UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Carrera de Ingeniería Electrónica

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE OPTIMIZACIÓN DE TRAYECTORIA EN EL ROBOT SAWYER PARA TAREAS DE CLASIFICACIÓN CON MONITOREO

Tesis para optar el título profesional de Ingeniero Electrónico

José Antonio Avalos Villa

Código 201410156

Asesor

Oscar E. Ramos

Lima - Perú

Febrero 2020

La tesis

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE OPTIMIZACIÓN DE TRAYECTORIA EN EL ROBOT SAWYER PARA TAREAS DE CLASIFICACIÓN CON MONITOREO

ha sido aprobada.

Arturo Rojas Moreno

Sergio Aranda Egúsquiza

Julien Noel

Dedicatoria:

A mi familia.

Agradecimientos:

A mi asesor Dr. Oscar E. Ramos por su continuo apoyo en mi formación y a los profesores del departamento de Ingeniería Electrónica, en especial, a Dr. Victor Murray y Dr. Ruth Canahuire por su apoyo incondicional en esta tesis.

A mis compañeros Sergio Cortez, Karina Vasquez y Claudia Gutierrez por sus comentarios enriquecedores.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

Pág.

ÍNDICE DE TABLAS

Pág.

RESUMEN

La evolución en la técnica de los procesos industriales ha exigido la mejora constante de las capacidades de la robótica como precisión, procesamiento u optimización de tiempo con el fin de extender sus aplicaciones. Para los robots manipuladores, la retroalimentación permite disminuir el error al identificar el estado de la posición inicial del objeto, de la posición final deseada y obstáculos en el área. Con dicha información, y empleando los controladores del fabricante, es posible ejecutar el movimiento del robot aplicando cinemática inversa. Sin embargo, dicho controlador no permite la optimizar factores como el tiempo de desplazamiento, la energía empleada, o la suavidad de la transición. Esto, principalmente, a la dificultad de generar un recorrido continuo sobre las infinitas soluciones de trayectorias, característico de los robots redundantes. En este sentido, la problemática se centra en obtener la trayectoria óptima en parámetros de tiempo y suavidad para un robot manipulador. Por ello, en esta tesis se propone el diseño e implementación de un sistema de control empleando una ecuación costo que permita obtener la trayectoria óptima continua con el menor tiempo y menores valores de sobre aceleración, con una ponderación entre ambas variables que el usuario definirá de acuerdo a la aplicación.

El sistema fue implementado en el robot Sawyer de *Rethink Robotics* para ejecutar una tarea de clasificación flexible con una interfaz de monitoreo online que emplea los servicios de nube y una retroalimentación por cámara que permite identificar la posición cartesiana del objeto de interés. El problema de optimización de trayectoria se resuelve empleando el método L-BFGS-B (Broyden–Fletcher–Goldfarb–Shanno) luego de transformar las restricciones mecánicas del robot a una única restricción, con el fin de reducir el costo computacional. Los resultados demostraron que el controlador propuesto puede ser escalado a distintos robots manipuladores permitiendo la mejora en la toma de decisiones, así como el incremento de seguridad al mejorar la estabilidad y el rendimiento de las aplicaciones con monitoreo en tiempo real.

PALABRAS CLAVES:
Control de trayectoria; Control óptimo; Robot Sawyer; Spline; Cinemática.

ABSTRACT

Evolution in industrial technique processes has required a constant improvement in robotic capabilities such as precision, processing or time optimization in order to extend its applications. For robot manipulators, feedback allows to reduced the error getting information about the initial positions of the object, the desired final position and obstacles in workspace. With this information, and using the manufacturer's controllers, it is possible to execute the movement of the robot by applying inverse kinematics. However, this controller does not allow the optimization of factors such as travel time, energy used, or smoothness of the transition. This is mainly due to the difficulty of generating a continuous path over the infinite trajectory solutions, characteristic of redundant robots. In this sense, the problem is focused on obtaining the optimal trajectory in time and smoothness parameters for a robot manipulator. Therefore, The purpose of this thesis is to design and implement a control system using a cost equation that allows obtaining the optimal continuous trajectory with the shortest time and minimum jerk allowing to weight both objectives according to the application.

