unidad en los MX (0.088°) implica un error de menos de medio milímetro ($\pi \frac{0.088°}{180°} \cdot 305mm = 0.47mm$).

Es en la velocidad que tiene un mayor efecto la discretización de los valores. Si bien los MX usan 1024 valores para la velocidad, la unidad corresponde a 0.916 rpm. En este proyecto se utilizó como velocidad máxima 16 (en unidades de los MX) lo que hace que solo haya 16 valores de velocidad a usar. Como se explica en la sección 5.4, lo que importa es la relación entre las velocidades y no tanto sus valores en sí. De todos modos, la discretización afecta, ya que las velocidades se escalan haciendo que la mayor de las tres sea 16. Las otras dos se escalan con el mismo factor para mantener la relación, pero además se deben redondear al natural entre 0 y 16 más cercano, para que sean un valor válido que el motor puede interpretar.

Al discretizar las velocidades deja de ser válido que los tres motores se mueven el mismo intervalo de tiempo. Unos terminan antes o después que otros porque al redondear las velocidades las relaciones se dejan de cumplir estrictamente. Ahora estos tiempos serán similares pero no exactamente los mismos, lo que hace que varíe la trayectoria teórica de la punta del brazo, tal como se ilustra en la figura 5.6. Se destaca que discretizar los ángulos también afecta esto mismo, pero en este caso tiene un efecto despreciable en comparación con la velocidad, que hace redondeos mucho mayores.



(a) Curvas simuladas (b) Zoom a la zona del ojo derecho

Figura 5.6: Simulación de los efectos que tiene discretizar la velocidad en un dibujo. En azul se muestran las curvas simuladas con los valores teóricos calculados de velocidad, mientras que en rojo se muestra la simulación con los valores discretos de velocidad y, por ende, tiempos diferentes. Se amplifica la región del ojo derecho donde se notan las diferencias.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Procesamiento de imágenes para extracción de curvas a dibujar

Un desafío crucial que presenta este proyecto es la representación de una imagen a través de trazos realizables por un brazo robótico. A partir de una imagen digital se deben extraer curvas que permitan plasmar su contenido. Estas curvas son una representación artística que no es única. En este capítulo se analizan los métodos ya propuestos por PARRA y también se introducen nuevas ideas para abordar este tema.

La solución que propone PARRA [2] usa algoritmos clásicos de procesamiento de imágenes. Estos algoritmos ponen atención en los bordes y en los niveles de intensidad o "sombras" de la imagen. En este proyecto se mantienen las ideas propuestas por PARRA pero se agregan nuevas opciones para representar la imagen. Teniendo más de una representación artística para una imagen es que se decide introducir el concepto de estilo para denominar a las distintas técnicas de extracción de curvas y a los resultados que se logra con cada una. Los algoritmos aquí propuestos combinan tecnologías más nuevas como redes neuronales convolucionales (CNN) con algoritmos clásicos como los ya utilizados en la solución actual.

Si bien PARRA funciona para imágenes de carácter general, se centra más que nada en retratos. Algunos de los estilos implementados son de uso exclusivo para retratos (como lo indicará su nombre "portrait"). Esto se debe a que al utilizar modelos de Machine Learning algunos están entrenados específicamente para caras y no ofrecen buenos resultados para otro tipo de imágenes.

6.1. Algoritmos anteriores

Como se nombró anteriormente, se cuenta con una solución implementada por PARRA para la extracción de curvas. Se divide el proceso en dos etapas. En primer lugar, una detección de bordes donde se puedan representar las principales formas de la imagen. En segundo lugar, una etapa de sombreado para marcar las regiones más oscuras de la imagen. Se destaca que tanto para la detección de bordes como para el sombreado se utiliza la imagen convertida a escala de grises. A continuación

se hace una breve descripción de dichos algoritmos y los resultados que obtienen. Por información más detallada véase la documentación del proyecto PARRA [2].

6.1.1. Detección de bordes

Para la detección de bordes se utilizó el algoritmo de Canny-Devernay. Este algoritmo está implementado en C por Gregory Randall y Rafael Grompone, presentado en IPOL [28]. Como resultado se obtienen distintos conjuntos de puntos con precisión de subpíxel que representan los bordes de la imagen.

Este algoritmo de detección de bordes cuenta con una etapa inicial donde se filtra la imagen con un kernel Gaussiano. Esto suaviza la imagen y quedan presentes solo los bordes más importantes. Este efecto está directamente relacionado con la desviación estándar σ del filtro. Cuanto más detalles se quieran eliminar, mayor debe ser σ. Luego se calcula el gradiente de la imagen y se hallan los bordes usando un umbral superior e inferior. Los máximos locales del gradiente se consideran bordes si superan el umbral superior. El segundo caso para ser considerado un punto del borde es superar el umbral inferior estando conectado a otro punto del borde.

PARRA probó distintos valores para estos tres parámetros (desviación estándar del filtro, umbral superior e inferior) llegando a un conjunto de valores que consideraban óptimos (3, 5 y 4 respectivamente). Estos valores son válidos sólo para imágenes de cierta resolución. Con imágenes de resolución de entre 512×512 y 1024×1024 píxeles estos valores funcionan correctamente. Cuando se probó con imágenes más grandes (de hasta 3000×3000) se detectaron demasiados detalles y los bordes perdían continuidad. Por esta razón se optó por cambiar el tamaño de cada imagen a procesar a 720 píxeles (el lado más grande) manteniendo la relación de aspecto, de modo que el algoritmo funcione como es esperado. Estos casos se pueden apreciar en la figura 6.1.

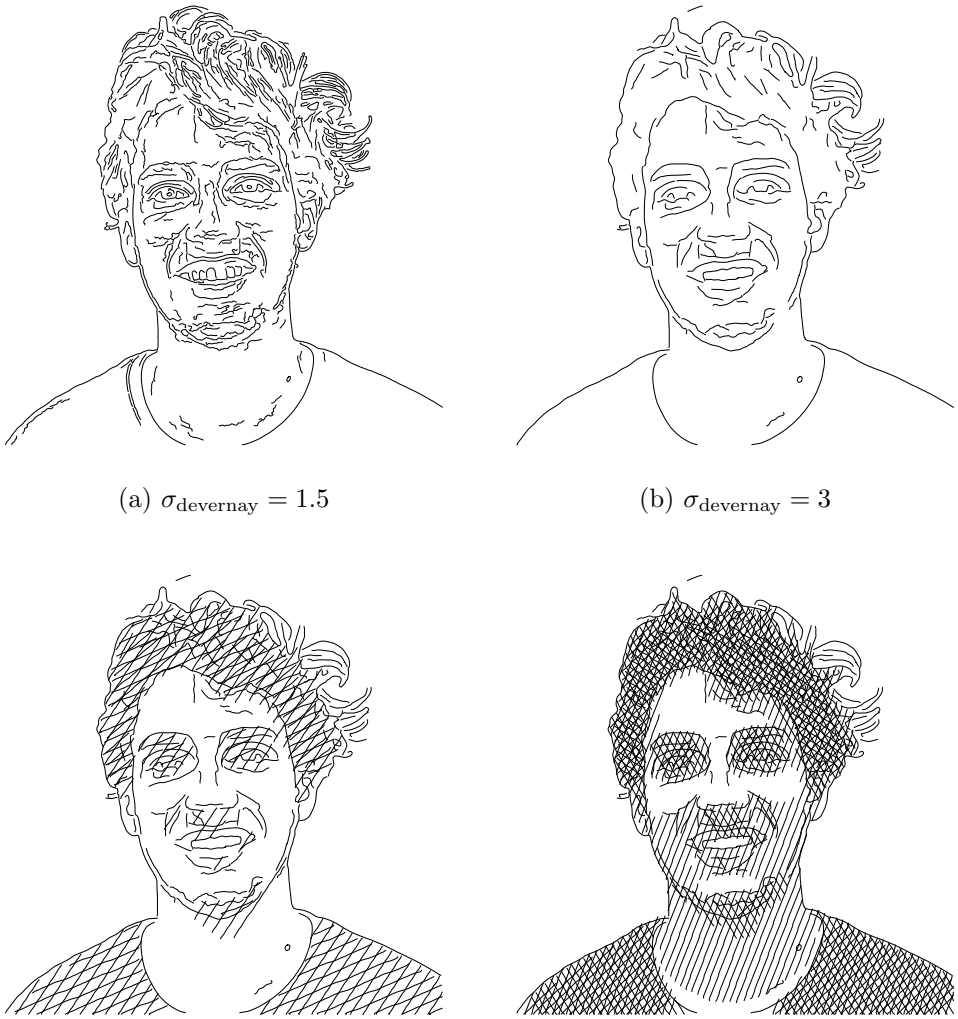
6.1.2. Sombreado

La detección de bordes extrae información sumamente importante para la representación de una imagen, pero carece de un factor importante que es los niveles de intensidad que puedan existir en diferentes regiones de la imagen. PARRA propone la idea de hacer trazos que rellenen una zona con mayor o menor densidad de líneas según qué tan oscura sea. Para identificar las diferentes zonas se filtra la imagen con un filtro de media y luego se divide en distintos niveles según la intensidad de píxel. Luego se pintan las zonas de los niveles más oscuros con líneas paralelas, con distinta densidad según el nivel. Surgen a partir de este método varios parámetros que define PARRA para ajustar el sombreado:

- **quantization_bits:** Número de bits para la cuantización de la imagen. Si se le atribuye un valor n a este parámetro entonces la imagen quedará dividida uniformemente en 2^n regiones de intensidad.
- **num_regions:** Número de regiones que se quieren sombrear. De las 2^n regiones en las que la imagen queda cuantizada, cuántas sombrear empezando

a contar desde la más oscura. Este parámetro debe ser menor igual que 2^n, siendo n el parámetro anterior.

- **filter_sigma:** Desviación estándar (sigma σ) del filtro Gaussiano que se le aplica a la imagen previamente.
- **frequency:** Ajusta la frecuencia del entramado. Si se quiere mayor densidad de líneas en las distintas capas o regiones se debe aumentar la frecuencia.



(a) $\sigma_{\text{devernay}} = 1.5$

(b) $\sigma_{\text{devernay}} = 3$

(c) freq = 0.06, q_bits = 3, num_regions = 4

(d) freq = 0.1, q_bits = 3, num_regions = 5

Figura 6.1: Distintos resultados simulados con PARRA para un misma imagen. Arriba solo detección de bordes con distintos valores de σ. Abajo detección de bordes y sombras con distintos valores de algunos parámetros.

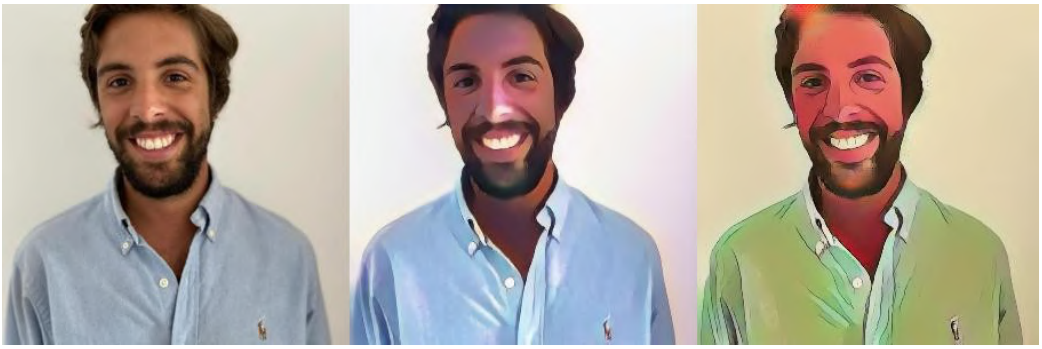
En la figura 6.1 se muestran distintos ejemplos de imágenes procesadas con sombreado. Los parámetros que PARRA usaba por defecto son: **frequency** = 0.06, **q_bits** = 3, **num_regions** = 5, **sigma** = 15

6.2. Nuevos algoritmos y estilos

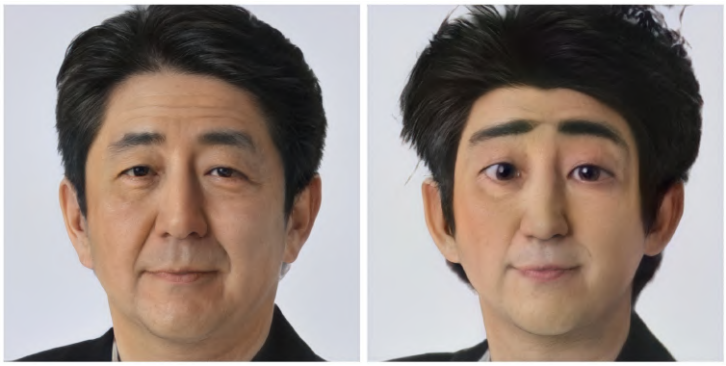
Dentro de las mejoras que se buscaron en este proyecto está agregar nuevos estilos de dibujo, es decir que para una misma imagen se tengan distintas representaciones artísticas. Como estilos se pensó en la idea de hacer transformaciones a la imagen previo a la extracción de curvas. Dada la detección de bordes y sombreado que hace PARRA, se podría preprocesar la imagen para lograr diferentes resultados finales.

6.2.1. Transformaciones en la imagen

Se investigaron varios trabajos que hacían transformaciones mediante el uso de modelos de Computer Vision entrenados con imágenes de *animé*, dibujos animados (estilo Disney o algún otro conocido). Se testearon varios de estos trabajos



(a) CartoonGan con distintos estilos japoneses [29]



(b) Red entrenada para pasar a estilo Disney. [30]

Figura 6.2: Algunos resultados de trabajos investigados para implementar los estilos. Estos no fueron utilizados porque la extracción de curvas posterior no logra reflejar el contenido original.

que tenían código disponible con los que se obtienen resultados interesantes como los que se muestran en la figura 6.2. Uno de los problemas que se encontró fue que la integración de todos estos trabajos con distintas implementaciones no era trivial por el hecho de que tenían requerimientos variados y no siempre compatibles. Además, la extracción de curvas que se usa para la imagen cruda no era igual de buena para todos los estilos. También se consideró usar un trabajo más clásico como el algoritmo *Style Transfer* [31], pero se notó de que la mayoría de la transferencia de estilo pasa por la información de color de la imagen de referencia, y se pierde al pasar a la etapa de extracción de curvas.

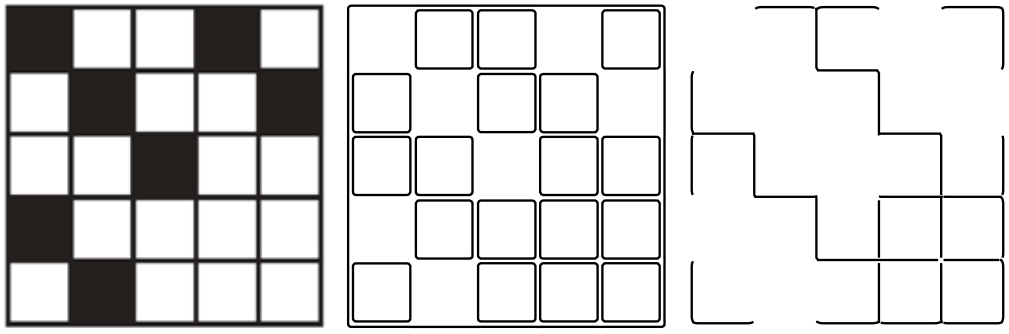
El problema general con estos estilos fue que al extraer las curvas luego de transformada la imagen, la representación resultante no refleja bien el contenido original. Se pierde mucha información entre la manipulación que hacen estos estilos y lo que luego se condensa en la detección de bordes y las sombras. Esto da lugar a una búsqueda de estilos orientada a transformaciones que simplifiquen la imagen y resuman la información más importante a la hora de dibujar. Se encontraron modelos de Computer Vision que generan imágenes que imitan ser dibujos humanos a lápiz o monocromáticos, tales como los que se pretende hacer con el brazo robótico. Se muestran ejemplos en la figura 6.3. Pero para extraer las curvas de estas imágenes tampoco fue conveniente usar la detección de bordes, por lo que se propuso un algoritmo más adecuado, que se describe a continuación.

(a) Portrait, U2-Net [32] (b) Informative drawing [33] (c) Informative drawing (v2) [33]

Figura 6.3: Distintas transformaciones de una misma imagen a partir de CNNs entrenadas para imitar distintos tipos de dibujos.

6.2.2. Nuevo algoritmo de extracción de curvas

Las imágenes de la figura 6.3 son la salida de distintas redes convolucionales que fueron entrenadas para imitar un dibujo. Es por eso que estas presentan líneas que se pueden interpretar como los trazos del dibujo. Esos mismo trazos son los que se querría extraer como curvas a dibujar. Ahora bien, al hacer la detección de bordes en este tipo de casos no se obtiene una curva sino dos, ya que una línea tiene un borde a cada lado. Estas líneas se conocen en la literatura del procesamiento



(a) Imagen a procesar (b) Detección de bordes (c) Detección de ridges

Figura 6.4: Ejemplo ilustrativo de la diferencia entre la detección de ridges y bordes. En (b) las líneas (ridges) son detectadas dobles mientras que en (c) solo se detectan una vez. Por otro lado, en (c) no se detectan los bordes de los cuadrados negros, porque son bordes simples

(a) (b) (c)

Figura 6.5: Detección de ridges para las imágenes procesadas de la figura 6.3 (en el mismo orden)

de imágenes como "ridges" (que en inglés es "cresta"). Se puede diferenciar entre ridge oscuro a una línea oscura en un fondo claro, o ridge claro si es al revés. Para estos casos los ridges son siempre oscuros ya que el fondo es siempre todo blanco. La idea de hacer detección de ridges con estas imágenes fue propuesta por Rafael Grompone, quien colaboró con el proyecto y además aportó su implementación en C de la detección de ridges, aún no publicada en IPOL. En la figura 6.4 se ilustra la diferencia entre hacer detección de bordes y detección de ridges con un ejemplo simple y en la figura 6.5 se muestran los resultados de hacer la detección de ridges para las imágenes transformadas.

Al hacer la detección de ridges luego de las transformaciones de la figura 6.3 se logra tener distintas representaciones de la imagen con curvas definidas que el brazo puede trazar. Dadas estas transformaciones se forma un conjunto de estilos que siguen una misma metodología. Primero se procesa la imagen para llevarla a otra en blanco y negro imitando algún estilo de dibujo. Esta transformación en

la imagen se logra a través de CNNs entrenadas con este objetivo. Luego de esta etapa se extraen los ridges como curvas a dibujar. Lo bueno de tener este método es que el modelo entrenado de la primera etapa es un bloque intercambiable. Se podría entrenar un nuevo modelo a gusto que transforme las imágenes con algún estilo deseado, o elegir cualquier otro ya entrenado como los que se encontraron y fueron utilizados. En la figura 6.6 se muestran los resultados a los que se llegan con esta metodología comparado con los resultados de PARRA.

(a) Imagen original

(b) Detección de bordes en la imagen original

(c) Retrato de la imagen

(d) Detección de ridges en el retrato

Figura 6.6: Comparación de estilos usando diferentes métodos de extracción de curvas para la imagen cruda y la imagen transformada. El estilo PARRA hace la detección de bordes en la imagen cruda mientras que con la nueva metodología propuesta se transforma la imagen y luego se hace detección de ridges.

Sucede también que en muchos casos, que si bien la imagen transformada representa un buen dibujo, no todo son ridges. Aparecen zonas donde las líneas o trazos que se podrían dibujar no están bien definidos, se mezclan entre ellos. Generalmente esto ocurre en las zonas donde hay pelo o vello facial y que se representa muchas veces con figuras sólidas o muy pocos rasgos. Este tipo de formas escapan a la detección de ridges y pueden generar huecos en la imagen a dibujar. Para solucionar esto se optó por agregar una etapa de sombreado si es necesario para cubrir y pintar las zonas sólidas que no son captadas.

(a) (b)

Figura 6.7: Caso de la figura 6.5b, donde la detección de ridges deja huecos. El contenido que falta representar se puede cubrir usando la técnica de sombreado para pintar las regiones más oscuras.

6.3. Estilos definitivos y comparación

Dada la investigación e implementación de nuevas posibilidades y formas de representar una misma imagen, se crean muchas combinaciones de parámetros y estilos que se pueden utilizar. La etapa de extracción de curvas quedaría conformada por las mismas dos etapas que tenía PARRA. Una primera donde se extraen las curvas principales del dibujo con las que ya se puede representar el contenido de la imagen. Esto sería la detección de bordes que hacía PARRA o los nuevos estilos implementados, que implican la transformación de la imagen y luego la detección de ridges. La segunda etapa es la de sombreado y es opcional. No todos los dibujos necesitan el agregado de sombras pues algunas representaciones quedan colmadas simplemente con las curvas de la etapa inicial. Sin embargo, en otros casos no es suficiente y el agregado de sombras puede terminar de manifestar el contenido. En todo caso, la decisión de agregar o no sombras es subjetiva, por eso la idea es dejar al usuario la posibilidad de agregarlas o no. Cabe destacar que las sombras en los

estilos nuevos son en la imagen ya procesada, es decir que toda la información se extrae luego de aplicar la transformación. Esto es diferente con el estilo PARRA que trabaja todo con la imagen cruda.

Se presentan a continuación una breve descripción de los cuatro estilos definitivos que el usuario tiene la posibilidad de utilizar:

- **PARRA**: Lleva el nombre de su proyecto creador y es el que fue presentado en la sección 6.1. Consta de la detección de bordes seguido del sombreado, ambas técnicas descriptas anteriormente. Es la base para los siguientes estilos ya que la idea central es la misma. Al usar algoritmos clásicos y de carácter general, con este estilo se pueden representar cualquier tipo de imágenes, y no solamente rostros. Las condiciones de iluminación pueden afectar mucho el resultado, sobre todo por las sombras.
- **Portrait**: A partir de una CNN entrenada para imitar retratos de personas [32] surge la idea de hacer extracción de ridges a estos retratos. Los resultados son novedosos y definitivamente son un estilo diferente. Lo bueno de hacer este procesamiento a la imagen es que extrae la información necesaria para dibujar una cara. Sin embargo, algunos rasgos no quedan bien representados solo con líneas y la detección de ridges no es suficiente. Sobre todo en la zona del pelo, si este es muy oscuro, el retrato queda con bloques sólidos que no pueden ser dibujados con simples líneas. En esos casos es necesario agregar el sombreado para poder representar la totalidad del retrato correctamente. La limitante de este estilo es que es específico para caras (como su nombre lo indica) porque son el tipo de imágenes con las que está entrenada la CNN.
- **Informative drawing v1**: Esta CNN [33] es más simple que la anterior, por ende es más pequeña y no tan específica. Este modelo está entrenado para hacer dibujos simples que describan las formas y líneas más importantes de la imagen de entrada. Sus resultados son los menos claros luego de la detección de ridges puesto que en el dibujo quedan muchas zonas donde no se detecta nada. Las sombras suelen ser necesarias para complementar y llenar los huecos (tal como en el ejemplo 6.7).
- **Informative drawing v2**: Esta es la misma CNN que la anterior, solo que está entrenada con otro estilo más peculiar. La característica de este modelo es que la salida es un dibujo con muchas líneas no ordenadas. Viendo la figura 6.3c puede notarse el estilo. Las zonas con más texturas tienen mayor cantidad de líneas. Puesto que en la imagen transformada se presentan ya muchas líneas, la detección de ridges suele ser suficiente para representar la imagen con este estilo.