The system was implemented in the Rethink Robotics Sawyer robot to execute a flexible classification task together with an online monitoring interface that uses cloud services and a camera feedback that allows to identify the working conditions in addition to the cartesian position of the object of interest. The path optimization problem is solved using the L-BFGS-B (Broyden-Fletcher-Goldfarb-Shanno) method after transforming the mechanical constraints of the robot to a single constraint, in order to reduce the computational cost. The results showed that the proposed controller can be scaled to different manipulative robots allowing the improvement in decision-making, as well as the increase in security by improving the stability and performance of applications with real-time monitoring.

KEYWORDS:
Trajectory control; Optimal control; Sawyer robot; Spline; Kinematic.

CAPÍTULO 1

INTRODUCCIÓN

En este primer capítulo se exponen los fundamentos que permiten dirigir el desarrollo de esta tesis. En ella se incluye la definición de la problemática, justificación, objetivo general y específico, y el alcance de la propuesta. Para cada uno de dichos puntos se expone el contexto y el sustento avalado por la bibliografía.

1.1 Descripción de la problemática

La robótica industrial [1] es pieza esencial en la automatización de procesos que busca englobar tareas como desplazamiento, clasificación, ensamblaje y demás operaciones relacionadas al manejo de materiales empleando, principalmente, robots manipuladores. Entre sus aplicaciones se encuentran, generalmente, tareas repetitivas con condiciones predefinidas de trabajo que buscan garantizar la efectividad de la ejecución de la tarea bajo condiciones mecánicas controladas con un alto costo de implementación. A pesar de ello, este proceso no logra eliminar los niveles de error debido a la incertidumbre en las operaciones mecánicas u otros factores humanos en su uso.

Por ello, los sistemas [2] de mayor complejidad buscan disminuir dichos factores con un modelos de retroalimentación empleando sensores discretos o cámaras externas que le permiten adquirir información para interpretar con mayor precisión su entorno a fin de verificar y adaptar el movimiento del robot a las condiciones verdaderas de trabajo en tiempo real. Con dicha información, al ejecutar el desplazamiento, se emplea fundamentalmente los controladores del fabricante que se enfocan principalmente en garantizar que los valores de ejecución no superen las restricciones mecánicas del robot para evitar

algún daño permanente en los motores o *hardware* del equipo. Dichos movimientos generados también se caracterizan por gran número de incremento y decremento repentino en los valores de aceleración, denominados picos, que genera un daño parcial en la vida útil de los motores, además de valores de inestabilidad que afecta el objeto a desplazar en el caso de robot manipuladores. Esto se debe a que dicho controlador no tiene como requerimiento la necesidad de optimizar factores como el tiempo de desplazamiento, la energía empleada, o la suavidad de la transición. Dichos factores son de vital importancia en las tareas industriales donde el tiempo de ejecución y el mantenimiento del robot representa altos costos para el usuario debido a su continuo uso.

La dificultad de incorporar dichos factores al controlador del fabricante, que generalmente se basa o es un controlador por posición, se debe principalmente a la dificultad de generar un recorrido continuo debido a las infinitas soluciones de trayectorias posible producto de los grados de libertad en el robot manipulador conocido también como robot redundante. En este sentido, la problemática a resolver se enfoca en cómo generar una trayectoria que optimice los parámetros de tiempo y suavidad para un robot manipulador redundante dirigido a una tarea de clasificación empleando retroalimentación por cámara.

1.2 Justificación de la problemática

La sobreaceleración es la primera derivada de la aceleración, o la tercera derivada de la posición respecto al tiempo. La trayectoria de un robot manipulador con sobreaceleración mínima no tendrá cambios o discontinuidades rápidas en la aceleración aplicada a través de los actuadores, y/o la aceleración de sus efectores finales afectando en menor medida la continuidad del movimiento. Es decir, disminuir los valores de sobreaceleración evidencia una menor vibración en el desplazamiento del robot al disminuir excesivos picos en la aceleración. Sin embargo, el lograr valores mínimos implica aumentar el tiempo de ejecución, que es un factor de gran criticidad donde la relación entre ambos es inversa.

En este sentido, la solución a una trayectoria óptima se obtiene luego de resolver la minimización de una ecuación de costo planteada que considere ambos factores: el tiempo y la sobreaceleración en la ejecución. Múltiples trabajos en antecedentes [3, 4] proponen definir parámetros que permitan al usuario ejercer una ponderación para cada componente de la ecuación o se han enfocado en generar una única solución [5–7] resultando ambos casos poco atractivos para la industria que busca una fácil interpretación y flexibilidad entre la intensidad de vibración y el tiempo de ejecución.

Por dicho motivo, la optimización de trayectoria propuesta busca encontrar los valores óptimos mediante una ecuación de costo que permita emplear un único factor de ponderación entre ambos factores, permitiendo al usuario elegir un tiempo adecuado de acuerdo a su aplicación, con las ventajas de una sobreaceleración reducida para cada caso. Para demostrar su funcionalidad, se define una tarea de clasificación flexible que permita emplear de forma continua el controlador junto a un sistema de monitoreo como modelo de tarea autónoma. El controlador propuesto se comparará con el controlador del fabricante a fin de demostrar las ventajas en dicha tarea, además de otras ecuaciones de costo propuestas en la bibliografía.

1.3 Objetivos

1.3.1 Objetivo general

El objetivo general de este trabajo es diseñar e implementar un sistema que permita al robot manipulador realizar tareas de clasificación flexible optimizando los valores de tiempo y sobreaceleración con monitoreo remoto.

1.3.2 Objetivos específico

Se plantean tres objetivos específicos:

a) Diseñar e implementar la optimización de trayectoria para generar la secuencia de tiempo de valores para la posición, velocidad, y aceleración a partir de la ruta geométrica de las posiciones de las articulaciones del robot.

b) Diseñar e implementar un algoritmo que permita identificar la posición cartesiana de los objetos a clasificar, a fin de generar la ruta geométrica.

c) Diseñar e implementar un sistema de monitoreo remoto que permita conocer el estado de las articulaciones del robot y de las condiciones de clasificación sobre los valores generados.

1.4 Alcances y limitaciones

El espacio de trabajo se encuentra limitado a las dimensiones físicas del robot, y no debe ser excedido debido a que son límites de seguridad establecidos por el fabricante en valores de velocidad, aceleración y sobreaceleración. El algoritmo de optimización propuesto se enfoca en los factores de tiempo y sobreaceleración excluyendo cualquier otra variable. Se empleará un factor que ponderará cada uno de los término de la optimización de acuerdo a las necesidades de la aplicación, con un rango de 10 %-90 % o proporcional entre $< 0.1 - 0.9 >$. La información empleada en el monitoreo se restringe a la obtenida a través del entorno de trabajo del robot y se emplea un servicio de nube para su acceso. La tarea propuesta la clasificación de recipientes en proporción a las dimensiones de la garra (traducción de *gripper*) del robot del manipulador.

CAPÍTULO 2

MARCO TEÓRICO

En este capítulo se busca explicar los conceptos mínimos necesarios para el diseño del sistema propuesto y compararlo con métodos aplicados en trabajos relacionados que nos permiten identificar el vocabulario y el funcionamiento para sustentar la metodología que se planteará en esta tesis. Este capítulo, además, se exponen los antecedentes organizado en base a cada objetivo planteado en 1.3.2.

2.1 Antecedentes

Esta sección presenta los trabajos relacionados a los tres objetivos particulares de esta tesis. En la sección 2.1.1 se explica el desarrollo de los algoritmos de optimización de trayectoria. En la sección 2.1.2 se introduce los métodos para la introducción de los modos de retroalimentación empleando cámaras. Por último, en la sección 2.1.3 se explican los procesos necesarios para ejecutar el monitoreo a distancia.

2.1.1 Modelos de optimización de trayectoria

Para cada configuración del robot, el conjunto de valores en cada articulación del robot determina la posición y orientación del objeto. Entre los modos de control, en [8] se plantea emplear valores de torque como entrada para cada articulación del robot obtenidas a partir de las coordenadas cartesianas empleando ecuaciones dinámicas no lineales. Sin embargo, el mayor inconveniente radica en el pequeño período de control que limita los cálculos que se pueden realizar entre cada muestra. Como solución, las coordenadas cartesianas transformadas a coordenadas articulares por medio de cinemática inversa muestran

mayor viabilidad de implementación al reducir considerablemente el costo en términos de complejidad computacional. Cabe resaltar que la principal desventaja de emplear cinemática consiste en que no se puede controlar de manera directa la fuerza o torque aplicados con el efector final, pero dado que el enfoque de esta tesis no se centra en una aplicación que requiera control de fuerza, esta desventaja no presenta una real limitación en este caso. Para los robots redundantes [9], definido como robots con más de 6 grados de libertad, aplicar la cinemática inversa genera infinitas soluciones, por lo cual la solución se determina en base a una serie de criterios particulares del objetivo de la aplicación.