Ejemplos variados de estos estilos se pueden ver en el capítulo 8.

6.4. Preprocesamiento de las curvas

Antes de pasar a la etapa de path planning explicada en el capítulo 5, las curvas deben preprocesarse. En primer lugar, las curvas extraídas con la detección

de bordes o ridges están en la escala de píxeles de la imagen de entrada. Para poder trabajar con el brazo, la curva debe estar representada en un sistema absoluto que se defina. En segundo lugar, el orden de las curvas extraídas es arbitrario. Se puede ordenar el conjunto de curvas para darle un sentido artístico en la ejecución del dibujo.

6.4.1. Escalado de las curvas

Las curvas que se extraen a partir de las imágenes son relativas dentro de la misma, pero deben ser trazadas en el espacio de dibujo, también denominado "canvas". En la sección 4.1.3 se establecen las restricciones del brazo y con eso se ve cuál es la zona alcanzable por este. Se puede definir una región rectangular dentro de esta zona, que defina el espacio del canvas, tal como se muestra en la figura 6.8.

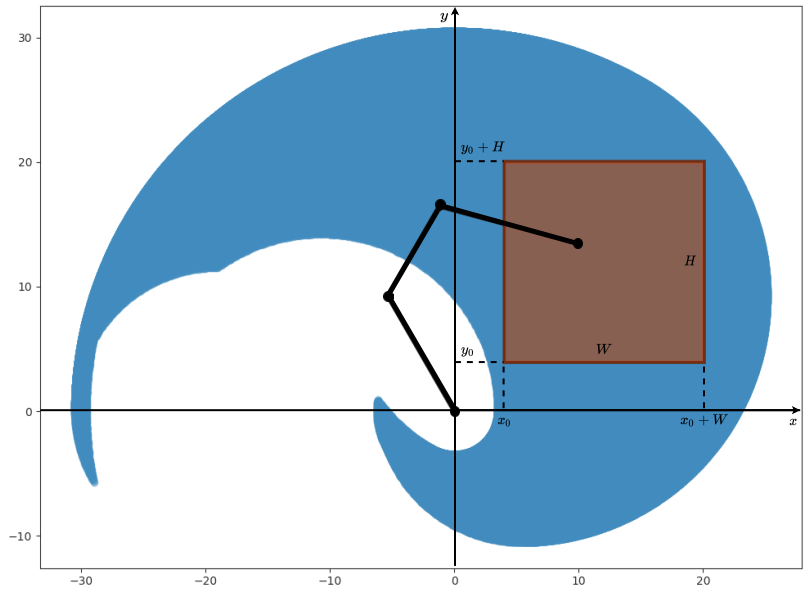


Figura 6.8: La región del canvas se define con un rectángulo cuya esquina inferior izquierda se ubica en (x_0, y_0) y tiene altura H y ancho W. En este proyecto, dada la zona que es alcanzable por el brazo, se eligió $x_0 = y_0 = 4cm$ y $H = W = 16cm$.

Los puntos de las curvas se referencian usando su ubicación medida en píxeles. Pero para que la etapa de path planning logre entender qué curvas reales se quieren trazar, se debe ubicarlas en el canvas, usando un sistema absoluto (en centímetros).

Se tiene un canvas ubicado en (x_0, y_0), de ancho W y alto H en centímetros, y un conjunto de curvas dentro de una imagen de $N \times M$ píxeles, tal como se muestra en la figura 6.8. Para convertir las curvas, primero se debe escalar ajustando las dimensiones de la imagen al canvas, de manera de aprovechar el espacio de dibujo al máximo. El factor de escala depende de la relación de aspecto de la imagen, y

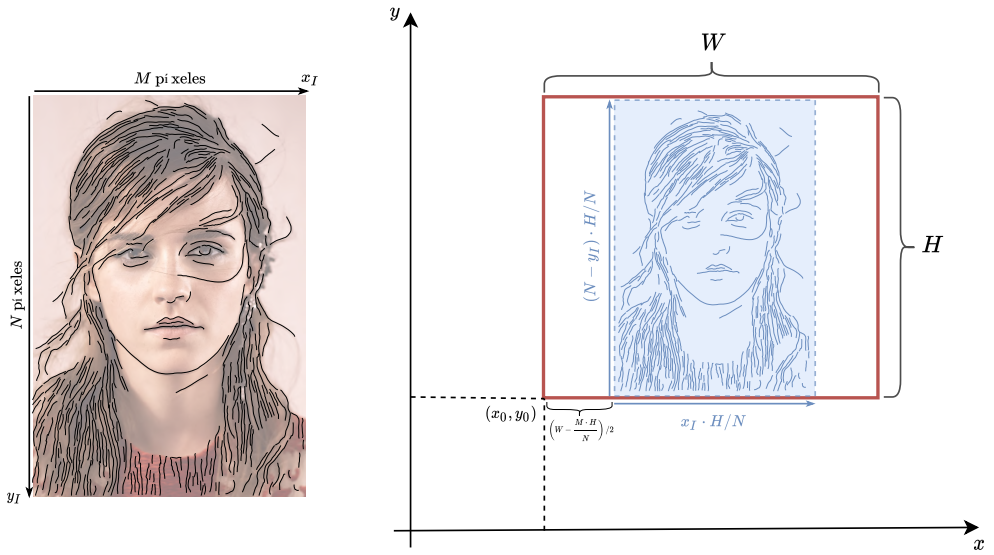


Figura 6.9: Conversión de las curvas de la imagen al canvas en el sistema absoluto. En este caso $H/N < W/M$, es decir que la imagen encaja a lo alto. Las curvas son escaladas usando el factor H/N. La coordenada vertical (y_I) se invierte porque en la imagen crece hacia abajo. Las curvas se trasladan a la posición (x_0, y_0) del canvas y la coordenada horizontal se centra dentro del mismo, ya que no se expandió para mantener la relación de aspecto.

según el caso se deberá expandir hasta encajar a lo ancho, o a lo alto. Una vez escaladas las curvas, estas se deben trasladar a la posición (x_0, y_0) del canvas en el espacio. Además la dimensión que no haya sido encajada deberá centrarse dentro del canvas para dejar la misma cantidad de espacio sobrante de cada lado. Otro detalle a tener en cuenta es que la coordenada vertical debe invertirse, ya que en la imagen crece hacia abajo y en el sistema absoluto hacia arriba. En la figura 6.9 se muestra un ejemplo completo del escalado de curvas.

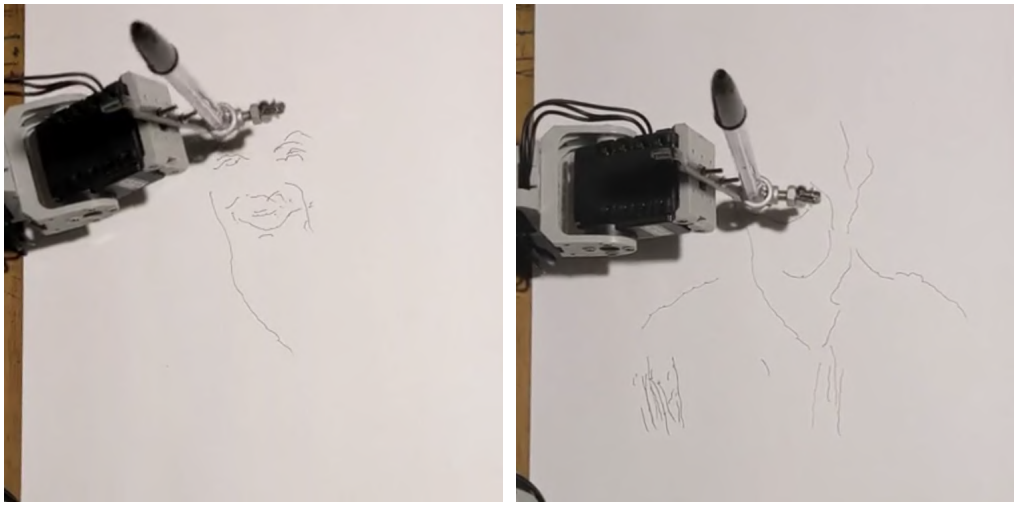
6.4.2. Orden de las curvas

Otra cuestión importante a la hora de la extraer las curvas es el orden en el que se las va a dibujar. Los algoritmos utilizados devuelven las curvas en un orden arbitrario que no necesariamente sería el más natural a la hora de dibujar. El problema de cómo definir un orden en las curvas adecuado a la ejecución del dibujo también fue abordado en el proyecto PARRA. Allí se consideraron dos factores a la hora de ordenar: el largo de la curva y la distancia de la misma a un punto de referencia, que podía ser la posición actual del lápiz o un centro fijo. Para ponderar estos dos factores definieron una función f que asigna un costo a cada curva discreta $\mathcal{C} = (\vec{c_n} = (x_n, y_n),\ n = 0, 1, ...N)$, dados su largo l y un punto de referencia $\vec{p} = (x_R, y_R)$:

$$f_{\vec{p}}(\mathcal{C}) = \text{mín}\left\{d(\vec{p}, \vec{c_0}), d(\vec{p}, \vec{c_N})\right\} - \alpha \cdot l,$$

donde d es la distancia euclídea entre dos puntos y α es el parámetro que regula el peso de cada uno de los factores.

Se calcula la distancia a los dos extremos y se toma el mínimo, esta distancia es la que hay que recorrer desde el punto de referencia para empezar a dibujar la curva. En caso de que la distancia mínima sea con el extremo final se da vuelta la curva y se asigna este extremo como inicial. Con valores grandes de α las curvas más largas tienen más importancia y serán elegidas primero más allá de su ubicación. Por el contrario, con valores pequeños de α se dibujan primero las curvas más cercanas al punto de referencia. Si este punto es fijo, se forma un centro de expansión alrededor del cual se dibujan todas la curvas. Si se toma el punto de referencia como la punta del lápiz, entonces se está minimizando la distancia que tiene que recorrer el lápiz para dibujar la siguiente curva cada vez.



(a) $\alpha = 0.1$ con centro fijo en la cara como punto de referencia

(b) $\alpha = 10$ usando la punta del lápiz como punto de referencia

Figura 6.10: Distintos órdenes para un mismo conjunto de curvas. Se muestra el grado de avance en el mismo momento para cada caso. En el primer caso el centro fijo actúa como centro de expansión, mientras que en el segundo se realizan primero todas las curvas más largas que le dan la forma al dibujo. Las imágenes fueron tomadas de los videos subidos al canal de YouTube. Se puede ver los ejemplos en *este link*[1].

[1]Lista de reproducción con videos de dibujos usando diferentes valores del parámetro α: `https://youtube.com/playlist?list=PLEQhiSLIfUNccVgfpGNM_zw5bOQGrE7Rl`

Capítulo 7

Implementación, pruebas y caracterización del sistema

En este capítulo se presentan en detalle diferentes pruebas que se hicieron a lo largo del proyecto. Debido a la variedad de áreas involucradas en este proyecto, las pruebas en general no poseen relación directa unas con otras. Por eso el capítulo contiene diversas secciones independientes entre sí. Las pruebas detalladas involucran: distintos problemas que surgieron del uso del nuevo modelo de motores MX, optimizaciones realizadas sobre la solución al problema de *inverse kinematics*, entre otras. Además, se presenta una caracterización de alto nivel del sistema punta a punta.

7.1. Rápido aumento de la temperatura en los MX-12W

A lo largo de la etapa de experimentación con los nuevos motores MX, se vio que había momentos en los que algunos de los motores dejaban de moverse. Esto ocurría después de que el brazo estuviese un largo tiempo dibujando, pero como en aquel entonces cada dibujo requería un tiempo considerable para dibujarse ($\sim$ 30 minutos) fue de vital importancia encontrar el motivo por el cual los motores se detenían.

Estudiando el manual electrónico del modelo MX, se vio que existía el registro **Shutdown** en su tabla de control. En este registro se almacena información sobre errores que causen que el motor se apague. Forzando la situación de nuevo, realizando un retrato que demorara mucho en dibujarse, se vio que el error que estaban experimentando los motores se debía a un sobrecalentamiento de los mismos. Esto es coherente con el hecho de que el error solo ocurriese luego de que los motores se usasen durante un largo tiempo.

Este sobrecalentamiento no ocurre si se usan los motores AX, por lo que se concluyó que es posible que las prestaciones adicionales de los MX generen un mayor consumo de corriente que los AX. De todas formas, por las ventajas que brindaban los MX, se decidió seguir utilizando el nuevo modelo para el proyecto.

Viendo de nuevo el manual electrónico de los motores, se puede ver que hay un

registro que controla la temperatura a la cuál el motor se apaga por sobrecalentamiento. Este registro se llama **Temperature Limit** y tiene un valor por defecto de 70°. Pero, por más de que es posible aumentar el valor almacenado en dicho registro, el fabricante no recomienda que se modifique a valores superiores a los 70°.

Por esto es que se decide utilizar un ventilador de pie externo para favorecer la disipación de calor de los motores. Los beneficios de esta incorporación se pueden ver en la figura 7.1. Luego, de ver la ventaja de incorporar un ventilador al sistema, se decide integrar dos ventiladores de computadora, de forma que no se requiera de un elemento externo para refrigerar al sistema.

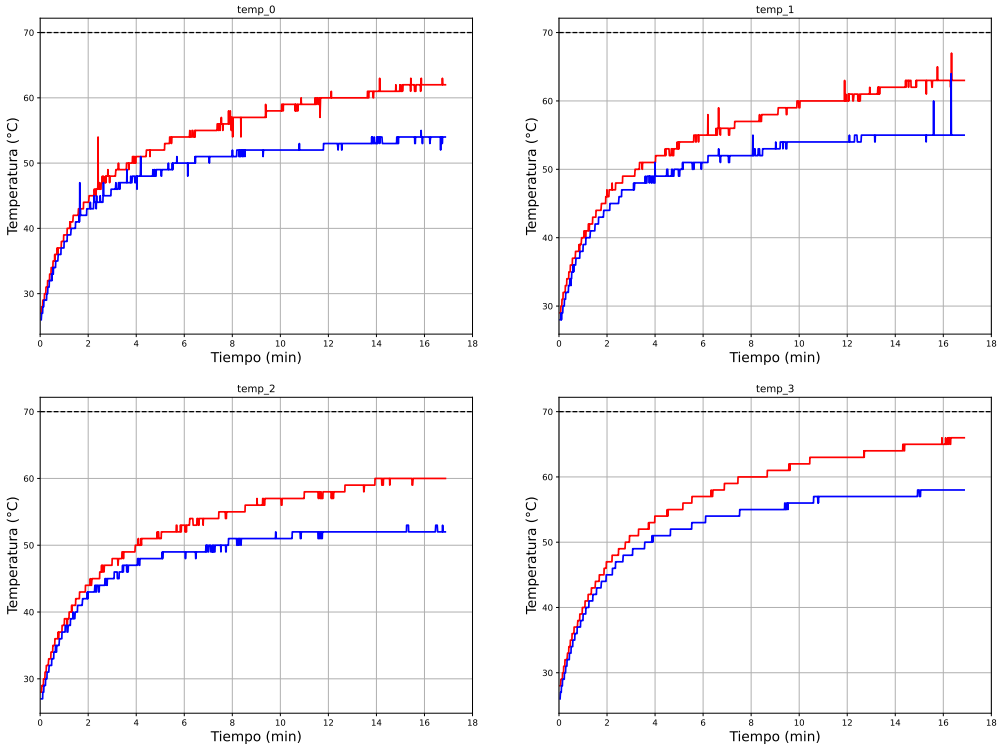


Figura 7.1: Gráficas de la temperatura de cada motor en función del tiempo en un dibujo. En rojo se puede ver la temperatura cuando no hay ventilador y en azul cuando si hay ventilador.

Evidentemente, al ser los ventiladores de computadora de menor potencia que uno de pie, se espera que los beneficios sean menores. Pero, como también se logró optimizar el tiempo en el que se realizan los dibujos, no se volvió a tener el problema de sobrecalentamiento por el resto del proyecto. Al menos, para dibujar entre 3 o 4 dibujos seguidos.

Por último, a modo de método preventivo, también se agregó al sistema un interruptor en serie con la fuente de los brazos, de forma de que sea sencillo cortar la alimentación del brazo cuando este no se esté usando. Por ejemplo, entre dibujo y dibujo.

7.2. Ajuste de las variables dinámicas de los motores

Como se mencionó en el capítulo 2, los motores utilizados tienen una gran cantidad de prestaciones. Estas se pueden ajustar dependiendo de la aplicación para la cual se los quiera utilizar. En particular, el ajuste de las variables dinámicas de los motores fue de gran importancia para este trabajo, ya que se quiso lograr un cierto nivel de precisión en el trazo. Si bien no se espera una precisión mecánica, si se quiere que el dibujo se identifique con la imagen a dibujar.

Los registros que regulan el movimiento de los motores son los registros que controlan las constantes del controlador PID (**P Gain**, **I Gain** y **D Gain** respectivamente), y la velocidad con la cual se mueve cada motor (**Moving Speed**). Además está el registro **Goal Acceleration**, que regula la aceleración tope con la cual los motores se pueden mover.

Ajustar todos estos parámetros es un trabajo para nada sencillo, ya que todos ellos se afectan mutuamente. Pero se debe definir algún criterio para comenzar por algún lado. En la primer imagen de la figura 7.2 se observa que la velocidad a la que se mueven los motores no queda bien definida para los valores por defecto que adoptan las constantes del PID, mientras que en la segunda imagen de la figura 7.2 se puede ver que para otros valores de constantes PID, la velocidad queda mucho más definida. Es por eso que se decide ajustar primero las constantes del PID y luego evaluar la velocidad de movimiento de los motores. El registro **Goal Acceleration** se deja para ajustar al final. Esto es debido a que los efectos de este no se consideran en el modelo realizado en el capítulo 4 y se vio que no generan un cambio significativo en la dinámica del sistema para las pruebas preeliminares.

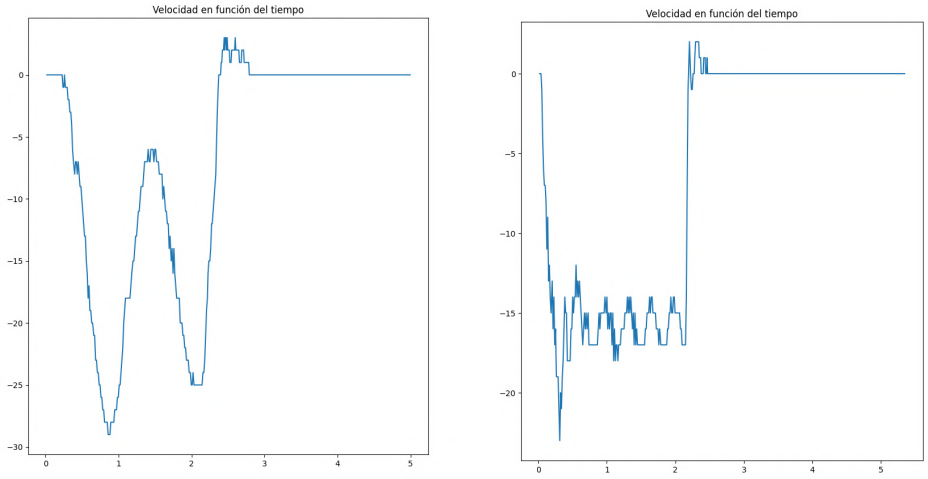
A continuación se presentan las pruebas para ajustar estos parámetros de los motores.

7.2.1. Controlador PID de los MX

El controlador PID es usado comúnmente como un controlador de retroalimentación que consiste de un término proporcional, uno integral y uno derivativo, de ahí el nombre. Una descripción detallada de la composición general de los controladores PID se puede ver en el apéndice D. Para el caso de los motores MX se tienen un controlador PID como el mostrado en la figura 7.3. Donde K_p es la ganancia proporcional, K_i es la ganancia integral y K_d es la ganancia derivativa. Según el manual electrónico de los MX, estas ganancias se realcionan con los valores de los registros **P Gain** $= P$, **I Gain** $= I$ y **D Gain** $= D$ según (7.1).

$$\begin{cases} K_p = P \cdot \frac{1}{8} \\ K_i = I \cdot \frac{1000}{2048} \\ K_d = D \cdot \frac{4}{1000}. \end{cases} \tag{7.1}$$

Un sistema lineal manejado por un controlador PID tiene tres tipos de respuesta: subamortiguada, sobreamortiguada y críticamente amortiguada como se puede ver en la figura 7.4. La respuesta subamortiguada es una respuesta que presenta un sobretiro. La respuesta sobreamortiguada es la respuesta mas lenta de todas



(a) Velocidad en función del tiempo para el codo del brazo, con valores en los registro de las constantes de PID: $P = 8$, $I = 0$, $D = 0$ y una velocidad de movimiento de valor de registro 16

(b) Velocidad en función del tiempo para el codo del brazo, con valores en los registro de las constantes de PID: $P = 255$, $I = 0$, $D = 0$ y una velocidad de movimiento de valor de registro 16

Figura 7.2: Gráficas de velocidad en función del tiempo para dos combinaciones distintas de constantes PID.