En este sentido, para ejecutar el movimiento de un robot manipulador redundante, se define una serie de puntos de trayectoria en términos de la orientación y posición cartesiana de la articulación final que contiene la estructura para manipular el objeto de interés. Esta serie de puntos se transforman en un conjunto de ángulos de articulación (llamados también *knots*) utilizando cinemática inversa donde la solución única se genera teniendo como referencia la posición anterior y el menor recorrido. Sin embargo, para construir la trayectoria es necesario determinar los valores intermedios; es decir, emplear los *knots* e interpolarlos para enviar los valores correspondiente a la frecuencia de control del robot. En consecuencia, existen múltiples métodos para la interpolación; no obstante, la optimización de trayectoria [6, 10–14] se basa en garantizar un movimiento continuo y suave; donde el método que nos permite controlar dichos parámetros es el *spline* [15].

En la literatura, se exponen múltiples tipos de *splines* para la generación de trayectoria: *spline* polinomiales, trigonométrico [10], cuántico [6, 16], B-spline [12] y *spline* cúbico [17]. El *spline* cúbico es el más empleado ya que proporciona continuidad en velocidad y aceleración en cada knot a la par de requerir bajo costo computacional. En [18] se menciona que aunque las curvas de *spline* cúbicas no proporcionan continuidad en la sobreaceleración (puesto que la tercera derivada sería una constante), esta característica produce limitaciones de sobreaceleración que permite importantes reducciones en la vibración. Por tanto, minimizar la sobreaceleración disminuye los errores de posición

de la articulación, causa menos vibraciones, limita el desgaste excesivo del robot y evita grandes oscilaciones que pueden ocurrir empleando polinomios de orden superior. Sin embargo, el primer y último *knot* pueden deteriorar la trayectoria del manipulador en caso no cumplan con iniciar o concluir los valores de velocidad, aceleración y sobreaceleración en 0, ya que representaría una variación abrupta ejecutada por el controlador del robot; por lo cual resulta necesario la introducción de ecuaciones de un orden mayor al menos para esos casos. Por ello, se han realizado varios estudios que se enfocan en técnicas para generar trayectorias suaves y optimizadas por distintos métodos.

En [17] se propone trayectorias polinomiales cúbicas formuladas y optimizadas por el método de poliedro. Por otro lado, en [19] se emplean los *splines* cúbicos para generar una trayectoria conjunta mediante la interpolación de posiciones intermedias y velocidades con una optimización analítica. De la misma manera, en [20] desarrolla un método para optimizar las trayectorias de spline cúbicas del espacio de tareas utilizando técnicas de optimización convexa. Para [18], la planificación de la trayectoria se forma a partir de un *spline* cúbico buscando la reducción de la vibración en las articulaciones del robot. Asimismo, en [4] se ofrece un modelo de splines introduciendo modelos polinomiales de quinto orden mientras que en [11], se enfoca en la planificación global de trayectorias mínimas utilizando un análisis de intervalos. En [13] se presenta un algoritmo de planificación de trayectoria de costo mínimo para manipuladores robóticos utilizando el método de Programación Cuadrática Secuencial (SQP del inglés *Sequential quadratic programming*). En [21] propusieron una técnica para planificar óptimamente el tiempo de las trayectorias de los robots utilizando técnicas de SQP para obtener la trayectoria óptima mientras que en [3] se reformula y optimiza empleando un polinomio de quinto orden. En [22] se desarrolla un marco general para sintetizar *splines* polinomiales óptimos para sistemas de movimiento rígido utilizando un marco de programación convexo empleando el método SQP para resolver el problema de optimización de trayectoria no lineal restringida.

La optimización implica analizar el comportamiento completo del sistema robótico a nivel tanto cinemático como dinámico. Esto conlleva a determinar que el objetivo se basa en optimizar algunos de los parámetros de trabajo mediante funciones de costo apropiadas. Los criterios de optimización más utilizados se pueden clasificar de la siguiente manera [5]:

- Minimización del **tiempo**, que está limitado a la productividad y donde la función de costo busca un mínimo global.
- Minimización de la **sobreaceleración**, que está limitada a la calidad del trabajo, la precisión y el mantenimiento del equipo.
- Minimización de la **energía consumida** o esfuerzo del actuador, ambos vinculados a los ahorros.

Los modelos expuestos han utilizado algoritmos clásicos de optimización numérica basados en la linealización sucesiva, utilizando la primera y la segunda derivada de las funciones en la planificación de trayectoria del manipulador con la introducción de criterios mixtos. Los mismos se han centrado en caso particulares como minimización híbrida (tiempo-sobreaceleración) como también en modelos basados en energía donde el método de optimización ha mostrado mejores resultados empleando el método SQP.

2.1.2 Modelos de retroalimentación por cámara

La capacidad de detectar e identificar obstáculos móviles o fijos representa un reto permanente en el desarrollo de actividades con robots ya que no existe un modelo único sino que debe de ser adaptado a las condiciones de la tarea. En los últimos años se han desarrollado distintos métodos y técnicas para resolver este problema. Las contribuciones más comunes se basan en datos de imágenes estéreo [23] o combinación de imágenes

con escaneo 3D [24]. Aunque estas contribuciones otorgan a los sistemas un cierto nivel de autonomía, la detección de objetos complejos en tiempo real utilizando sensores de bajo costo, compactos y fáciles de implementar sigue siendo un objetivo importante de investigación.

En el caso de aplicarlo en un robot manipular redundante, se emplea una cámara externa que evitará que el robot únicamente repita la tarea en cada momento de clasificación. La segmentación de la imagen dependerá de los objetos a clasificar por el robot, donde se busca emplear casos prácticos al clasificar respecto a la superficie del objeto realizando una inspección 2D.

En [25] se describe una técnica directa para rastrear la mano humana basada en imágenes adquiridas por un sistema de cámara estéreo activo, en un modelo de interacción hombre-máquina: al detectar la posición de la mano, el robot puede interpretar un gesto de dirección humana empleando los dedos como la especificación de un objetivo a clasificar. Otro modelo en [26], esta compuesto por secuencia de tres procesos. El primero es una segmentación rápida en el espacio de color HSV de un parche plano ubicado en el efector final, es decir, la segmentación mediante la clasificación de píxeles y la distancia euclidiana como una medida de similitud; la segunda etapa consiste en la selección de una región de interés, la extracción de puntos de características y el seguimiento en esa región; y finalmente, en el tercer proceso, los puntos de característica se pueden usar para estimar la homografía entre el marco de referencia mundial y el marco de imagen. La metodología propuesta se ejecuta en tiempo real y es aplicable para el control visual y la compresión. Otro modelo en [27], plantea una técnica para el control de retroalimentación y guía de sistemas robóticos: la información recopilada del entorno es empleada en una tarea de supervisión visual que garantiza la correcta clasificación de los objetos utilizando la interacción de dos cámaras. El control se cambia de la cámara maestra, observando el espacio de trabajo, a la cámara de la articulación extrema que ve el objetivo donde el esquema de control se valida con estudios de simulación y experimentación con un robot

ABB y MATLAB. Finalmente, en [28] se emplea un robot de la misma familia del robot Sawyer, el robot Baxter, para tareas de clasificación, que consiste en utilizar retroalimentación visual para evitar errores mecánicos en la clasificación de un grupo de objetos en función de su color y forma. Dicho proceso incluye procesamiento de imágenes, cinemática inversa y un algoritmo de automatización que permite que la tarea sea definida por el usuario o por un objetivo específico. Con esta información se inspira a desarrollar el proceso en complemento con la segmentación en el espacio HSV.

2.1.3 Modelos de monitoreo con servicio de nube

La integración de los robots industriales con el internet es una tecnología emergente conocida como *cloud robotic* [29, 30] o *robótica con servicio de nube*. El desarrollo de esta tecnología se inició con la incorporación del internet de las cosas (conocido como *IoT* que deriva de *Internet of Things*) a través de las ventajas que ofrece y la rápida adaptación en el mercado dirigido a las aplicaciones que empleen robots principalmente del rubro industrial. La integración del *IoT* resulta crucial para la virtualización, control y monitoreo a distancia del robot mediante internet.

ROS (*Robot Operating System* o Sistema Operativo Robótico) [31] es el principal framework empleado para el desarrollo de software/aplicaciones para robots que gestiona el funcionamiento total de su sistema (mayor detalle en la sección 2.4) a la vez de ser una de las principales herramientas en la investigación en el campo de la robótica. A fin de extraer información para lograr una actividad de monitoreo, resulta necesario en primera instancia extraerla desde ROS. A nivel de conexión, ROS permite controlar un robot desde una estación de trabajo utilizando un solo maestro ROS empleando una conexión directa; sin embargo, esta solución no es escalable y está limitada al uso de una red de área local. El escenario típico es que cada robot inicie su propio nodo ROS Master, y los usuarios puedan controlar el robot desde sus estaciones de trabajo para usar el mismo ROS Master ejecutándose en el robot. Este enfoque estándar no permite monitorear el robot de forma

nativa a través de internet ya que los robots generalmente no tienen una dirección IP pública ya sea por cuestiones de infraestructura o seguridad.