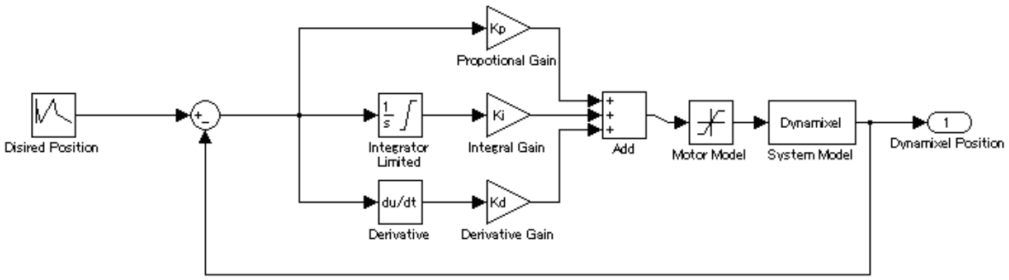


Figura 7.3: Diagrama de bloques del controlador PID de los motores MX

para llegar al valor estable. Por ultimo la respuesta críticamente amortiguada esta entre ambas respuestas anteriores.

Para este proyecto se requiere una respuesta entre críticamente amortiguada y sobreamortiguada, de forma que no haya un sobretiro y el dibujo realizado quede "manchado" por las idas y vueltas de los motores.

7.2.2. Ajuste PID de los motores

Como se vio en la sección 2.3, los motores sin carga se pueden mover de forma bastante aproximada a una rampa de posición. Pero cuando se acoplan los motores entre sí para formar el brazo, cada motor adquiere una cierta carga. Por lo que,

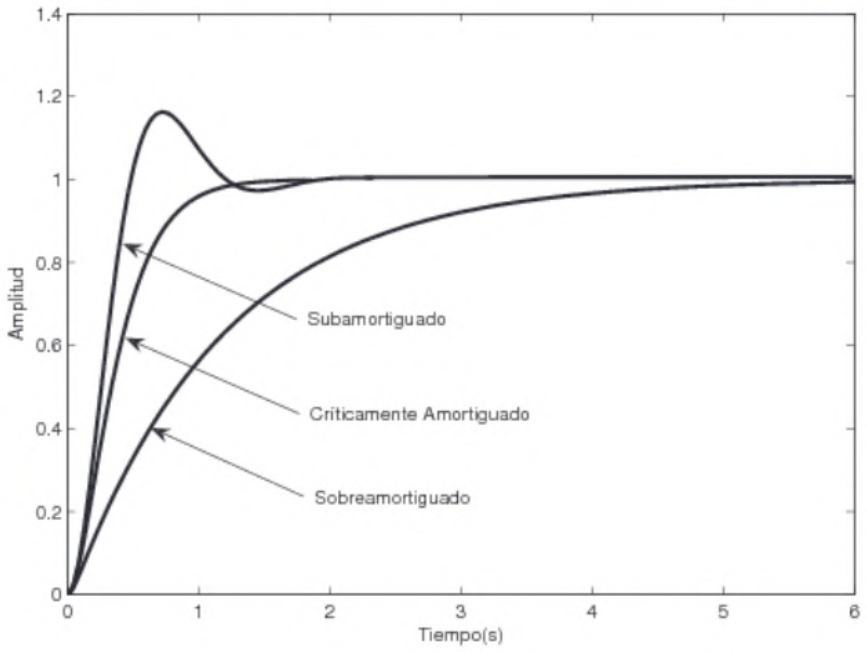


Figura 7.4: Distintos tipos de respuesta escalón.

para obtener un movimiento lo más aproximado a una rampa de nuevo, se deben de modificar las constantes PID de los motores.

El PID de los motores fue ajustado en dos etapas del proyecto. Un primer ajuste fue realizado poco después de la decisión del cambio de motores, cuando solo se tenía firmware experimental y no había un procesamiento capaz de generar comandos para dibujar curvas arbitrarias. En dicho ajuste, se llegaron a unas constantes del PID que generaron unos efectos no deseados cuando se comenzó a dibujar retratos de personas. Es por eso que se realizó una segunda etapa de ajuste, luego de haber desarrollado casi por completo el software de procesamiento. En dicho momento se podían generar los comandos para curvas arbitrarias con facilidad y además ya se realizaban dibujos completos de personas en un tiempo razonable.

Primer etapa de Ajuste

Esta primera etapa de ajuste tuvo, a su vez, dos estrategias de ajuste del PID. La primera consistió en analizar gráficas del la posición en función del tiempo para cada motor con distintas combinaciones de constantes PID. Se hizo que cada motor realizara un mismo movimiento, múltiples veces, con distintas combinaciones de constantes PID. Si bien esta estrategia funcionó bien para ajustar la gráfica del motor correspondiente al codo, los sucesivos motores se volvían inestables antes de llegar a la respuesta deseada.

Además, las pruebas fueron realizadas con movimientos largos (del orden de las

decenas de grados) para cada motor. De esta forma los movimientos eran precisos en régimen, pero al arranque y al frenar se generaban sobretiros no deseados. Esto no se logró corregir con los mismos movimientos largos. A esta altura del proyecto no se intentó ajustar a movimientos más cortos puesto a que se tenía pensado que las curvas a dibujar deberían ser más bien largas que cortas.

De esta estrategia de ajuste se llegó a la conclusión de que la carga a la que esta sometido cada uno de los motores varía de forma sustancial con la posición del brazo. Esto se debe a que, al enrollar o estirar el brazo, cambia el momento de inercia del mismo. Además se concluyó que, en general, funcionaban mejor las configuraciones de PID que compensaran más los motores más comprometidos. Es decir, que el motor correspondiente al hombro tendría una compensación mayor al del codo, el codo tendría mayor compensación que la muñeca, y así. O lo que es lo mismo, que lo mejor es que se verifique:

$$\begin{cases} K_{p_0} > K_{p_1} > K_{p_2} > K_{p_3} \\ K_{i_0} > K_{i_1} > K_{i_2} > K_{i_3} \\ K_{d_0} > K_{d_1} > K_{d_2} > K_{d_3}. \end{cases} \tag{7.2}$$

La segunda estrategia consistió en utilizar el método de Ziegler-Nichols, desarrollado en el apéndice D. Dicho método fue utilizado para el motor que más rápidamente llegaba a la inestabilidad con la estrategia anterior, es decir, el motor que corresponde a la muñeca. Pero, aplicando Ziegler-Nichols se llegó a valores de integrador muy altos y la inestabilidad del motor seguía siendo un problema. Por esto se descartó el utilizar esta segunda estrategia para seguir ajustando los parámetros.

Únicamente a partir de la primer estrategia, se obtuvieron ciertos valores para las constantes del PID que no eran completamente satisfactorias. Sin embargo, eran mejores que las configuraciones por defecto, así que se utilizaron a partir de ese momento del desarrollo del proyecto. Se decidió seguir adelante con estas constantes y luego intentar compensar los efectos de un PID no sintonizado en la etapa de path planning. Es de esta forma que surge, en parte, la técnica desarrollada en 5.3 del algoritmo RDP adaptivo.

Segunda etapa de Ajuste

En esta segunda etapa de ajuste ya se tenían generados los algoritmos basados en la herramienta del c-space, el firmware en su versión casi final, y la posibilidad de generar los comandos correspondientes a curvas arbitrarias. Es así que se decide realizar otra etapa de ajuste del PID a un alto nivel. Es decir, observando los dibujos correspondientes a ciertas curvas, y evaluando qué tan parecido era el dibujo realizado con las curvas simuladas que se debían dibujar.

Se decidió realizar primero pruebas para figuras sencillas (cuadrados) y evaluar los efectos del PID en los mismos. Luego de elegir un juego de constantes PID que se consideraran razonables para estos cuadrados, se pasaría a dibujar retratos, de forma de evaluar la eficacia de este método. Cabe destacar que se realizó la prueba de ajuste de parámetros primero sobre los cuadrados, porque eran más rápidos

de dibujar que el retrato de una persona. Todavía no se tenían optimizados los tiempos de ejecución. Además, se decidió utilizar el algoritmo RDP sin su versión adaptiva, de forma que una sobre-aproximación fuese debida a este.

Algunos de los resultados de esta etapa de ajuste se pueden ver en la figura 7.6, que son representaciones del brazo de la figura 7.5. En esta figura se observa que, en general, las figuras que son menos precisas en los trazos, son más precisas en las posiciones por las cuales tienen que pasar (por ejemplo las esquinas de los cuadrados). Mientras que los cuadrados que tienen trazos más suaves, pierden precisión en los puntos por los que tienen que pasar. Más aún, se ve que alguno de los cuadrados con bordes suaves no llegan a cerrar sus esquinas.

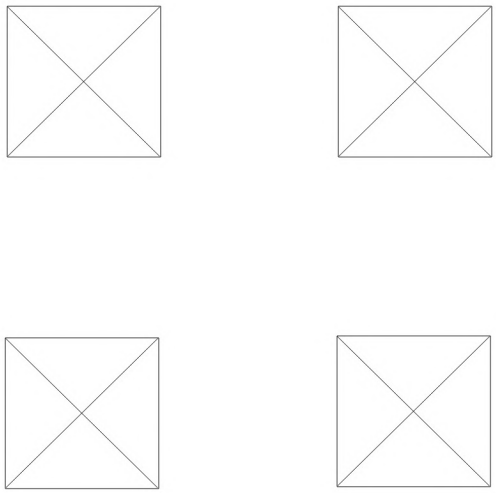


Figura 7.5: Cuadrados de 5cm de lado que se utilizaron para sintonizar los PID de cada motor.

Se prueba entonces el dibujo de un retrato con estos parámetros para evaluar las posibles mejoras, y se obtiene lo que se muestra en la figura 7.7. En esta se observa algo parecido a lo que se observó para los cuadrados. Se mejora la suavidad de las curvas, pero se pierde la precisión de los puntos por donde tiene que pasar la curva. Además, algunas de las curvas dibujadas están completamente fuera de lugar. Se decidió que este no era un efecto conveniente y se descartaron los valores obtenidos de esta forma.

Finalmente, en lo que duró la segunda etapa de pruebas del PID, se logró optimizar de forma radical el tiempo de ejecución de dibujo. Esto permitió realizar el dibujo de algunos retratos en aproximadamente cinco minutos. Con esto fue posible realizar sucesivas pruebas como las de los cuadrados, pero en los retratos que realmente se querían dibujar. Este ajuste también se hizo con el RDP adaptivo ya que se vio conveniente usar todos los recursos disponibles para un ajuste final. Se puede comparar el resultado obtenido previo a este ajuste en la figura 7.8 con los posteriores a este último ajuste en la figura 7.9. En estas se ve la disminución importante de las imprecisiones del brazo. Estos parámetros, fueron considerados como un ajuste satisfactorio del PID a hechos de este proyecto.

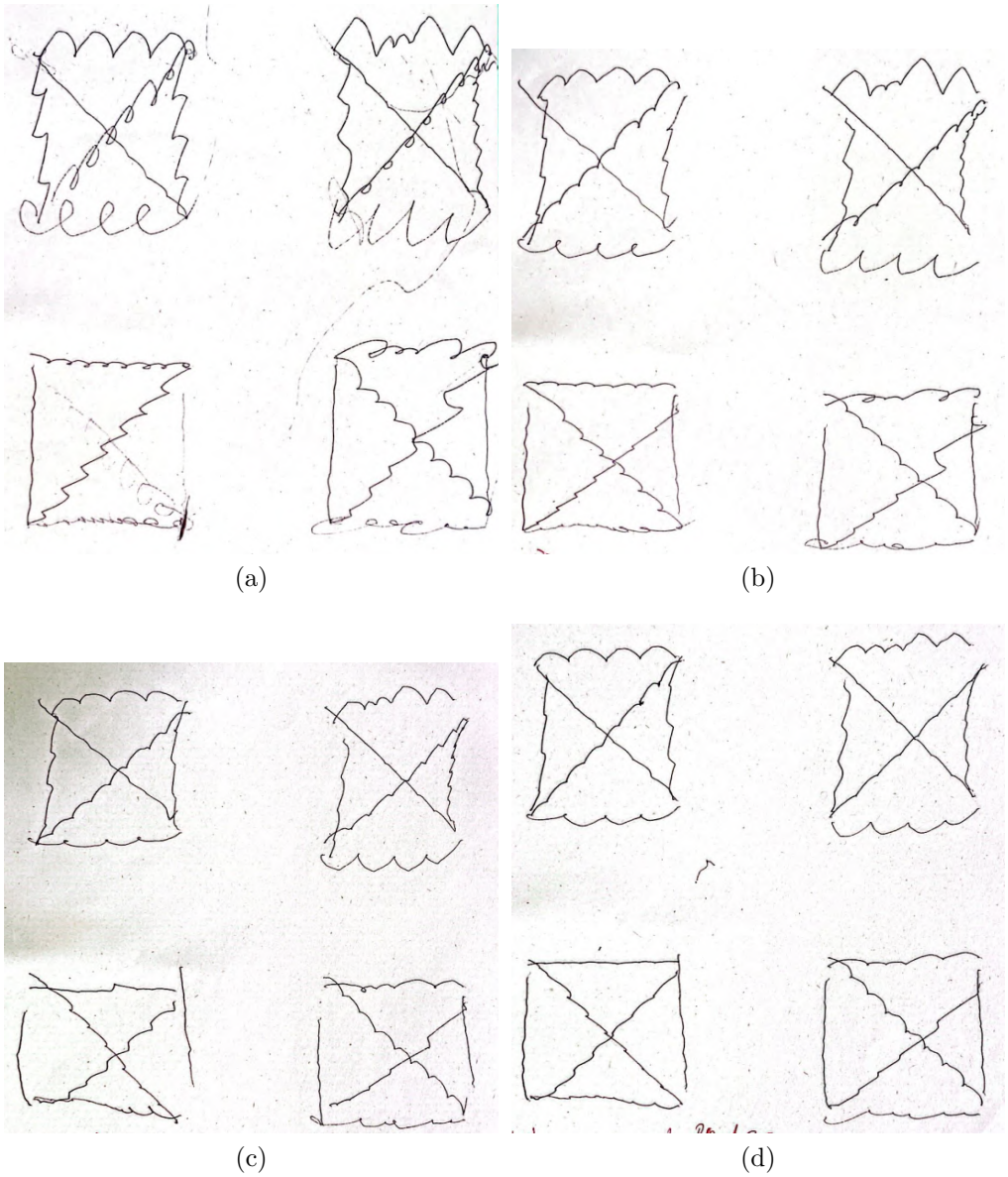
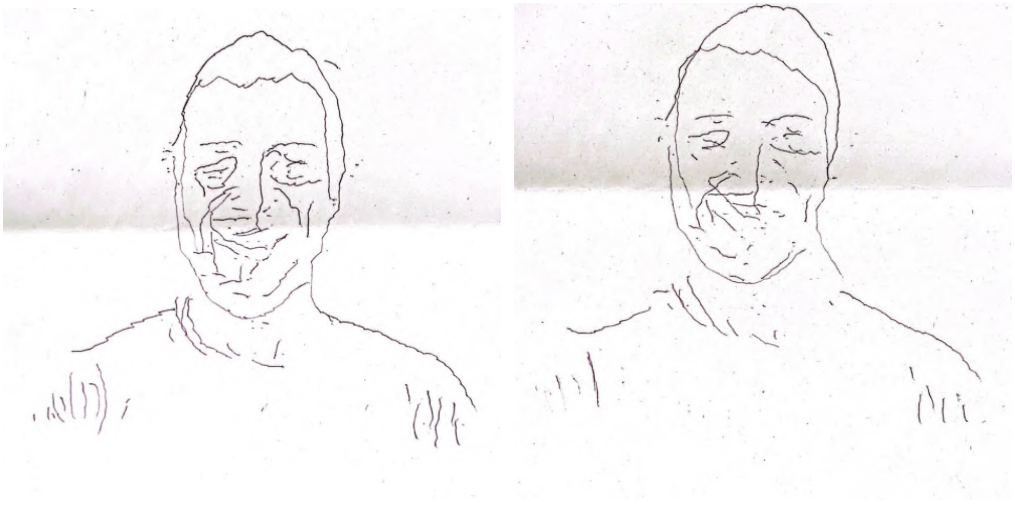


Figura 7.6: Comparación de cuadrados dibujados por el brazo variando los parámetros de cada PID.

7.2.3. Ajuste de la velocidad de los motores

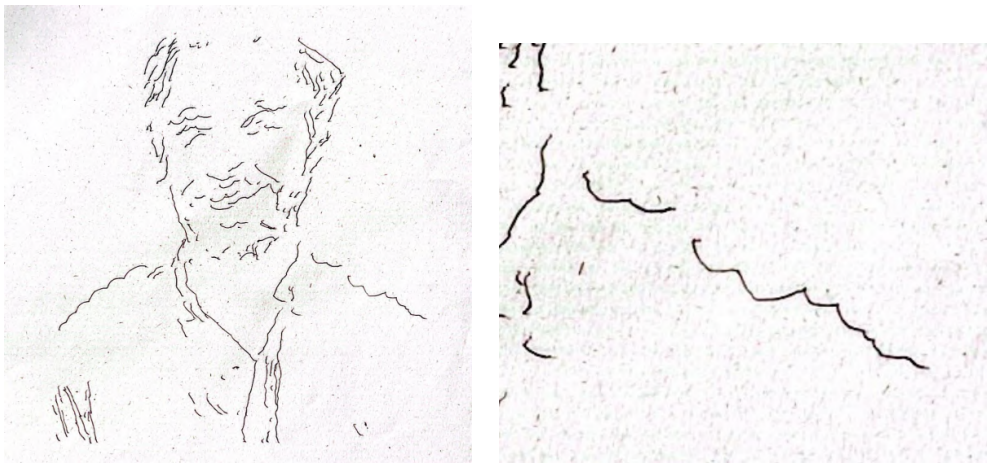
Hablar del ajuste de la velocidad de los motores puede parecer contradictorio, puesto a que las velocidades se deben definir en función del movimiento que se desee realizar. Pero como se vio en el capítulo 4, la velocidad absoluta de movimiento de los motores no es relevante, sino que importa la velocidad relativa entre motores para realizar un trazo dado. Por esta razón se puede configurar una velocidad máxima de movimiento y normalizar la de todos los motores para que siempre haya al menos un motor moviéndose a esta velocidad máxima (como se hizo en



(a) Imagen realizada con las constantes de PID obtenidas antes del segundo ajuste.

(b) Imagen realizada con las constantes de PID obtenidas luego del ajuste de las constantes con los cuadrados.

Figura 7.7: Comparación de un mismo dibujo realizado con las constates de PID obtenidas del primer ajuste, y las constantes de PID obtenidas luego de un segundo ajuste.



(a) Dibujo sin haber modificado el PID.

(b) Zoom a la zona donde se ve el peor efecto del PID.

Figura 7.8: Primera prueba que se realizó para ajustar el PID con retratos.

el capítulo 5). Es entonces que las pruebas realizadas para ajustar la velocidad buscan encontrar solamente una velocidad máxima de movimiento.

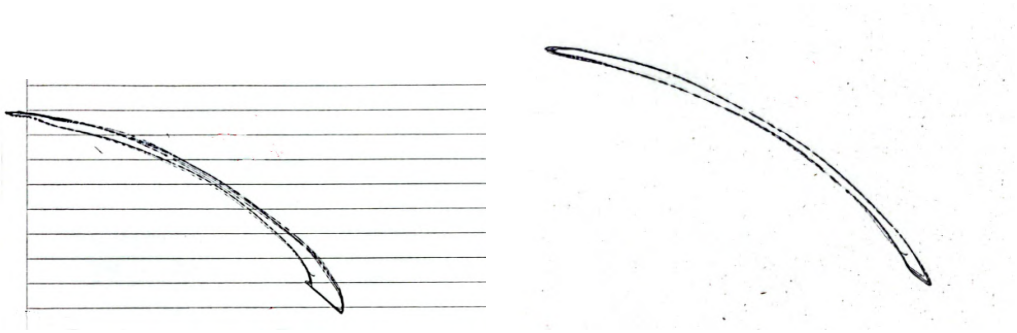
Esta velocidad máxima tuvo que decidirse luego de la primera etapa de ajuste de constantes PID. Es por esto que se tomó como criterio para elegir este valor, el de priorizar la "estabilidad" del brazo al moverse, eligiendo el mayor valor de velocidad posible. El criterio de estabilidad fue a percepción visual, pero formalmente tuvo que ver con minimizar el sobretiro que tienen los motores en su movimiento.

(a) Dibujo con los PID final

(b) Zoom a la zona problemática con los PID finales

Figura 7.9: Dibujo final que se realizo con los grupos de PID resultantes de las pruebas.

Imágenes de los trazos realizados para las pruebas de la velocidad se pueden ver en la figura 7.10.



(a) Trazo realizado por el brazo con una velocidad alta y poca estabilidad (mucho sobretiro).

(b) Trazo realizado por el brazo con una velocidad baja y mayor estabilidad (poco sobretiro).

Figura 7.10: Fotos de los trazos realizados por el brazo cuando se le indica realizar un arco de circunferencia perfecto. Se observa que en ninguna de las imágenes el camino de ida es el mismo que el de vuelta.

Sobre este ajuste se debe mencionar que se hizo en una etapa del proyecto en la cual no se tenían ciertas decisiones tomadas. Por ejemplo, cómo se representarían las curvas, si se priorizarían movimientos cortos o largos, que tanta resolución se necesitaría en los valores que pudiese tomar la velocidad, entre otras. Entonces, naturalmente ocurre que el valor obtenido no es el óptimo para las condiciones del resto del sistema. En particular, se observa en la sección 5.5 que el hecho de tener pocos niveles en los cuales cuantizar la velocidad resulta en efectos no deseados

sobre el dibujo. Esto, en principio, podría mejorarse realizando un nuevo ajuste del valor de la velocidad máxima, para así tener más niveles en los cuales cuantizar. Pero este segundo ajuste sobre la velocidad máxima no se realizó para priorizar una completitud del resto del proyecto.

7.2.4. Ajuste del goal acceleration

Con el valor del registro **Goal Acceleration** se hizo algo parecido a el ajuste de la velocidad máxima. Como se ajustó en una primera etapa del proyecto, se priorizó la estabilidad del sistema.

Pero como la aceleración estaba más vinculada con el PID (en el sentido en que no se tenía que modificar para cada movimiento) que la velocidad, también se realizó un ajuste del valor de **Goal Acceleration** posterior a la segunda etapa de ajuste del PID. El criterio que se tomó fue el mismo que al principio, priorizar la estabilidad del movimiento y la suavidad del trazo.

El valor final de este registro **Goal Acceleration**, fue tal que el tiempo que demoraría el brazo en llegar a la velocidad de régimen es de $2s$. Evidentemente esto no pasa, y el brazo frena antes, cuando está lo suficientemente cerca de las posiciones finales de los ángulos. Este hecho claramente rompe el modelo realizado en el capítulo 4 de velocidades escalonadas[1], pero a hechos prácticos se vio que se realizaban dibujos más suaves y fluidos con esta aceleración, por eso se dejó un valor tan pequeño.

7.3. Solver IK y estudio de tiempos de ejecución

La utilización de la formalización en el c-space implica correr IK cientos o miles de veces. Deben hallarse las configuraciones de ángulos de todos los puntos (x, y) de todas las curvas. El tiempo de cómputo que lleva hallar una solución de inverse kinematics pasa a ser clave. Se habían realizado pruebas simples con librerías ya implementadas en Python pero nunca se había tenido en cuenta su eficiencia. Se estudiaron entonces las distintas opciones que había y la posibilidad de hacer una implementación propia de un Solver IK.

7.3.1. Librerías en Python de Inverse Kinematics

Se estudiaron las librerías más enfocadas al problema, siendo estas también las más simples y fáciles de utilizar. Existen muchas librerías e implementaciones además de las aquí presentadas. Algunas son más generales de robótica y cuentan con soluciones para otros problemas también.

[1]De todas formas, si se sigue verificando instante a instante la relación entre las velocidades, la curva recorrida en el c-space sigue siendo una recta.

tinyik

En primera instancia se había usado `tinyik` [34] para hacer algunas pruebas preliminares y entender mejor de qué se trataba resolver un problema de *inverse kinematics*. Como su nombre indica este es un paquete pequeño y específico. Se puede crear fácilmente una cadena de juntas genérica en tres dimensiones y con ella hacer la cinemática directa o inversa según se quiera la posición del *tooltip* a partir de la configuración de ángulos o viceversa. Por ser una implementación simple, no se puede asignar restricciones al movimiento angular de las juntas. Eso implica que la solución que se halle puede estar fuera de la zona establecida en la sección 4.1.3 y no sea válida para este problema. Además el tiempo de cómputo del solver IK es extremadamente lento, tomando hasta un segundo para resolver la configuración de ángulos para un punto. Esto llevó a buscar otras alternativas, ya que los tiempos para computar todo un dibujo con cientos de curvas y miles de puntos eran prohibitivos.

ikpy

Se buscó por librerías un poco más elaboradas y más eficientes y se encontró `ikpy` [35]. Además de resolver *inverse kinematics*, cuenta con un herramienta para visualizar la cadena de juntas, y es compatible con formatos para guardar y cargar objetos de robótica (por ejemplo URDF, *Unified Robot Description Format*). En este caso, sí se podía establecer restricciones en los ángulos de cada junta, lo que ya era una ventaja. Además el tiempo de cómputo para cada solución es menor, logrando resolver IK más de 25 veces por segundo. Este es un mejor tiempo, pero de igual forma procesar un dibujo entero podría llevar algunos cuantos minutos.

7.3.2. Implementación propia

Las dos librerías investigadas servían para este caso por su implementación genérica, pero se necesitaba optimizar el tiempo de cómputo lo más posible. Analizando la metodología usada por `ikpy` se implementó un solver IK específico para este problema: un brazo planar de tres juntas. Esta implementación es más limitada para simplificar el trabajo y no se tienen en cuenta generalidades del problema de *inverse kinematics* que no estén involucradas en este caso de uso.

En primer lugar, al ser un brazo planar, se puede eliminar una dimensión que en los otros casos era siempre constante. En segundo lugar, también se limita a un brazo que tuviera exactamente tres juntas. La cinemática directa se puede hallar fácil y eficientemente usando la ecuación 4.1. Para resolver la cinemática inversa se utilizan métodos iterativos. Por su parte, `ikpy` hace uso de `scipy` [36], que cuenta con un módulo de optimización donde se encuentran métodos para minimizar funciones. La función a minimizar es la distancia entre el tooltip y la posición objetivo, es decir, hay que encontrar la terna de ángulos en el c-space (que sería el dominio de la función) que hace que la posición del tooltip sea lo más cercana posible a la deseada. En el caso de `ikpy` se usa la distancia euclídea en $\mathbb{R}^2$ (ya que son dos puntos del plano). Para esta implementación se decide usar la

distancia al cuadrado para no tener que hacer la raíz cuadrada y reducir el tiempo de cómputo. Además, el método `scipy.optimize.miminmize` (ver documentación) utilizado cuenta con la posibilidad de especificar límites en el dominio de la función a minimizar (véase el parámetro *bounds*). Con esto se asegura que la solución es una configuración válida del brazo. Otra ventaja es que en el proceso iterativo de la minimización se debe dar una condición inicial para empezar a iterar. Para el caso de una curva se quiere continuidad entre los puntos de la misma, entonces para cada punto se usa como condición inicial la solución anterior. Esto no solo asegura la continuidad en el espacio de configuración, como se estableció en el capítulo 5, sino que también acelera el proceso de resolución cuando los puntos están cerca.

Esto optimizó de manera drástica el cómputo de la cinemática inversa. Se lograron hacer en promedio unas 350 resoluciones IK por segundo. Esto se pudo optimizar aún más calculando explícitamente el jacobiano de la función distancia que se minimiza. Si este no se provee, `scipy` hace una estimación del mismo pero esto conlleva más iteraciones y es más lento. Usando el jacobiano analítico se logró hacer alrededor de 1000 resoluciones por segundo, mejorando casi tres veces el resultado anterior que ya era bueno. Con estos tiempos de cómputo, procesar un dibujo lleva algunos segundos, dependiendo de la cantidad de curvas del mismo.

7.3.3. Comparación de los tiempos de cómputo

A modo de resumen, se reportan las velocidades de ejecución de los distintos métodos. Para hacer la estimación de la velocidad se midió el tiempo que llevaba ejecutar la resolución IK para 1000 puntos independientes (que no pertenecen a la misma curva, cada resolución es independiente) ubicados aleatoriamente dentro de lo que sería la hoja de dibujo.

La evaluación se realizó en distintos dispositivos. En primer lugar se tiene una notebook HP que cuenta con un procesador Intel I7 (7th generation), 16 GB de RAM y con Ubuntu 18.04 como sistema operativo. Por otro lado, se probó también en una Raspberry PI 4 con 8 GB de RAM. En la tabla 7.1 se muestran los resultados. Se aprecia que la implementación propia es mucho más eficiente en ambos dispositivos, si bien la Raspberry es considerablemente más lenta.

Tabla 7.1: Velocidad de resolución de inverse kinematics para distintas implementaciones en dos dispositivos utilizados. Nótese que 'pbz' hace referencia a la implementación hecha por este equipo, PicassoBotZ.

	PC	**Raspberry**
tinyik	10 it/s	2.4 it/s
ikpy	31 it/s	5.8 it/s
pbz s/jacobiano	350 it/s	73 it/s
pbz c/jacobiano	1000 it/s	200 it/s

7.4. Caracterización del sistema

Luego de tener el sistema completo **PicassoBotZ** funcionando adecuadamente, es necesario realizar una caracterización del mismo. De esta forma se puede saber cuáles son las salidas esperables antes ciertas entradas. A continuación se presentarán todos los parámetros del sistema a modo de resumen. Luego, se presentarán diversas imágenes de figuras geométricas sencillas que se utilizan como entrada al sistema, junto con sus correspondientes salidas. Por último, se presentan imágenes de retratos reales que se ingresan al sistema junto con todas sus salidas intermedias correspondientes.

7.4.1. Parámetros del sistema

El sistema **PicassoBotZ** tiene muchos valores paramétricos definidos en todas sus etapas de procesamiento. Se presenta a continuación la tabla 7.2 con todos estos parámetros, junto con una breve descripción de los mismos. Estos parámetros se encuentran ordenados por la etapa de Software en la cuál se utilizan. Además, se muestra en la última columna los valores utilizados para las pruebas finales de este proyecto.

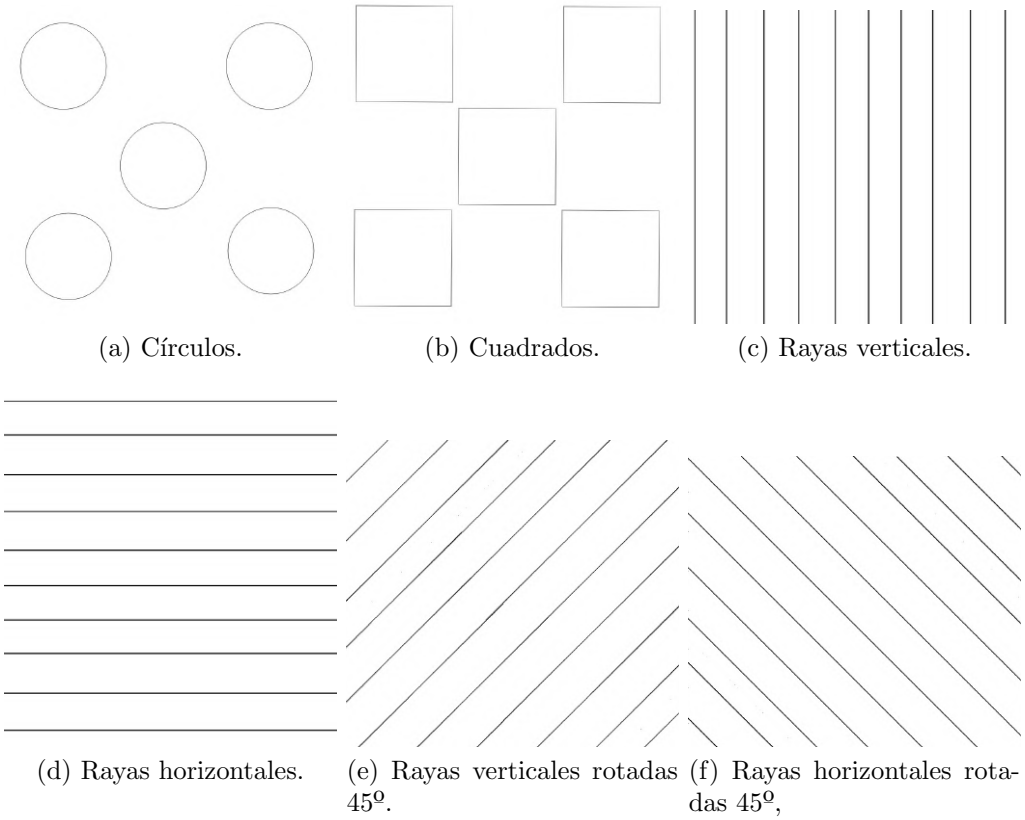
Tabla 7.2: Lista ordenada por etapa y tarea de los parámetros ajustables del sistema. La última columna presenta el valor que se usa en el sistema por defecto.

Etapa	Tarea	Parámetro	Detalle	Valor
Image processing	Detección de bordes	σ	Desvío estándar del filtro Gaussiano	3
		low_th	Umbral inferior para Canny	4
		high_th	Umbral superior para Canny	5
	Sombreado	num_regions	Número de regiones a sombrear	5
		q_bits	Cantidad de bits con la que cuantiza la imagen	3
		frequency	Frecuencia de las líneas de sombreado	0.06
		filter_sigma	Sigma del filtro Gaussiano previo	5
	Ubicación del canvas	x_0	Coordenada x de la esquina inferior izquierda en cm	4
		y_0	Coordenada y de la esquina inferior izquierda en cm	4
		W	Ancho del canvas en cm	16
		H	Altura del canvas en cm	16
	Ordenamiento de curvas	α	Pondera el largo de la curva y su cercanía al lápiz	0.2
		pos_0	Posición inicial del lápiz para ordenar en cm	(12, 12)
		centro_fijo	Si el orden es con un punto de referencia fijo o no	True (Sí)
Path planning	RDP adaptivo	ε_0	Valor base del epsilon de RDP	0.05
		max_dist	Máxima distancia admitida entre puntos del plano en cm	0.5
Control por firmware	Control de los motores	max_vel	Velocidad máxima en unidad de los motores	16
		goal acceleration	Valor del registro Goal accelartion de los motores MX	2
		P	Constante proporcional para cada motor (en orden)	[112, 96, 32, 20]
		I	Constante integrador para cada motor (en orden)	[14, 12, 4, 0]
		D	Constante derivativo para cada motor (en orden)	0

Figuras y movimientos sencillos

Para el valor de todos los parámetros expresados en la tabla 7.2, se ingresaron al sistema imágenes de formas geométricas sencilla, de forma de poder observar la precisión del sistema a la hora de dibujar. Las entradas utilizadas se pueden ver en la figura 7.11, y el dibujo realizado para cada una de estas entradas se puede ver en la figura 7.12. Se recuerda que la articulación fija del brazo se encuentra a la izquierda y abajo de estos dibujos realizados.

Se observa en la figura 7.12, en el dibujo que se corresponde a los círculos, que estos no quedaron redondos. Además, se observa que los círculos de las esquinas



(a) Círculos. (b) Cuadrados. (c) Rayas verticales.

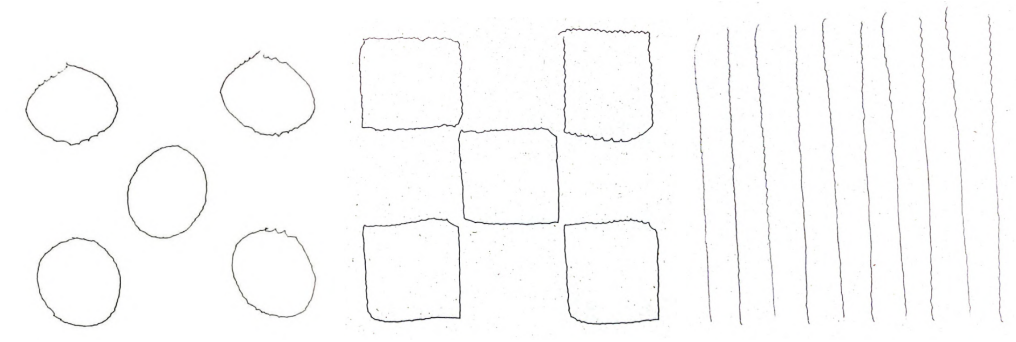
(d) Rayas horizontales. (e) Rayas verticales rotadas 45º. (f) Rayas horizontales rotadas 45º,

Figura 7.11: Figuras simples que se utilizaron para caracterizar el sistema final.

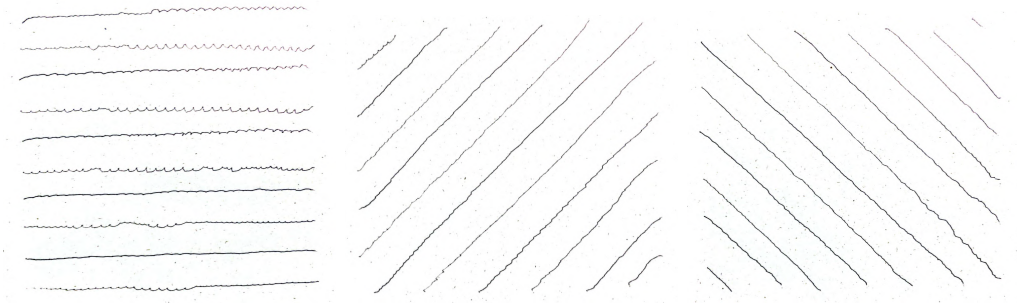
(a excepción del de abajo a la izquierda) tienen zonas en las que se generó un ripple. Para la imagen de los cuadrados se observa que se tiene el mismo efecto de ripple en el cuadrado más alejado del brazo (arriba a la derecha). Los demás cuadrados también tienen un poco de este efecto pero en menor medido. En ellos es más evidente la imprecisión en alguna de sus esquinas que se ven curvadas o cortadas. También se nota que algunos de los lados de los cuadrados no llegan a estar rectos, curvándose significativamente en su trazo. Por último, están los dibujos de las rectas con las distintas orientaciones. Salvo el caso de las líneas horizontales, se ve que los resultados son bastante positivos. El espaciado entre las curvas falla para las líneas pequeñas de las figuras con orientación a 45°, tanto de la horizontal como de la vertical. Pero, en las líneas horizontales si se observa un evidente problema. El ripple de las mismas es significativo, siendo la zona de mayor promedio de ripple por curva la de la izquierda arriba.

De todos estos dibujos se concluye que las zonas de las esquinas (a excepción de la de abajo a la izquierda) de la zona de dibujo son las menos precisas para seguir curvas. En estas se genera un efecto de ripple que no se genera en el resto de las regiones.

Se verá que para los retratos realizados estas zonas no son muy utilizadas. Esto se debe a que en general la cara y la parte del torso a dibujar de las personas entra



(a) Círculos dibujados (b) Cuadrados dibujados (c) rayas verticales dibujadas



(d) rayas horizontales dibujadas (e) rayas verticales rotadas 45º dibujadas (f) rayas horizontales rotadas 45º dibujadas

Figura 7.12: Figuras simples que se utilizaron para caracterizar el sistema final dibujadas por el brazo.

sin mayor problema en la zona central y baja de la región en donde se dibuja.

Figuras y movimientos complejos

Como el objetivo del brazo es dibujar retratos de personas, luego de caracterizado el sistema y ver cómo se comporta el brazo dibujando patrones geométricos simples, se comenzó a realizar retratos. Además de dibujar a los integrantes del grupo, también se decidió dibujar rostros de celebridades, de forma de tener una variedad de pruebas, tanto en la etapa de *image processing*, como en la ejecución de dibujos del brazo. En el capítulo 8 se muestran ejemplos de dibujos de rostros realizados por el brazo con sus respectivos pre-procesamientos.

Como conclusión general se puede decir que los dibujos del brazo son semejantes a las curvas que se extraen. No se tiene un gran precisión en los trazos y esto se puede ver muchas veces en rasgos faciales como los ojos o la boca, donde el detalle es importante. Se asume que esto es en gran parte porque en la práctica no se cumple el modelado ideal que se hace de los motores y su movimiento a velocidad constante. También se aprecia que las curvas extraídas para sombrear son líneas paralelas y equiespaciadas. Sin embargo, el brazo dibuja algunas más cercas que otras.

En muchos casos, se ve que la posibilidad de tener o no sombras es una ventaja. Si bien los gustos son subjetivos a cada persona, se concluyó que para retratos con el pelo suelto las sombras quitan movilidad en el dibujo y generan que este se vea mas rígido. Sin embargo, si no se usan sombras se aprecia que el pelo luce mas natural y menos rígido. También se ve en los retratos de Juan Pablo, que las sombras favorecen a la barba, ya que estas definen de mejor forma la zona de la barba y del pelo en estos retratos. Por otro lado, en los retratos de Santiago se ve que hay un compromiso entre qué se prefiere, si la naturalidad del pelo o la delimitación de la barba. Por último, en los retratos de Diego se ve que según el estilo que se elija se detallan más algunos rasgos de la cara u otros.

7.5. Pruebas del sistema en la Raspberry PI

En el capítulo 2 se habló de que hay diferentes unidades de procesamiento posibles con las que el sistema puede operar. La mayoría de las pruebas y el desarrollo fueron realizados usando PCs. Cuando surge la idea de dejar el sistema funcionando con una Raspberry Pi, surge también la necesidad de probar que todo funcione igual de bien en este dispositivo. Si bien hoy en día cuentan con muy buenas prestaciones, no es lo mismo que trabajar con una PC.

La incorporación de nuevos estilos que hagan uso de *Machine Learning* y *Computer Vision* implica también nuevos requerimientos. Las CNNs usadas pesan algunos MB y pueden implicar el uso de mucha memoria RAM. Al hacer la inferencia se llegan a consumir algunos GB de RAM por la cantidad de capas que se tienen que procesar. Es entonces que se adquirió una Raspberry Pi 4 con 8GB de RAM para tener un sistema con las mejores prestaciones posibles.

Se pudo conformar el sistema completo con la cámara web y el Arduino conectados a la Raspberry en vez de la PC, tal como se explica en el capítulo 2. Se descargó e instaló el repositorio en la Raspberry para que pudiera suplir a la PC como bloque de procesamiento. Para hacer la mayoría de las pruebas ni siquiera se necesitó conectar el monitor o teclado ya que se comandó todo desde otra PC conectada por SSH a la Raspberry. La previsualización de resultados se pudo hacer usando *X11 forwarding* [37], que despliega las ventanas con las figuras en la PC que está conectada como cliente. Una vez procesada la imagen y los comandos generados, se logró ejecutar el dibujo enviando los comandos al Arduino con su programa previamente cargado. Los resultados finales son iguales que con la PC normal. Se describen algunas diferencias y/o detalles en el proceso:

- **Velocidad de procesamiento:** La velocidad de la Raspberry es considerablemente menor. En la tabla 7.1 se aprecia que la velocidad de la Raspberry ejecutando la prueba de los solvers de *inverse kinematics* es alrededor de 5 veces más lenta que la PC normal para todos los casos. Mientras que el procesamiento de una imagen tomaba algunos segundos en PC, en la Raspberry puede llegar a tomar un minuto o más.
- **Procesamiento de los estilos:** El estilo PARRA es ejecutado sin problemas ya que no es tan demandante como con el resto de los estilos. Por su

parte, los nuevos estilos introducidos vistos en el capítulo 6 usan CNNs, cuya inferencia puede llegar a hacer uso de algunos GB de RAM. En la Raspberry no se puede hacer la inferencia para imágenes muy grandes, ya que si no el sistema da error al tratar de alocar demasiada memoria. El problema real es que no se pueden usar los 8GB de la Raspberry y este error aparece cuando se superan los 3GB, pero no se encontró una forma de poder usar toda la memoria disponible.

- **Instalación de paquetes:** Uno de los paquetes de Python no se pudo instalar normalmente con el administrador de paquetes de Python. En particular, este problema surgió con el paquete de `PyTorch` [38], que es el utilizado para la parte de las CNNs. Hubo que conseguir binarios ya compilados para la arquitectura del procesador de la Raspberry para poder instalarlo.
- **Conexión SSH**: La conexión remota al dispositivo se hizo estando las Raspberry y la PC dentro de la misma LAN (Local Area Network). Si se quisiera conectarse desde otro lado, habría que configurar el Router donde se encuentra la Raspberry para que envíe los pedidos SSH a la IP asignada a la Raspberry. Además habría que darle un DNS a la red donde esté conectada la Raspberry.

Capítulo 8

Realizaciones del brazo

En este capítulo se muestran algunos ejemplos de dibujos realizados por el brazo con sus respectivos pre-procesamientos. Para cada imagen se hizo cuatro dibujos con los cuatros estilos definitivos. El formato de las figuras es igual para todas y es el siguiente:

- **Primera fila:** Se muestra la imagen original seguida de sus transformaciones: portrait, drawing_v1, drawing_v2, de izquierda a derecha.
- **Segunda fila:** Se muestra el siguiente paso con respecto a la imagen que se tenga arriba. En el caso de la imagen se hace la detección de bordes (estilo PARRA) y para las transformaciones se hace detección de ridges.
- **Tercera fila:** Se muestran las mismas curvas que en la segunda fila pero ahora agregando sombra.
- **Cuarta fila:** Se muestran imágenes escaneadas de los dibujos realizados por el brazo a partir de las curvas. En cada caso se escogió la opción con o sin sombras.

Para varios de estos dibujos se hicieron grabaciones del brazo dibujando. Los videos se pueden encontrar en el **canal de YouTube del proyecto**[1]. En este canal se pueden encontrar también videos de otras pruebas de dibujo, o más específicas, como lo son pruebas del funcionamiento de los motores, movimientos con distintos ajustes PID. En particular, los videos de estos dibujos se encuentran en una **lista de reproducción** con dibujos variados de personas[2].

[1]Canal de YouTube de PicassoBotZ: `https://www.youtube.com/channel/UC_RwOw_3f1VHXfqaSbZ7LFA`

[2]Lista de reproducción con dibujos de personas: `https://youtube.com/playlist?list=PLEQhiSLIfUNdkBvNkDt3G3G72I3qyKaQK`

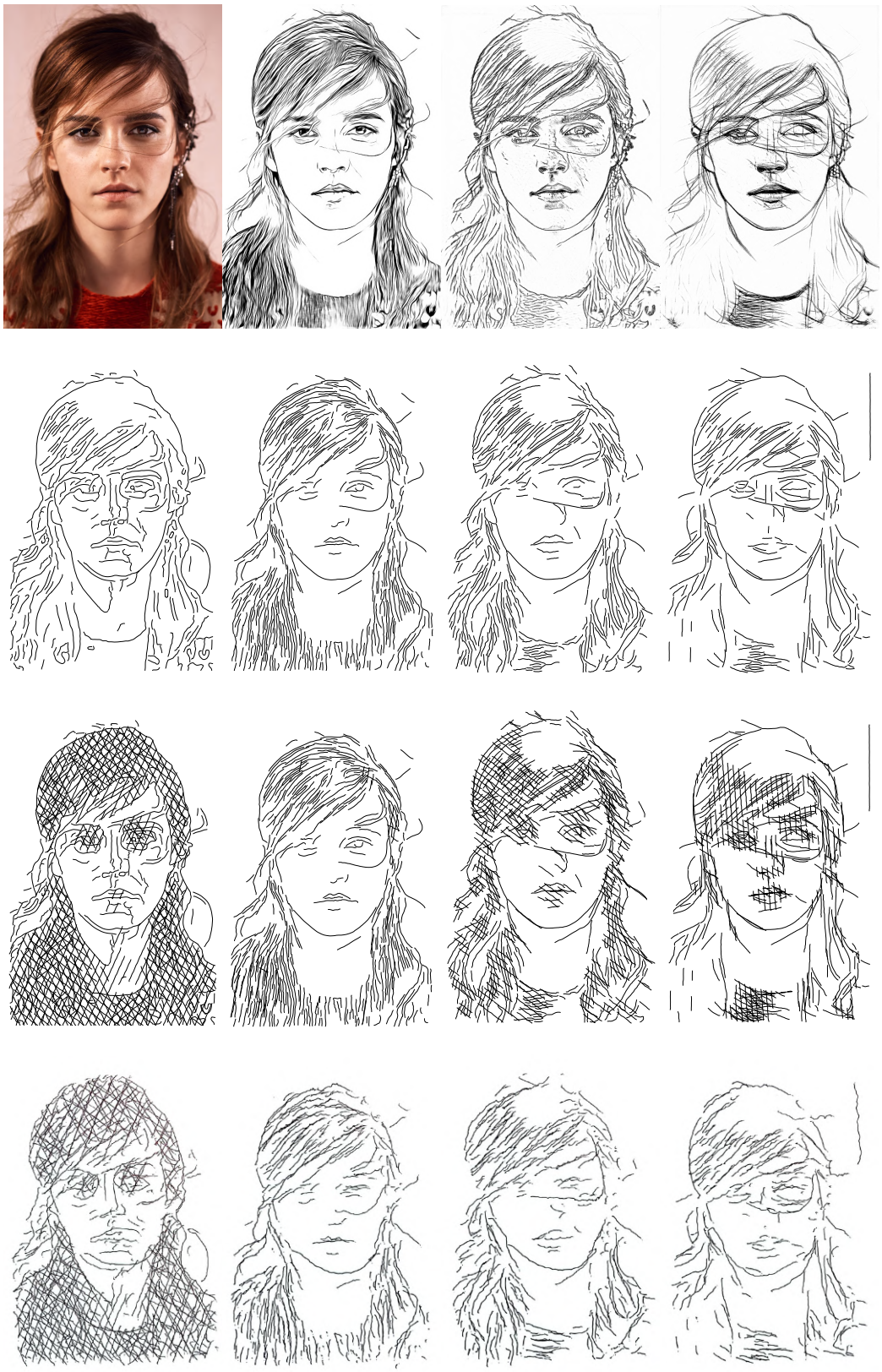


Figura 8.1: Retratos de Emma Watson. Para las realizaciones se escogió (de izquierda a derecha): Parra con sombras, portrait sin sombras, Informative drawing v1 sin sombras, Informative drawing v2 sin sombras.

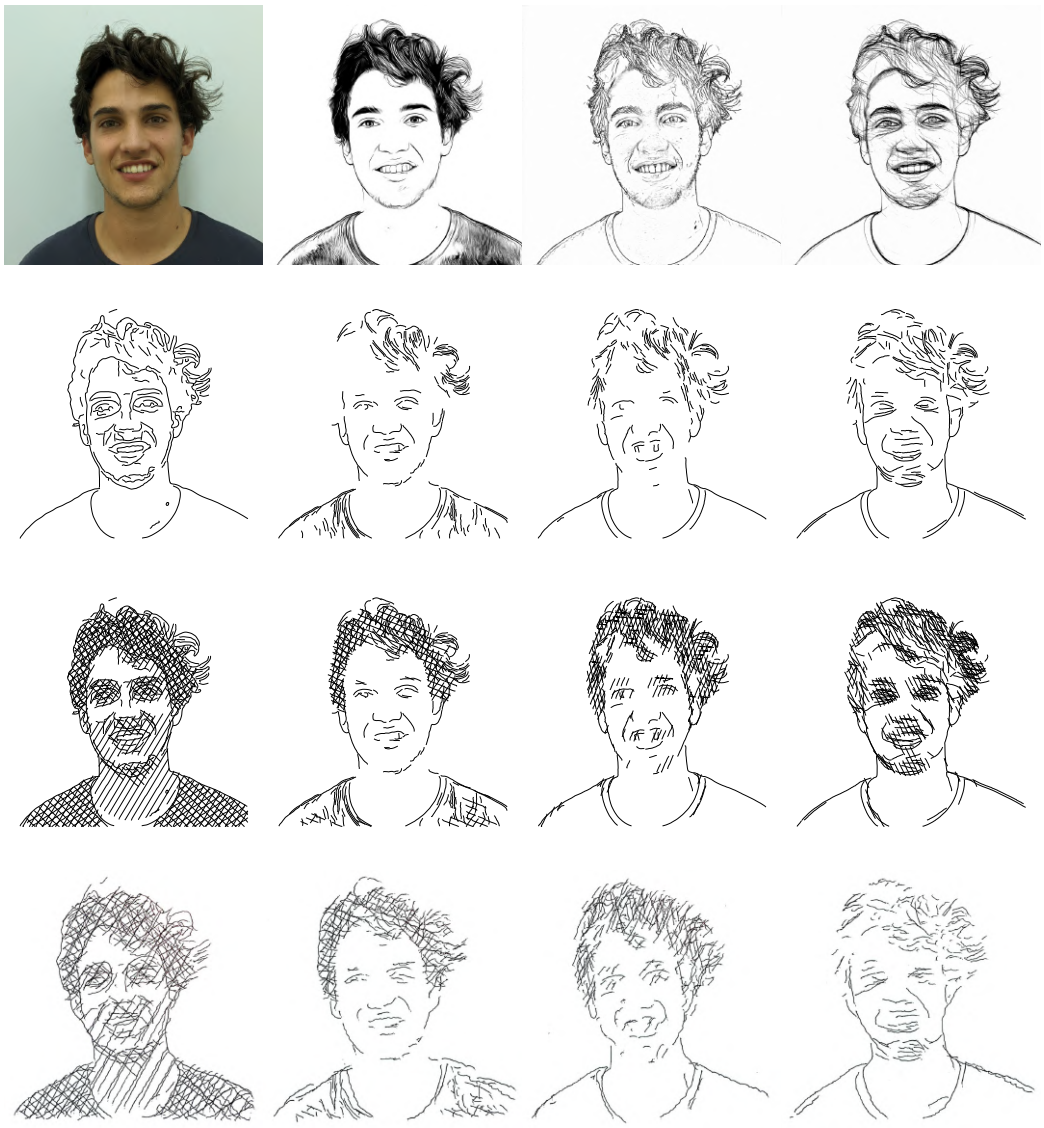


Figura 8.2: Retratos de Diego. Para las realizaciones se escogió (de izquierda a derecha): Parra con sombras, portrait con sombras, Informative drawing v1 con sombras, Informative drawing v2 sin sombras.

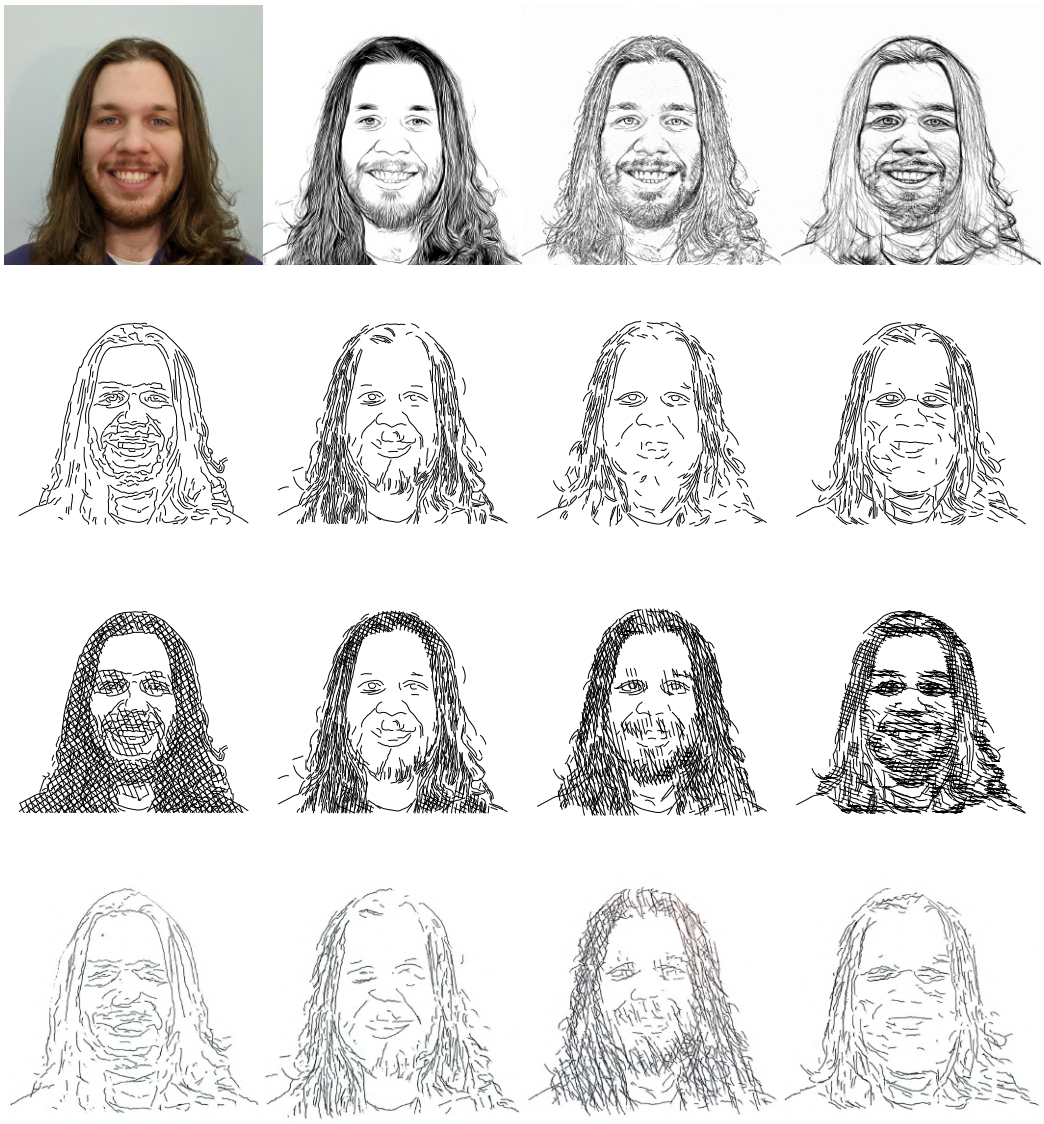


Figura 8.3: Retratos de Santiago. Para las realizaciones se escogió (de izquierda a derecha): Parra sin sombras, portrait sin sombras, Informative drawing v1 con sombras, Informative drawing v2 sin sombras.

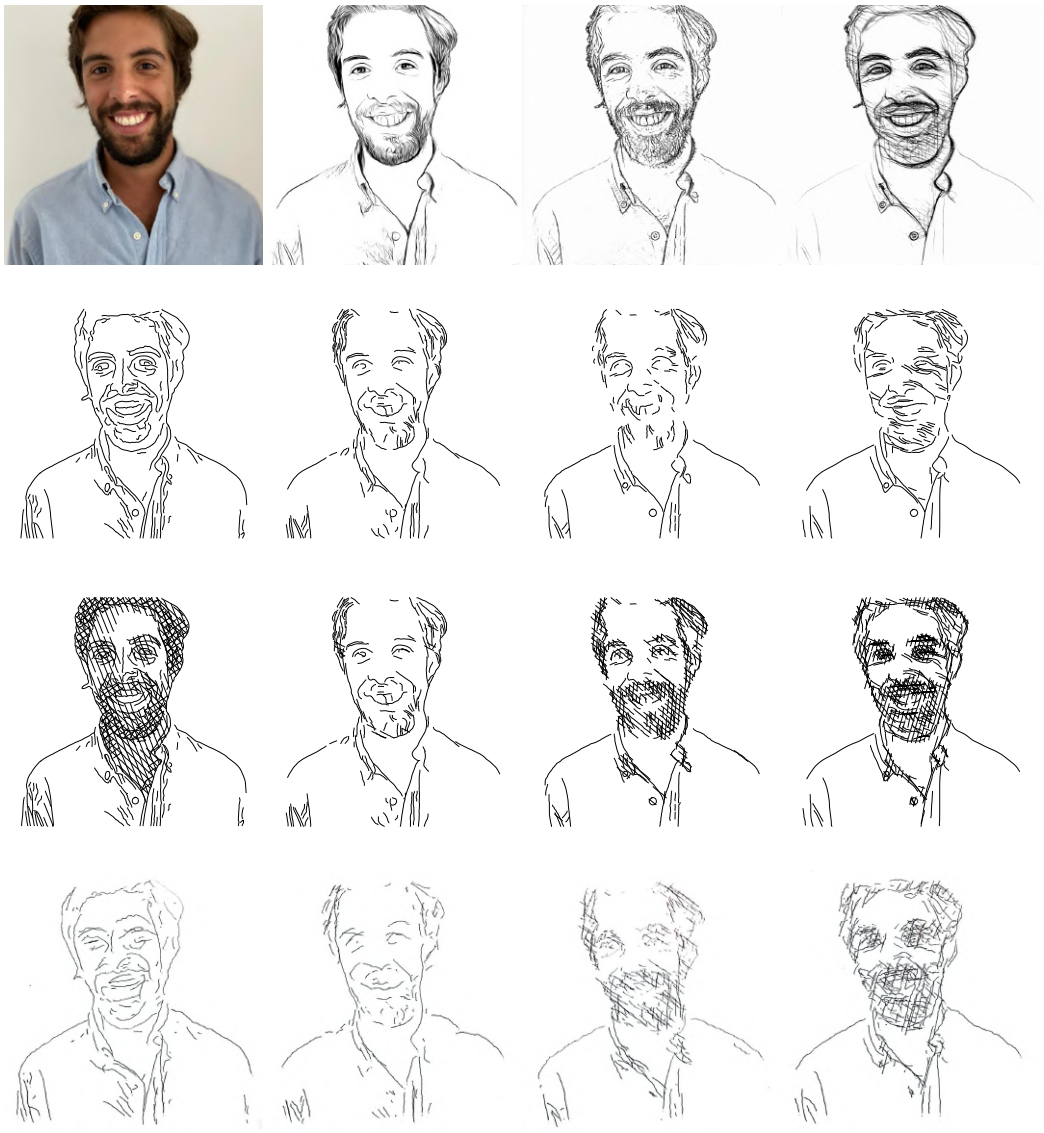


Figura 8.4: Retratos de Juan Pablo. Para las realizaciones se escogió (de izquierda a derecha): Parra sin sombras, portrait sin sombras, Informative drawing v1 con sombras, Informative drawing v2 con sombras.

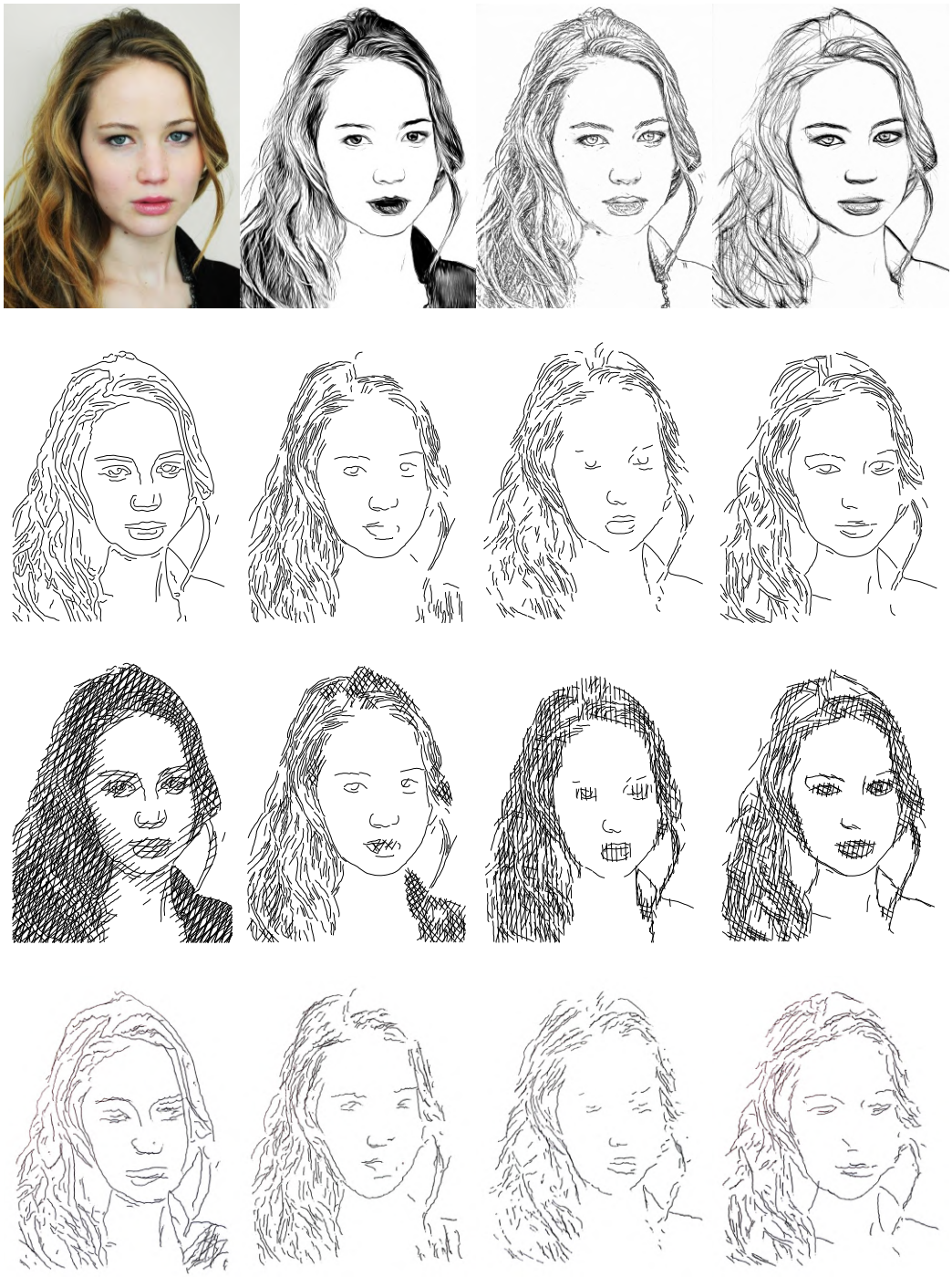


Figura 8.5: Retratos de Jennifer Lawrence. Para las realizaciones se escogieron todas las opciones sin sombra.

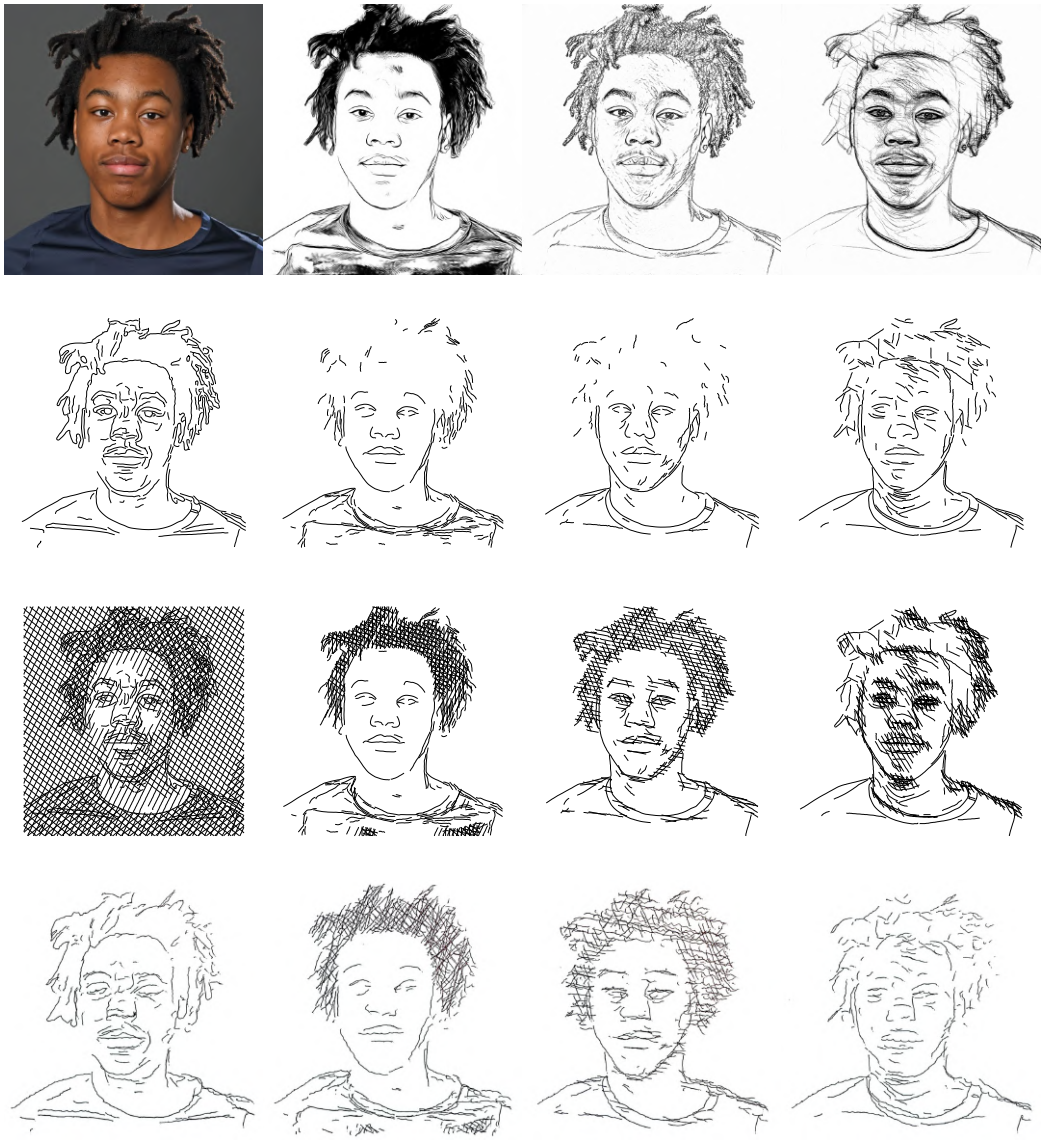


Figura 8.6: Retratos de Scottie Barnes. Para las realizaciones se escogió (de izquierda a derecha): Parra sin sombras, portrait con sombras, Informative drawing v1 con sombras, Informative drawing v2 sin sombras.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 9

Conclusiones

9.1. Conclusiones generales

Se logra desarrollar un sistema completo de punta a punta unificando las ideas de ambos proyectos anteriores. A partir de una imagen capturada por una cámara, se tienen distintos estilos y opciones para hacer una representación mediante un conjunto de curvas. Se logra traducir estas curvas a movimientos de un brazo robótico con un algoritmo nuevo. También se construyó y ajustó el brazo como sistema físico que se comanda para poder ejecutar los dibujos.

Para cada una de las distintas partes se hizo uso del trabajo y la experiencia de los proyectos anteriores. En algunos casos hubo que reformular algunas soluciones para poder incorporarlas al sistema final, mientras que en otros se logró adaptar el trabajo existente e incluso mejorarlo.

En el caso de la etapa de procesamiento de imágenes, se pudo construir sobre lo que PARRA ya tenía hecho. La extracción de curvas que propusieron con la detección de bordes y sombreado funciona perfectamente en el sistema final. Incluso esto inspiró la metodología utilizada en los nuevos estilos propuestos, solo que haciendo la detección de curvas con un algoritmo diferente que se ajusta mejor a los resultados de estos estilos. La técnica de sombreado se utiliza tal cual la implementó PARRA también para estos casos.

Por otro lado, para la etapa de path planning también se tomó el trabajo realizado por PARRA pero introduciendo cambios más fundamentales. El algoritmo propuesto por este equipo para planificar el trazado de una curva se replantea, introduciendo conceptos nuevos como el c-space. Sin embargo, se tiene en común el uso de *inverse kinematics* y el modelado de los movimientos del brazo con velocidades angulares constantes. El hecho de traducir la curva entera al espacio de configuración asegura la continuidad en los ángulos. Esto da fluidez en los trazos y hace que los movimientos sean mucho más naturales, lo cual era uno de los principales objetivos del proyecto.

Con respecto al hardware, se introducen dos grandes diferencias con respecto al equipo PicassoBot: El uso de motores MX-12W en lugar de AX-12A y la morfología del brazo. Ambos cambios sugeridos por el equipo PicassoBot como trabajo a

futuro. En este proyecto se decidió incorporar el modelo de motores MX que es más nuevo y cuenta con mejores prestaciones. Esto implicó sintonizar los controladores PID de cada motor, de forma de tener una fidelidad en los trazos al realizar los dibujos. También se reconstruyó el brazo, cambiando su morfología para que se parezca más a la de un humano.

Se descubrieron e idearon nuevos algoritmos, para los cuales se debió estudiar y aprender sobre áreas de conocimiento nuevas y específicas. Tal es el caso de los nuevos estilos en la etapa de procesamiento de imágenes, donde se incorporan tecnologías novedosas como las CNNs pertenecientes al área de *Computer Vision*. Otro caso es el algoritmo de path planning ideado, para el cual se estudió en profundidad técnicas de *inverse kinematics* y el *c-space*, ambas pertenecientes al área de la robótica.

9.2. Trabajo a futuro

A lo largo del proyecto se observó que Tresset implementó nuevas versiones de su brazo robótico. Según sus redes sociales, se puede ver que el brazo actual cuenta con solo dos motores modelo AX-12A (el que sostiene el lápiz y el anterior). Los motores del hombro y el codo ahora son de la linea MX, más precisamente son MX-64AT. Por estar razón, una posible mejora en el hardware podría ser la utilización de diferentes modelos de motores según cada articulación.

También se vio viable cambiar el funcionamiento del brazo, de manera que se agregara un joystick y se pudiera mover el brazo en tiempo real. Siguiendo por esta línea de pensamiento se podría incorporar una tablet digitalizadora, para que el brazo vaya dibujando en tiempo real lo que se dibuje en esta. Esto es posible debido a que el programa del Arduino ejecuta de a un comando enviado y no espera tener un conjunto de comandos previo a dibujar.

Por otro lado, en un sistema final con una Raspberry, se podría usar sus pines GPIO con los motores y de esta forma independizarse de la necesidad de usar un Arduino. Esta versión generaría un sistema más simple.

Por último, se propone la idea de la cámara, no solamente para que tome fotos, sino para que haga una realimentación en tiempo real del dibujo para ir generando correcciones de este y este se parezca mas a la foto inicial.

Apéndice A

Algoritmos no definitivos para el Path Planning

En el capitulo 5 se describe un método para traducir las curvas en los movimientos definidos para el brazo. Como se explica en la sección 4.1.2, los movimientos del brazo son a velocidad constante, es decir que teóricamente se modela las velocidades angulares de los motores como pulsos en el tiempo (todas con el mismo soporte). Este tipo de movimientos son curvas que se parametrizan fácilmente como en la ecuación (4.4). El desafío está en describir cada curva con movimientos de este tipo, siendo lo más fiel posible a la curva. El proyecto PARRA había trabajado en esto y propuso dos soluciones teóricas. En este proyecto también se consideró otra opción antes de la solución definitiva.

A.1. Algoritmos del PARRA

El proyecto PARRA modelaba los movimientos de la misma forma. Por ende, necesitaban una posición inicial y final, y una velocidad angular con la cual trasladarse. Cada curva se describía con un solo movimiento, pero las curvas que estos pueden describir son limitadas. La solución que se halló a este problema fue partir las curvas que superaran un cierto largo máximo, ya que las curvas cortas son posibles de representar por los movimientos a velocidad constante. Siendo estrictos, las curvas cortas se pueden representar con un error lo suficientemente pequeño como para no ser percibido.

Una vez que el conjunto de curvas estaba conformado por curvas cortas, se procedía a parametrizarlas. El problema de partir las curvas largas era tratarlas luego como curvas independientes, porque como su parametrización no era continua, al dibujarlas se perdía también la continuidad en los movimientos. No es solo por lograr naturalidad en el brazo que se quiere continuidad en los movimientos sino que también porque los trazos son más precisos que si se dibujan como distintos segmentos independientes.

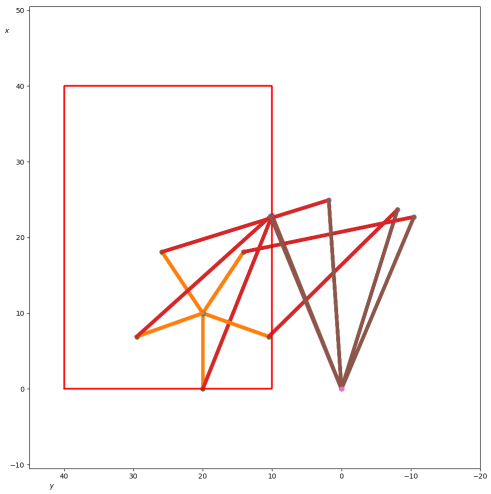
De todos modos, los métodos propuestos por PARRA son válidos y, en el marco teórico, encuentran fieles representaciones de las curvas.

A.1.1. Método con tensores

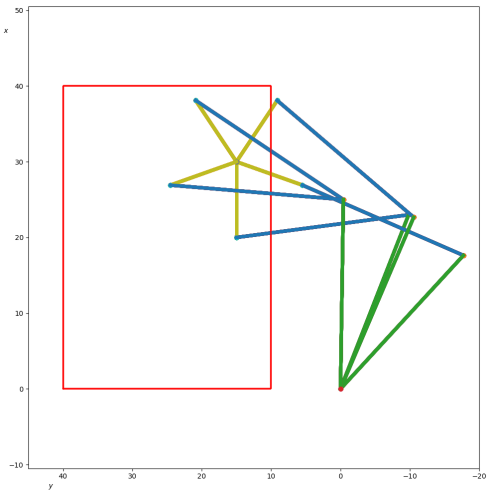
El primer método ideado por PARRA era exhaustivo y probaba con un conjunto denso de combinaciones posibles para la configuración de ángulos de los motores. Se armaban tensores con una grilla de las combinaciones de los tres ángulos para probar cuál era la mejor solución. La idea es que para cada curva se toman el punto inicial y final, y se aplica *forward kinematics* para todas las combinaciones de los tensores quedándose con las N configuraciones que más se acerquen al punto inicial y las M que más se acerquen al punto final. Dado esto se tiene $N \times M$ combinaciones de configuraciones iniciales y finales, cada una definiendo una parametrización, ya que las velocidades están dadas por el arco que deba recorrer cada ángulo en un intervalo de tiempo fijo para los tres motores. Luego de que se tienen estas parametrizaciones se elige la que tenga menor error con la curva teórica que se quiere dibujar. Con valores razonables de N y M (se usaba alrededor de 20 para cada uno) y curvas no muy largas se encontraban parametrizaciones fieles de la curva teórica. La desventaja de este método es que es poco eficiente y lento comparado con otros porque se prueba con un número grande de configuraciones innecesariamente, cuando en realidad se puede usar la cinemática inversa para resolver esa parte de forma más eficiente.

A.1.2. Método con *inverse kinematics*

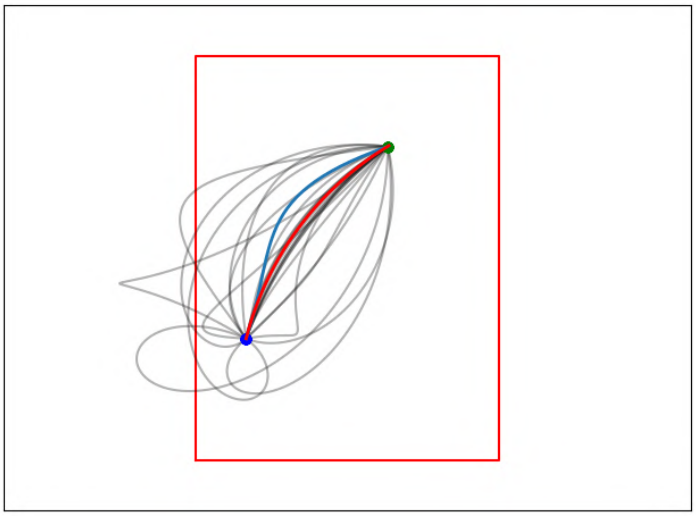
El método definitivo que propuso PARRA fue con *inverse kinematics*. Básicamente es la misma solución, solo que ahora se hallan las $N \times M$ combinaciones de configuraciones iniciales y finales resolviendo el problema con IK. No se hizo IK para las tres juntas si no que se simplificó un poco el problema dividiendo el brazo en dos: El brazo y el antebrazo como un brazo de 2 juntas y la mano con una sola junta. La idea era que dado un punto del plano (que sería el punto inicial y final de una curva en este caso) se encuentran N configuraciones proponiendo N posiciones para la muñeca en una circunferencia alrededor del punto en cuestión. Luego para cada una de esas N posiciones de la tercera junta, se resuelve IK usando la primera parte del brazo con 2 juntas. Esto se ilustra en la figura A.1Lo interesante es que la solución de IK para un brazo de 2 juntas se puede hallar analíticamente con trigonometría. Una vez halladas las N configuraciones iniciales y las M finales, se procede de la misma forma que en el método anterior. Esta solución es más eficiente ya que hacer uso de la cinemática inversa acelera drásticamente el proceso y no es necesario hacer una búsqueda exhaustiva, además de que se encuentran soluciones de forma cerrada. Además se asegura que las configuraciones encontradas son solución y no una aproximación de la misma como lo eran en el método anterior. Una explicación más detallada de este método se puede leer en la documentación de PARRA [2].



(a) $N = 5$ posibles configuraciones iniciales



(b) $M = 5$ posibles configuraciones finales



(c) En **negro**, $N \times M$ curvas que se pueden realizar. En **azul**, la curva objetivo que se quiere representar y en **rojo**, la curva que mejor aproxima

Figura A.1: Ejemplo del método con *inverse kinematics* utilizado por PARRA para parametrizar una curva. En este caso se usa $N = M = 5$ para ilustrar, los resultados son mejores cuando se usan valores más grandes de N y M. Imágenes extraídas de la documentación de PARRA [2].

A.2. Algoritmos no definitivos en este proyecto

A.2.1. Descenso por gradiente

Tomando como base la metodología propuesta por PARRA, inicialmente se apostó por mejorar esa solución. El cambio fue que en vez de tomar un conjunto de parametrizaciones y evaluar cuál es la que tiene menor error, se trabaja con una parametrización genérica de la curva y se optimiza la configuración de ángulos inicial y las velocidades angulares para minimizar el error con la curva teórica objetivo. La curva parametrizada se describe en el plano siguiendo las ecuaciones planteadas en (4.4). En este caso, $\begin{bmatrix}\theta_0 & \varphi_0 & \psi_0\end{bmatrix}$ sería la configuración inicial de ángulos y $\begin{bmatrix}\omega_\theta & \omega_\varphi & \omega_\psi\end{bmatrix}$ serían las velocidades angulares. Cabe destacar que la se toma la parametrización de la curva con $t \in [0, 1]$, y de esta manera la configuración final de ángulos se puede obtener sumando las velocidades a la configuración inicial.

La curva teórica que se quiere representar es un conjunto discreto de puntos, ya que se trabaja con señales digitales. Ahora bien, la parametrización planteada es de una curva continua. Para hacer el error entre ambas curvas se muestrea la parametrización con la misma cantidad de puntos que la curva teórica, en intervalos uniformes. Entre cada par de puntos se hace la distancia y luego se toma la distancia media como error. La idea es optimizar $\begin{bmatrix}\theta_0 & \varphi_0 & \psi_0\end{bmatrix}$ y $\begin{bmatrix}\omega_\theta & \omega_\varphi & \omega_\psi\end{bmatrix}$ para minimizar este error. Para ello, se toma un valor inicial para estos y se itera haciendo descenso por gradiente. Esto implica hacer el gradiente del error con respecto a los parámetros y actualizar los mismos usando este gradiente.

El problema con este método es que se minimiza el error de la toda curva en conjunto, punto a punto, pero no se asegura que el inicio y el final sean exactos. Si bien en la teoría el error era muy pequeño (en el orden de uno o dos milímetros), en la práctica esos errores se ven amplificados por las no idealidades de los motores. Se podría haber trabajado más en mejorar este método, pero se avanzó en la idea de usar el método descrito en el capítulo 5, que daba mejores resultados.

Apéndice B

Extensiones del modelado e intuiciones sobre el c-space

En el presente apéndice se desarrollan resultados, visualizaciones y justificaciones acerca de las temáticas alrededor de la herramienta del c-space. Estas intuiciones no fueron incluidas en el desarrollo del documento principal por su nivel de minuciosidad, ya que distraerían del objetivo principal del proyecto. El cual es, en definitiva, hacer funcionar el brazo para que dibuje.

B.1. Puntos del plano alcanzables por el brazo

Para realizar la prueba de cuáles son los puntos alcanzables por la punta (*tooltip*) del brazo, se usará la parametrización y los ejes coordenados definidos en 4 para el brazo. Esta se puede refrescar en la figura B.1. Además, la prueba se hará de forma iterativa, probando cuáles son los puntos alcanzables por un brazo de 1, 2 y finalmente 3 articulaciones, de alguna forma "armando" el brazo de a articulaciones. Este razonamiento puede luego extenderse a n articulaciones. Además, el esquema planteado en esta prueba servirá para posteriormente caracterizar la topología de las curvas en el c-space que se corresponden a una posición del tooltip.

B.1.1. Puntos alcanzables por $\mathbf{r}_1$

Considérese el brazo formado solamente por el segmento L_1, con articulación en $\mathbf{r}_0$. Como se dijo en el capítulo 4, el punto $\mathbf{r}_0$ esta fijo al origen de coordenadas. Además, $\mathbf{r}_1$ está unido a $\mathbf{r}_0$ por un segmento de longitud constante L_1. Como el segmento L_1 puede rotar libremente sobre su extremo $\mathbf{r}_0$, resulta trivial que $\mathbf{r}_1$ pertenece a una circunferencia de centro $\mathbf{r}_0 = (0, 0)$ y radio L_1. Es decir, $\mathbf{r}_1 \in \mathcal{C}_{\mathbf{r}_0, L_1}$. Pueden verse estos puntos alcanzables por $\mathbf{r}_1$ en la figura B.2.

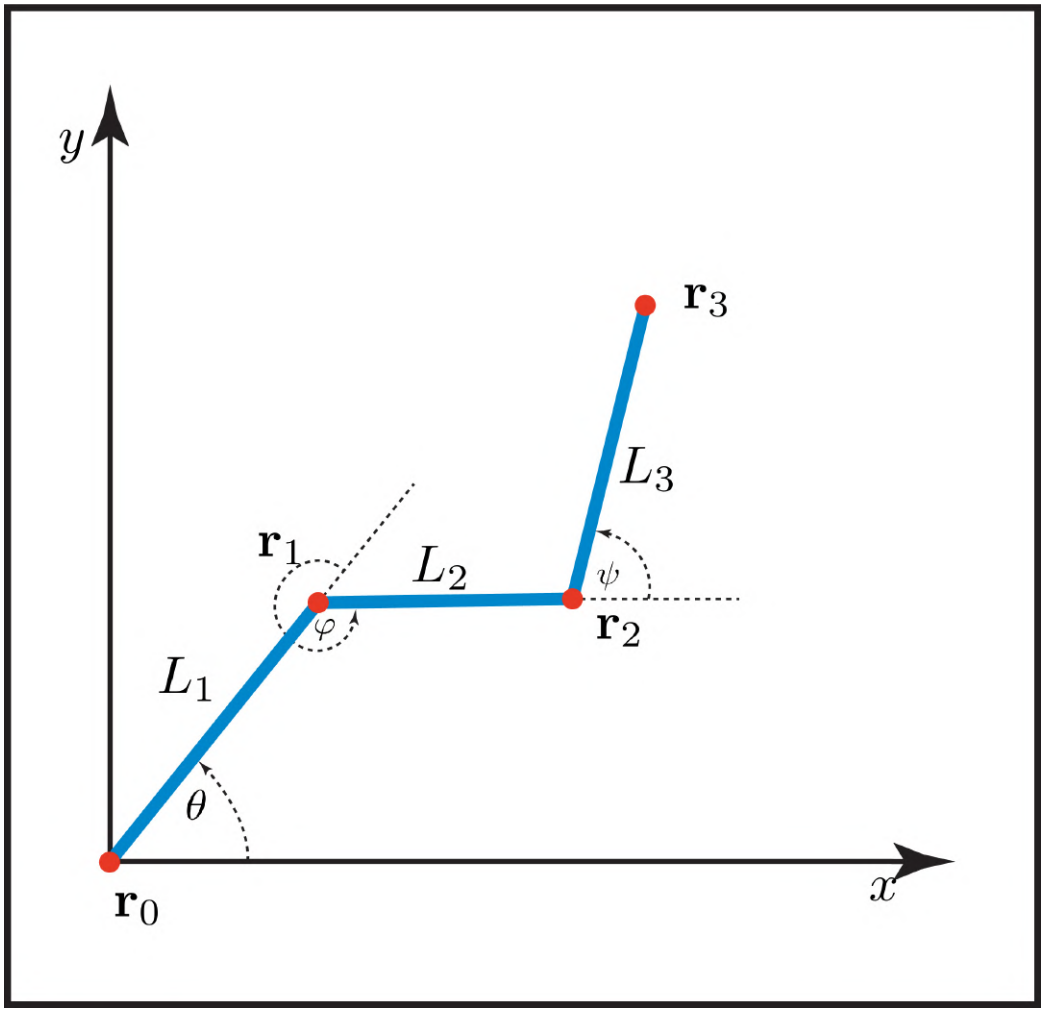


Figura B.1: Brazo geométrico y su parametrización completa en el plano.

B.1.2. Puntos alcanzables por $\mathbf{r}_2$

Ahora, considérense los segmentos L_1 y L_2 por separado. Por la parte anterior, los extremos de L_1 deben cumplir:

$$\begin{cases} \mathbf{r}_0 = (0,0) \\ \mathbf{r}_1 \in \mathcal{C}_{(0,0),L_1}. \end{cases} \tag{B.1}$$

Si se consideran los extremos del segmento L_2 sin las condiciones (B.1), estos extremos solamente deben verificar que $d(\mathbf{r}_1, \mathbf{r}_2) = L_2$. Entonces, si se fija $\mathbf{r}_2$ a un punto arbitrario del plano, se obtiene la condición sobre $\mathbf{r}_1$ siguiente:

$$\mathbf{r}_1 \in \mathcal{C}_{\mathbf{r}_2,L_2}. \tag{B.2}$$

Se puede ver entonces, que para que el brazo efectivamente se pueda formar, $\mathbf{r}_1$ debe de cumplir simultáneamente las condiciones (B.1) y (B.2). O lo que es lo

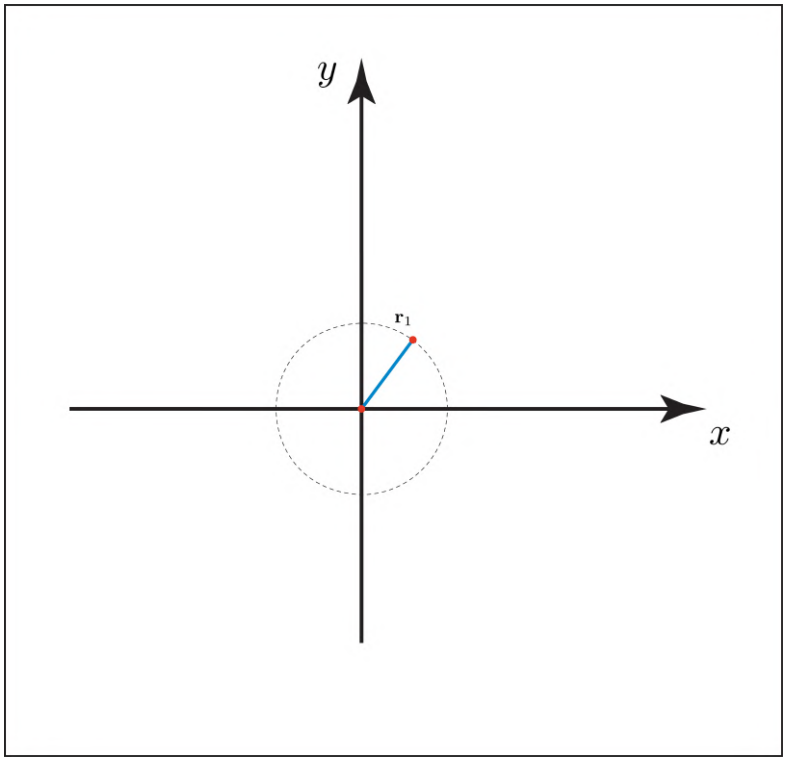


Figura B.2: Región de puntos alcanzables por la articulación $\mathbf{r}_1$. Representada por la circunferencia punteada.

mismo, $\mathbf{r}_1$ debe de pertenecer a ambas circunferencias $\mathcal{C}_{(0,0),L_1}$ y $\mathcal{C}_{\mathbf{r}_2,L_2}$. Y como la posición relativa de estas circunferencias solamente depende de la elección de un $\mathbf{r}_2$, el problema de encontrar los $\mathbf{r}_2$ factibles se convierte en un problema de hallar los centros de $\mathcal{C}_{\mathbf{r}_2,L_2}$ tales que $\mathcal{C}_{(0,0),L_2}$ y $\mathcal{C}_{\mathbf{r}_2,L_2}$ se intersecten en un conjunto no vacío de puntos.

En particular, como $L_1 > L_2$ para el brazo de este proyecto, se puede ver en la figura B.3 que para una semirecta radial dada, las circunferencias se intersectan en al menos un punto para $(L_1 - L_2) \leq \|\mathbf{r}_2\| \leq L_1 + L_2$

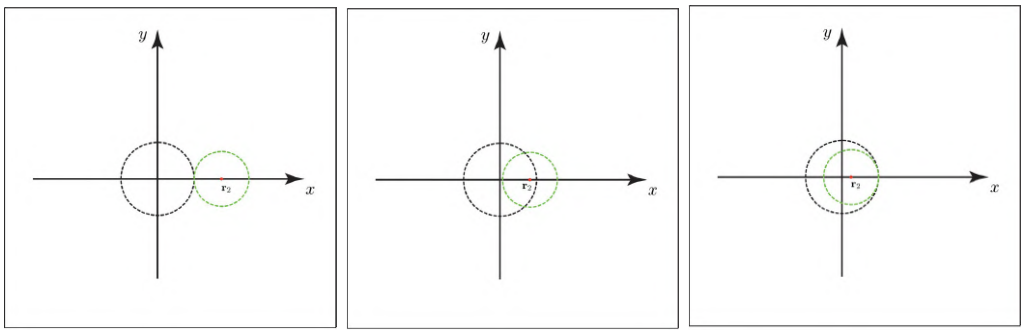
Cómo el problema tiene una simetría en el ángulo θ, si $(\rho\cos\theta, \rho\sin\theta)$ es alcanzable por $\mathbf{r}_2$, entonces también lo es $(\rho\cos(\theta+\delta), \rho\sin(\theta+\delta))$, $\forall\delta \in \mathbb{R}$. De esta forma, se puede realizar una rotación de la semirecta mostrada en B.3 para obtener toda la región de puntos alcanzables por $\mathbf{r}_2$. Un anillo de radio interior $L_1 - L_2$ y radio exterior $L_2 + L_1$. Esta región se puede ver en la figura B.4.

Para la siguiente demostración, convendrá denominar a dicha región de puntos alcanzables $\mathcal{R}_2$.

B.1.3. Puntos alcanzables por $\mathbf{r}_3$

Considerando ahora, por un lado el brazo compuesto por L_1 y L_2, y por otro lado el segmento L_3, se puede llegar a realizar un análisis análogo al anterior.

Considerando el brazo formado por L_1 y L_2, por la parte anterior, se deduce



(a) (b) (c)

Figura B.3: Posiciones relativas con intersección entre las

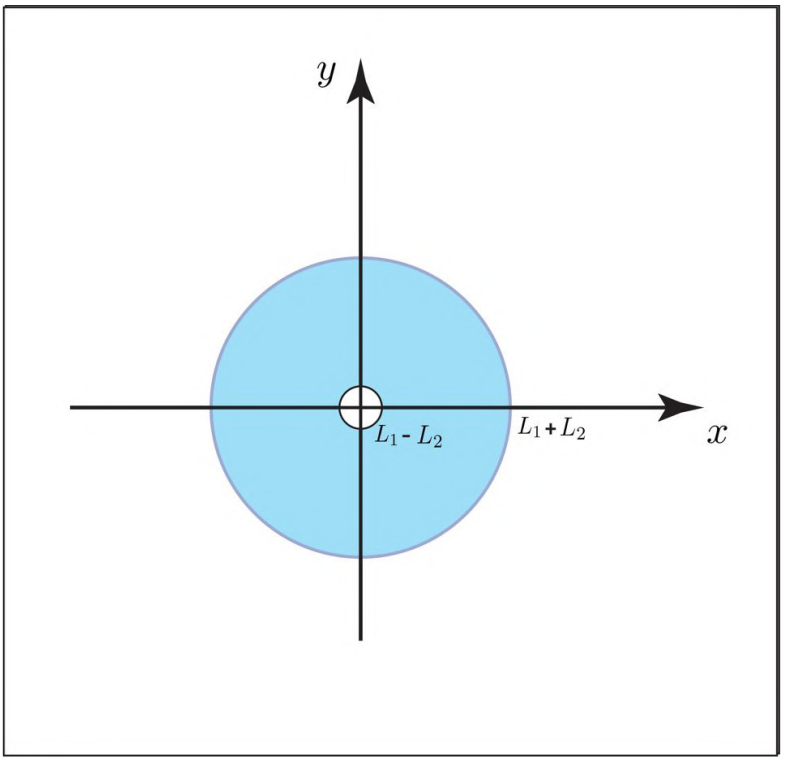


Figura B.4: Región de puntos alcanzables por la articulación $\mathbf{r}_2$. Representada por la region celeste entre las dos circunferencias.

que $\mathbf{r}_2 \in \mathcal{R}_2$. Además, considerando el segmento L_3 y fijando un valor arbitrario de $\mathbf{r}_3$, se tiene que $\mathbf{r}_2 \in \mathcal{C}_{\mathbf{r}_3, L_3}$. Entonces, los puntos alcanzables por $\mathbf{r}_3$ son aquellos que hacen que $\mathbf{r}_2$ verifique las dos condiciones anteriores a la vez. Es decir, que el problema se traduce a obtener los centros de las circunferencias para los cuales esta intersección es no nula. En la figura B.5 pueden ver todas las posiciones relativas distintas entre estas regiones. Obsérvese que al ser para este proyecto $L_3 \geq (L_1 - L_2)$, la circunferencia $\mathcal{C}_{\mathbf{r}_3, L_3}$ no puede ser contenida completamente

dentro del círculo de radio interior de $\mathcal{R}_2$ y, por lo tanto, las regiones se intersecan para radios arbitrariamente pequeños (incluyendo el cero). Finalmente, se puede ver la región de puntos alcanzables por $\mathbf{r}_3$ en la figura B.6.

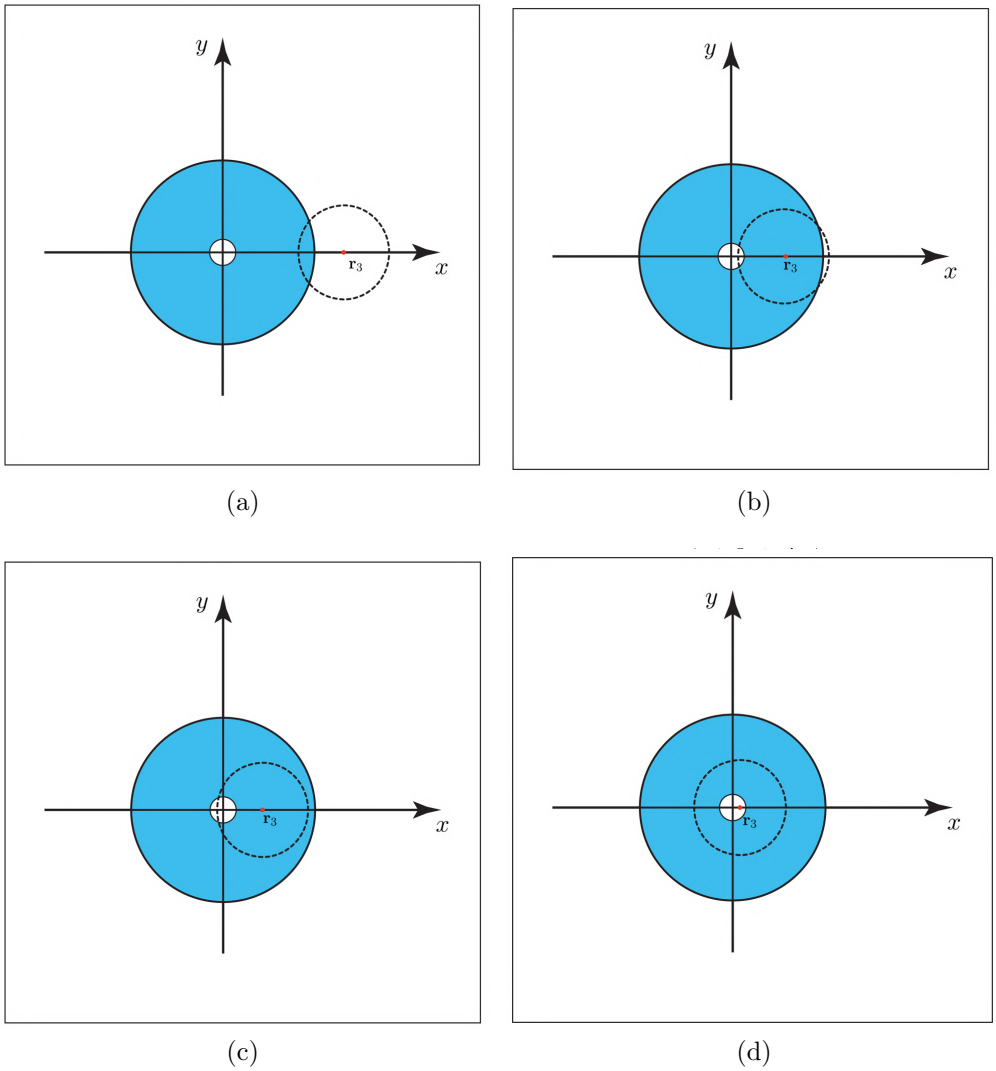


Figura B.5: Posiciones relativas de $\mathcal{C}_{\mathbf{r}_3,L_3}$ con respecto $\mathcal{R}_2$

B.2. Curvas de puntos alcanzables

En esta sección se responderá a la pregunta del capítulo 4: ¿Se puede trazar cualquier curva formada por puntos alcanzables de forma continua en los ángulos?

Como las curvas en el caso de este proyecto están formadas por un conjunto discreto de puntos, la demostración se centrará en ese caso. Es decir, ¿Se puede realizar una curva en el espacio (x, y) que pase exactamente por una cantidad

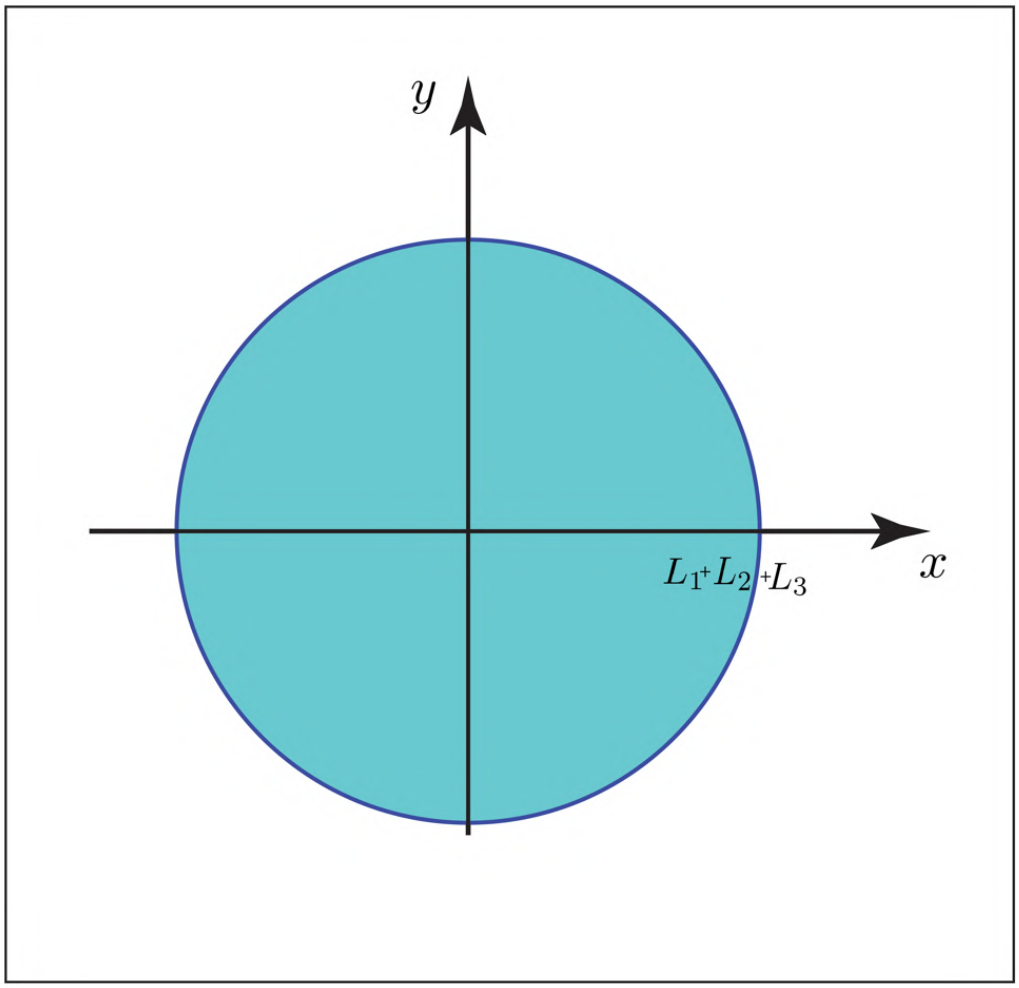


Figura B.6: Region de puntos alcanzables por $\mathbf{r_3}$.

arbitraria de puntos alcanzables? Lo que es más, bastará con probarlo solamente para un par de puntos.

Tomemos los primeros dos puntos de la curva discreta $\mathbf{s}_0$ y $\mathbf{s}_1$. Ambos puntos tendrán al menos un correspondiente en el c-space por ser alcanzables. Llámesele a estos correspondientes $\mathbf{q}_0$ y $\mathbf{q}_1$ respectivamente.

Al ser el c-space convexo (por ser S^3) se puede trazar un segmento h entre $\mathbf{q}_0$ y $\mathbf{q}_1$. Este segmento es evidentemente un conjunto conexo.

Luego, al aplicarle a este segmento la función de *forward kinematic* en (4.1), obtenemos una curva $FK(f)$ en el espacio (x, y). Y esta curva es conexa puesto que la curva original lo es y la función $FK(\mathbf{q})$ es continua.

Este razonamiento se puede repetir para el siguiente par de puntos consecutivos de la curva ($\mathbf{s}_1$ y $\mathbf{s}_2$), teniendo en cuenta que se debe de elegir el mismo representante $\mathbf{q}_1$ de $\mathbf{s}_1$ que para el caso anterior. De esta forma queda demostrado que efectivamente se puede realizar una curva en el espacio (x, y) que pase exactamente por una cantidad arbitraria de puntos alcanzables.

B.3. Representación de un punto (x, y) alcanzable en el c-space

En esta sección se probará que a un punto alcanzable por el brazo en el espacio (x, y) le corresponde (en general) una curva.

Recordamos las ecuaciones de inverse kinematic del capítulo 4, dada una posición del tooltip (x_3, y_3) y un ángulo total Φ se tiene que:

$$\begin{cases} \psi_l = \Phi - \theta_l - \varphi_l \\ \varphi_l = \alpha - \pi \\ \theta_l = \beta + \gamma, \end{cases} \tag{B.3}$$

$$\begin{cases} \psi_d = \Phi - \theta_d - \varphi_d \\ \varphi_d = \pi - \alpha \\ \theta_d = \beta - \gamma, \end{cases} \tag{B.4}$$

donde se cumple (4.7):

$$\begin{cases} x_2 = x_3 - L_3 \cos \Phi \\ y_2 = y_3 - L_3 \sin \Phi \\ \alpha = \arccos\left(\frac{L_1^2 + L_2^2 - x_2^2 - x_1^2}{2L_1 L_2}\right) \\ \beta = \arctan\left(\frac{y_2}{x_2}\right) \qquad \text{(en el cuadrante apropiado)} \\ \gamma = \arccos\left(\frac{L_1^2 - L_2^2 + x_2^2 + x_1^2}{2L_1\sqrt{x_2^2 + y_2^2}}\right) \end{cases} \tag{B.5}$$

Tomando solamente el caso de las soluciones brazo izquierdo en (B.3), se ve que para un único punto (x_3, y_3), los ángulos θ, ϕ y ψ quedan parametrizados en Φ. O al menos en los intervalos de este parámetro en los cuales existan soluciones IK.

B.4. Topología de las curvas en el c-space

En esta sección se caracterizará la topología de las curvas correspondientes a la IK de un punto alcanzable en el espacio (x, y) así como fue expuesto en el capítulo 4. Es decir, que el IK de un punto del espacio (x, y) en el espacio S^3 puede ser solamente:

- una curva cerrada,
- dos curvas cerradas.

Para esto primero se demostrará que las curvas de IK para distintos puntos no se pueden cruzar, de esta forma se independizan unas curvas de otras. Luego, para puntos en determinadas regiones del espacio (x, y) se realizará el análisis de, dados los posibles puntos de todas sus articulaciones en el espacio (x, y), cómo pueden ser las curvas en el c-space. Además, para esto se hará uso de las ecuaciones de IK (B.4), (B.3) y (B.5).

B.4.1. No intersección de curvas

Es sencillo ver que dos curvas correspondientes en el c-space a dos puntos distintos en el espacio (x, y) no se pueden cruzar.

Sean los dos puntos en el espacio (x, y), (x_1, y_1) y (x_2, y_2). Además, sean $\mathbf{q}_1(\Phi)$ y $\mathbf{q}_2(\Phi)$ su representación en el c-space.

Si existe un punto $\mathbf{s}_0$ que pertenece tanto a $\mathbf{q}_1(\Phi)$ como a $\mathbf{q}_2(\Phi)$, entonces la representación de dicho punto en el espacio (x, y) debe ser (x_1, y_1) y (x_2, y_2). Pero cpmo la relación desde el c-space al espacio (x, y) se expresa mediante la función FK, un punto en el c-space no puede representar múltiples puntos en el espacio (x, y). Lo que implica que el punto $\mathbf{s}_0$ no pueda existir y, a su vez, que las curvas en el c-space para distintos puntos en el espacio (x, y) son disjuntas.

B.4.2. Topología individual de las curvas

La topología de cada una de las curvas se caracterizará a partir de la posición de su punto correspondiente en el espacio (x, y). Y, como se verá a continuación, la topología de las mismas solo dependerá de la posición relativa entre la circunferencia $\mathcal{C}_{\mathbf{r}_3,L_3}$ y la región de puntos alcanzables para $\mathbf{r}_3$. Es decir, las cuatro posibilidades que se muestran en B.5.

Para esto primero se debe observar que, dada una posición fija alcanzable de $\mathbf{r}_3$, se tiene (al menos) un arco de circunferencia de posibles posiciones para $\mathbf{r}_2$ (ver figura B.7). Pero, dado un $\mathbf{r}_2$ perteneciente a este arco (o lo que es lo mismo, fijar un valor para Φ), hay solamente dos posibilidades para la posición de $\mathbf{r}_1$, las correspondientes soluciones *brazo izquierdo* y *brazo derecho*. Cada una de estas soluciones se corresponde a un triángulo que se puede formar entre los puntos $\mathbf{r}_0$, $\mathbf{r}_1$ y $\mathbf{r}_2$ Estas dos soluciones son las expresadas en las ecuaciones de IK.

Solamente a partir de estas ecuaciones se puede ver que cada solución (brazo izquierdo y brazo derecho) se corresponde a una curvan en el c-space.

De ahora en más, se separará la prueba en cuatro casos, que son las posiciones relativas que puede tomar la curcunferencia $\mathcal{C}_{\mathbf{r}_3,L_3}$ con respecto a $\mathcal{R}_2$. Como ya vimos, estas posiciones relativas dependen solamente de la distancia a la cual se encuentre $\mathbf{r}_3$ del origen, por lo que se nombrarán de la siguiente forma:

- **Caso** 1: $\|r_3\| \in [13.9, 30.5]$
- **Caso** 2: $\|r_3\| \in [9.1, 13.9]$
- **Caso** 3: $\|r_3\| \in [7.5, 9.1]$
- **Caso** 4: $\|r_3\| \in [0, 7.5]$.

Los valores extremos de estos intervalos son completamente dependientes de las longitudes de los tres segmentos que conforman el brazo. Y los aquí expuestos son los correspondientes a el brazo con el que se trabajó en este proyecto.

Caso 1

Para este caso se tiene que las soluciones brazo izquierdo y brazo derecho coinciden para dos puntos. Ambas intersecciones entre el $\mathcal{C}_{\mathbf{r}_3,L_3}$ y la circunferencia exterior de la región $\mathbf{R}_2$. De esta forma, ambas curvas se juntan y se genera una sola.

Caso 2

Para este caso, ahora se tienen dos arcos de circunferencias, en cada uno de los cuales se va a tener que las soluciones brazo derecha y brazo izquierda coinciden también en un par de puntos. Entonces se van a tener dos curvas cerradas.

Caso 3

Para este caso se tiene lo mismo que en el Caso 1, en el que las soluciones brazo izquierdo y brazo derecho coinciden en dos puntos.

Caso 4

En este caso se tiene algo un poco más raro. Se observa que no hay punto de intersección entre $\mathcal{C}_{\mathbf{r}_3,L_3}$ y los bordes de la región alcanzable por $\mathbf{r}_2$. De esta forma no se van a poder juntar las soluciones brazo izquierdo y brazo derecho. Pero, considerando el ángulo Φ, se ve que este puede rotar indefinidamente, por lo que para cada solución, brazo izquierdo y brazo derecho, se tiene una curva cerrada que recorre la totalidad del ángulo Φ.

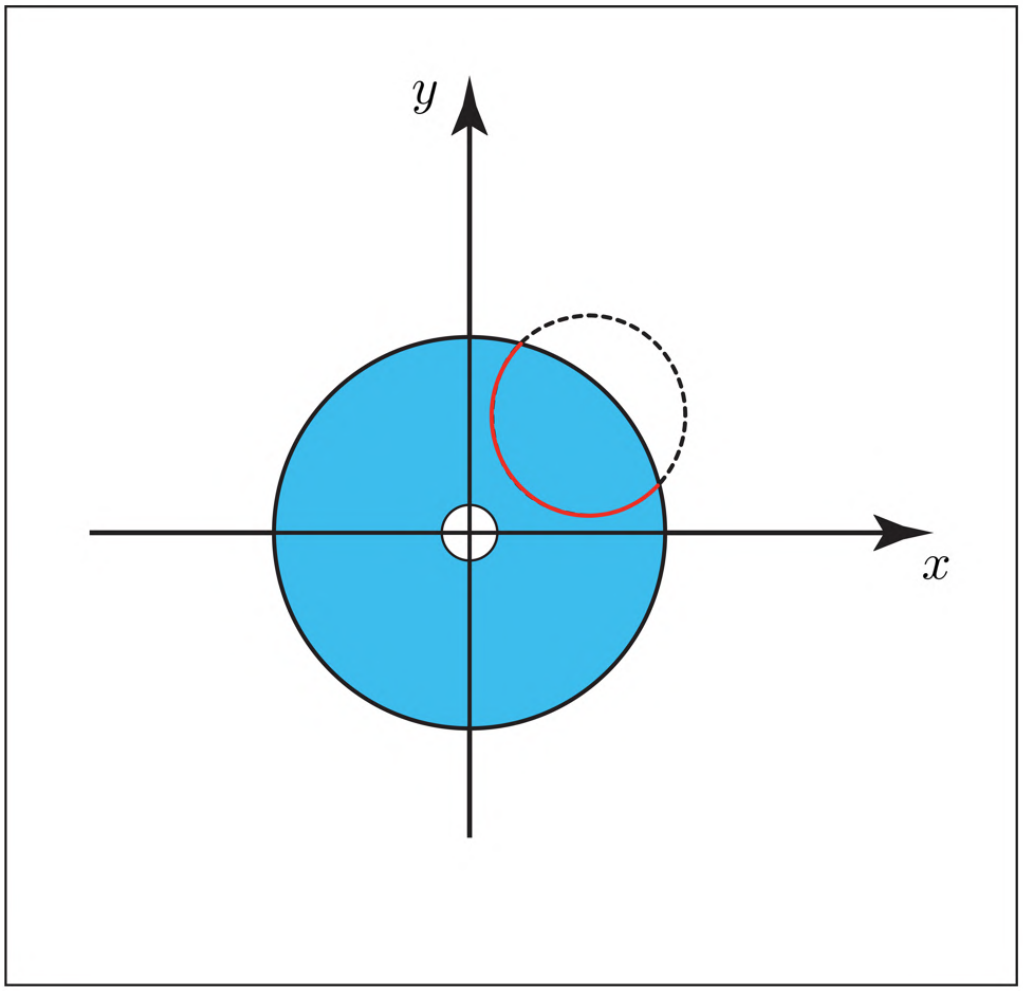


Figura B.7: Muestra el arco de circunferencia que contiene a todas las posiciones posibles de $\mathbf{r}_2$, dado $\mathbf{r}_3$.

Apéndice C

Pseudocódigo del algoritmo RDP

En el presente apéndice se presenta sencillamente un pseudocódigo del algoritmo RDP utilizado en la etapa de path planning de este proyecto.

Algoritmo 2: Pseudocódigo del algoritmo rpd. Consultado en [27]

Input : **P**: Poligonal en el c-space.
ε: ε del algortimo RDP

Output: **p**: submuestreo de la poligonal original

```
rdp(P, ε)
    dmax = 0
    index = 0
    end = length(P)
    for i = 2 to (end - 1) do
    |   d = perpendicularDistance(P[i], Line(P[1], P[end]))
    |   if d > dmax then
    |   |   index = i
    |   |   dmax = d
    |   end
    end
    p = [ ]
    if dmax > ε then
    |   recResults1 = rdp(P[1...index], ε)
    |   recResults2 = rdp(P[index...end], ε)
    |   p = recResults1[1...length(recResults1) - 1],
    |     recResults2[1...length(recResults2)]
    else
    |   p = P[1], P[end]
    end
    return p
```

Esta página ha sido intencionalmente dejada en blanco.

Apéndice D

Controladores

En este apéndice se presentan con mayor detalle los controladores que conforman el PID. También se discute sobre un método empírico para encontrar las constantes del PID.

D.1. Componentes de un controlador PID

D.1.1. Control proporcional

Es un sistema de control donde se encuentra una relación lineal y continua entre el valor de la variable controlada y el elemento final de control. El termino proporcional lleva la posición de error a cero,

$$u(t) = K_p e(t)$$

, donde K_p es la ganancia proporcional, $u(t)$ es la entrada de control y $e(t)$ es el error en el tiempo t. Este error se calcula como $e(t) = r(t) - y(t)$, donde $r(t)$ es el punto de referencia y $y(t)$ es la salida del sistema.

La figura D.1 muestra un diagrama de bloques para un sistema controlado por un controlador proporcional.

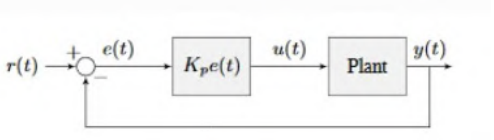


Figura D.1: Diagrama de bloques de un controlador proporcional.

La ganancia proporcional da la relación entre la variable de salida del controlador y la variación de la señal de entrada. Si la ganancia es grande, demuestra sensibilidad ya que si cambia levemente la entrada, genera un cambio grande en la salida. Este control presenta un error en estado estable, es un error ocasionado cuando el tiempo tiende a infinito que es cuando la variable nunca llega al valor de referencia.

D.1.2. Control derivativo

La salida del controlador viene dada por la relación de la derivada del error respecto al tiempo, multiplicado por una constante. El control derivativo lleva el error de velocidad a cero. Este controlador limita que tan rápido puede cambiar el error. A su vez, normalmente se utiliza junto a un proporcional, de forma que la ecuación queda

$$u(t) = K_p e(t) + K_d \frac{de}{dt}$$

donde K_p es la ganancia proporcional, K_d es la ganancia derivativa y $e(t)$ es el error en el tiempo t. En la figura D.2 se observa el diagrama de bloques para un sistema controlado por un controlador PD. Este tipo de controladores tiene un

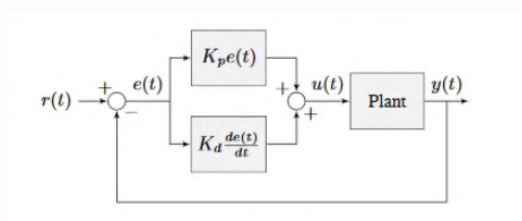


Figura D.2: Diagrama de bloques de un controlador proporcional derivativo.

controlador proporcional para la posición K_p y un controlador proporcional para la velocidad K_d.

D.1.3. Controlador integral

La salida del controlador integral está dada por el acumulo del área entre el punto de referencia y la salida durante un tiempo, o sea del error. A su vez, normalmente se utiliza junto a un proporcional, de esta forma la ecuación de este control compuesto es

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau$$

, donde K_p es la ganancia proporcional, K_i es la ganancia integral, $e(t)$ es el error en el tiempo t y τ es la variable de integración. La imagen D.3 muestra un diagrama de bloques para un sistema controlado por un controlador PI.

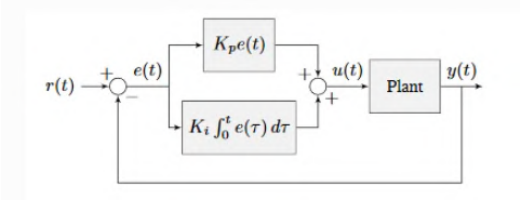


Figura D.3: Diagrama de bloques de un controlador proporcional integral.

Cuando el sistema está cerca del punto de referencia en estado estable, el término proporcional puede ser muy pequeño para llevar la salida hasta el punto

de referencia, y el término derivativo es cero. Esto puede resultar en un error de estado estable. Una manera usual para eliminar el error de estado estable es integrar el error y sumarlo a la entrada de control, esto aumenta el esfuerzo de control hasta que el sistema converja.

D.1.4. Controlador PID

Finalmente el controlador PID es usado comúnmente como un controlador de retroalimentación que consiste de un término proporcional, uno integral y uno derivativo, de ahí el nombre.

El término proporcional lleva el error de posición a cero, el término derivativo lleva el error de velocidad a cero y el término integral acumula el área entre el punto de referencia y la salida durante el tiempo y suma el resultado de la entrada de control.

La ecuación que rige este controlador es la siguiente

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de}{dt}$$

Donde K_p es la ganancia proporcional, K_i es la ganancia integral, K_i es la ganancia derivativa, $e(t)$ es el error en el tiempo t y τ es la variable de integración. En la figura D.4 se muestra un diagrama de bloques para un sistema controlado por un controlador PID.

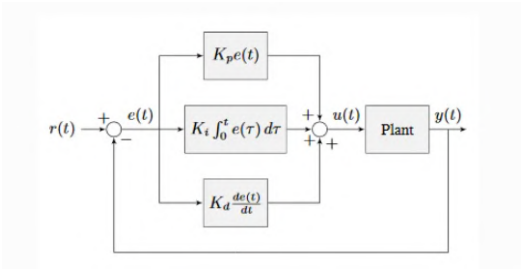


Figura D.4: Diagrama de bloques de un controlador PID

Un sistema manejado por un controlador PID generalmente tiene tres tipos de respuesta: subamortiguada, sobreamortiguada y críticamente amortiguada como se puede ver en la figura D.5. Donde la respuesta subamortiguada es una respuesta que presenta un sobretiro. La respuesta sobreamortiguada es la respuesta más lenta de todas para llegar al valor estable. Por último la respuesta críticamente amortiguada está entre ambas respuestas anteriores.

El objetivo consiste en tener una respuesta lo menos sobreamortiguada posible de manera que el sistema no oscile. Para esto se suelen sintonizar las variables de PID y así evitar que esto suceda. Una forma que se utiliza comúnmente para ajustar estas variables, es el método de Ziger-Nichols. Este sugiere parámetros en función de la respuesta al escalón experimental de la planta.

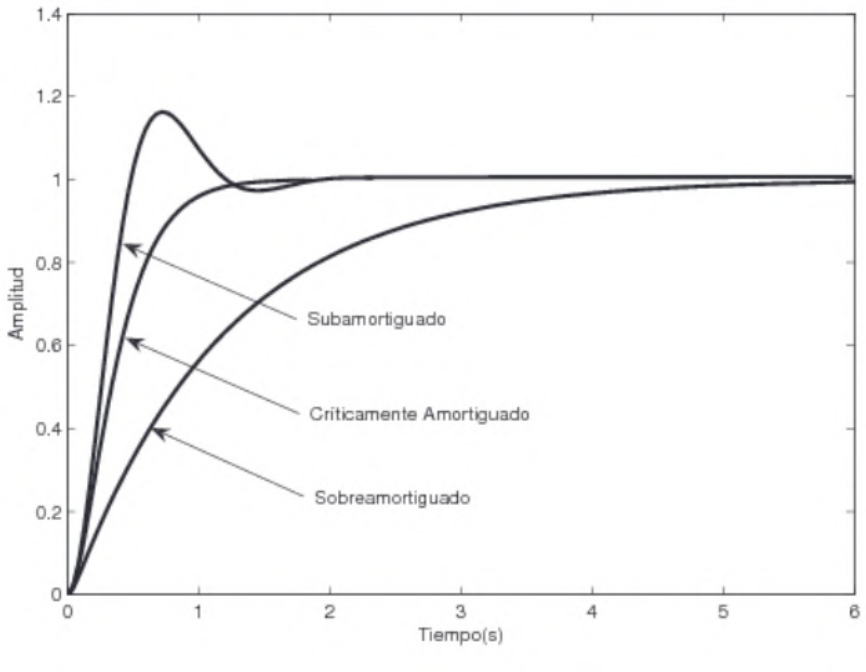


Figura D.5: Distintos tipos de respuesta escalón.

D.2. Método de Ziegler-Nichols

El método de Ziegler-Nichols permite ajustar o "sintonizar" un controlador PID de forma empírica, sin necesidad de conocer las ecuaciones de la planta o del sistema controlado. Estas reglas de ajuste propuestas por Ziegler y Nichols fueron publicadas en 1942 y desde entonces es uno de los métodos de sintonización más ampliamente difundido y utilizado. Los valores propuestos intentan conseguir en el sistema realimentado una respuesta al escalón con un sobrepulso máximo del 25 %, que es un valor robusto con buenas características de rapidez y estabilidad para la mayoría de los sistemas.

El método de sintonización de reguladores PID, permite definir las ganancias proporcional, integral y derivativa a partir de la respuesta del sistema en lazo abierto, o a partir de la respuesta del sistema en lazo cerrado. Cada uno de los dos ensayos se ajusta mejor a un tipo de sistema.

Debido a que el PID persiste dentro dentro del motor, no fue posible sacarlo de forma de sintonizarlo con la respuesta escalón. De esta manera se utilizó la sintonización en lazo cerrado.

Este método no requiere retirar el controlador PID del lazo cerrado. En este caso, solo hay que reducir al mínimo la acción derivativa y la acción integral del regulador PID. El ensayo en lazo cerrado consiste en aumentar poco a poco la ganancia proporcional hasta que el sistema oscile de forma constante ante cualquier perturbación. Esta oscilación debe ser lineal, sin saturaciones. En este momento hay que medir la ganancia proporcional, llamada ganancia crítica o K_c, y el periodo

Tabla D.1: Tabla de parametros de Ziegler - Nichols

Control	K_p	K_i	K_d
P	$0.50\ K_c$		
PI	$0.45\ K_c$	$0.54\ \frac{K_c}{T_c}$	
PD	$0.80\ K_c$		$0.075\ K_c T_c$
PID	$0.59\ K_c$	$1.18\ \frac{K_c}{T_c}$	$0.075\ K_c T_c$

de oscilación T_c en segundos. Una vez medidos estos dos valores, se pueden calcular los parámetros del controlador PID con acción solo proporcional (P), proporcional e integral (PI), proporcional y derivativa (PD) o proporcional integral y derivativa (PID). La constante K_p corresponde a la ganancia proporcional, K_i es la ganancia integral y K_d es la ganancia derivativa.

Esta página ha sido intencionalmente dejada en blanco.

Referencias

[1] D. López Boffa, P. Massonnier, and L. Sirio, "Picassobot," dec 2020.

[2] J. Arruti, M. Ottavianelli, and A. Solari, "Parra : Artífice de realizaciones robóticas artísticas," oct 2020.

[3] HRobotLab, "Página sobre Neo en la RoboCup." [Online; accessed 07.05.2022].

[4] Boston Dynamics, "Página oficial de Boston Dynamics sobre Spot." [Online; accessed 07.05.2022].

[5] Hanson Robotics, "Página oficial de Hanson Robotics sobre Sophia." [Online; accessed 07.05.2022].

[6] DrawBo, "Página de Medium de DrawBo." [Online; accessed 07.05.2022].

[7] landzo, "Página comercial de Quincy." [Online; accessed 07.05.2022].

[8] P. Tresset and F. Fol Leymarie, "Portrait drawing by Paul the robot," *Computers & Graphics*, vol. 37, p. 348–363, 08 2013.

[9] O. Deussen and T. Lindemeier, "E-david : Wissenschaftlicher versuch und malendes monstrum." Zufallszwänge - Roboterbilder zwischen Wissenschaft und Kunst, Konstanz, 2013.

[10] P. Tresset and O. Deussen, "Artistically skilled embodied agents," *AISB 2014 - 50th Annual Convention of the AISB*, 01 2014.

[11] SSH developer team, "Protocolo de comunciación ssh."

[12] OpenCV developer team, "Methods to read and write images."

[13] Robotis, *Manual electrónico de los motores AX-12A*.

[14] Robotis, *Manual electrónico de los motores MX-12W*.

[15] Robotis, "Protocolo de comunciación dynamixel 1.0."

[16] N. Ruiz, "Cánones de la figura humana." `https://www.dsigno.es/blog/diseno-de-moda/canones-de-la-figura-humana`.

[17] M. Gudino, "Comparacion de los modelos de arduino." [Online; accessed 08.05.2022].

[18] Raspberry PI, "Sistemas operativos de Raspberry PI," 2021. [Online; accessed 08.05.2022].

[19] A. A. Huertos, "Comparacion de los modelos de raspberry pi," 2021. [Online; accessed 08.05.2022].

[20] jumejume1, "AX-12A Servo Library." `https://github.com/jumejume1/AX-12A-servo-library`.

[21] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A Limited Memory Algorithm for Bound Constrained Optimization," *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[22] J. Morales and J. Nocedal, "Remark on "Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound Constrained Optimization"," *ACM Trans. Math. Softw.*, vol. 38, p. 7, 11 2011.

[23] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *The Computer Journal*, vol. 7, pp. 308–313, 01 1965.

[24] Wikipedia contributors, "Polygonal chain — Wikipedia, the free encyclopedia," 2021. [Online; accessed 2-May-2022].

[25] D. H. Douglas and T. K. Peucker, "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature.," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10 (2), pp. 112–122, 1973.

[26] "RDP in PyPI." `https://pypi.org/project/rdp/`.

[27] Wikipedia contributors, "Ramer–douglas–peucker algorithm — Wikipedia, the free encyclopedia," 2022. [Online; accessed 6-May-2022].

[28] R. Grompone von Gioi and G. Randall, "A Sub-Pixel Edge Detector: an Implementation of the Canny/Devernay Algorithm," *Image Processing On Line*, vol. 7, pp. 347–372, 2017. `https://doi.org/10.5201/ipol.2017.216`.

[29] mnicnc404, "CartoonGAN-TensorFlow2." `https://github.com/mnicnc404/CartoonGan-tensorflow`.

[30] J. Pinkney, "Toonify." `https://github.com/justinpinkney/toonify`.

[31] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2414–2423, 2016.

[32] X. Qin, Z. Zhang, C. Huang, M. Dehghan, O. Zaiane, and M. Jagersand, "U2-Net: Going Deeper with Nested U-Structure for Salient Object Detection," vol. 106, p. 107404, 2020.

[33] "Informativedrawings." https://huggingface.co/spaces/carolineec/informativedrawings/tree/main. Accessed: 2022-03-30.

[34] lanius, "tinyik." https://github.com/lanius/tinyik.

[35] P. Manceron, "IKPy." https://github.com/Phylliade/ikpy.

[36] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[37] J. Gettys, P. Karlton, and S. McGregor, "The X Window System, Version 11," *Software Practice and Experience*, 1990. http://www.hpl.hp.com/techreports/Compaq-DEC/CRL-90-8.pdf.

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

Esta página ha sido intencionalmente dejada en blanco.

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Esta es la última página.
Compilado el martes 4 octubre, 2022.
http://iie.fing.edu.uy/