Distintos métodos han sido propuestos a fin de superar la limitante del área local. En [32], se propone la creación de un protocolo de comunicación ligero denominado ROSLink, inspirado en MAVLink, para permitir una interacción basada en la nube entre los robots ROS y sus usuarios. La idea básica surgía de agregar un puente ROSLink encima de ROS para cada robot, de manera que este puente de comunicación permita el control del estado del robot mediante mensajes serializados JSON[33] (*JavaScript Object Notation* o *Notación de objeto basado en JavaScript*). ROSLink es un nodo ROS que accede a todos los temas y servicios de interés en ROS, y envía información seleccionada en mensajes ROSLink, serializados en formato JSON. Para nuestro caso particular, no es de interés ejercer el control de los nodos del robot sino el de poder monitorear las condiciones del robot por lo que podemos reducir el empleo de recursos necesarios. Por ello, se construyen un mensaje que resuma los parámetros de ROS para ejecutar una acción. Las buenas prácticas de dicha propuesta sugieren que los mensajes se representen como cadenas JSON. Dicho formato se emplea en el intercambio de datos porque es independiente de la plataforma, y el formato de representación de datos es independiente del idioma.

En [34], se presentó el planteamiento y ejecución de una plataforma robótica móvil y su integración en un entorno de servicio de nube con ROS. En ella se construyó una plataforma robótica móvil con diferentes sensores que incluía una cámara de profundidad, integrada con *Arduino* y *Raspberry Pi* como medio para conectar los sensores, el sistema de accionamiento y el procesamiento local de señales a bordo y con capacidad de comunicación inalámbrica para la transmisión y recepción de datos. Esta propuesta es una demostración de la posibilidad de interactuar diferentes tipos de hardware por medio de la convergencia de información empleando un único formato de medio compatible, donde su plataforma empleaba un servidor local para acceder en cualquier punto de la red. Dicho modelo demuestra el escalamiento al emplear distintos generadores de información y de

arquitectura de transmisión. En nuestro modelo, se deriva esta actividad a servicios de nube del mercado. De los trabajos mencionados, a continuación se menciona los conceptos que definirán nuestra metodología.

2.2 Conceptos básicos de Robótica

La introducción de los robots en la industria [35] se originó a mediados de 1960 por la necesidad de dos tecnologías: (1) control mecánico con alta precisión para la fabricación y (2) teleoperadores para el manejo remoto de materiales peligroso. Desde entonces, su impacto ha permitido la disminución de costo y la optimización en la producción. No limitándose a ello, su alcance se ha expandido a áreas como: biomecánica, neurociencias, simulación virtual, animación, cirugía, sensores, entre otras. En contraparte, los desafíos de las nuevas áreas emergentes están demostrando ser una fuente abundante de estimulación que alientan un mercado cada vez más competitivo.

2.2.1 Cinemática

Los robots o mecanismos robóticos son sistemas de cuerpos rígidos conectados por articulaciones. La Federación internacional de Robótica (IFR), basado en la norma ISO 8373:2012 [36], define al robot industrial como un robot manipulador multiusos, reprogramable que puede fijarse en su lugar o ser móvil para su uso en aplicaciones de automatización industrial. Una de las autoridades de este campo [9] expone distintas topologías para representar dichas conexiones, siendo dos modelos los principales: cadenas en serie y mecanismos paralelos. Las cadenas en serie son un sistema de cuerpos rígidos en el que cada miembro está conectado a otros dos, a excepción de los miembros primero y último miembro. Por otro lado, un mecanismo paralelo es un sistema en el que hay dos miembros conectados entre sí por múltiples cadenas de otros miembros y articulaciones.

La mayoría de robots industriales están formados por cadenas cinemáticas en serie, donde la cantidad de articulaciones representa los grados de libertad.

La cinemática de un robot describe la posición, velocidad, aceleración y todas las derivadas de orden superior que componen el mecanismo del robot en relación a otro cuerpo, sin considerar las fuerzas o torques ejercidos por los actuadores. La cinemática permite definir el espacio de trabajo de un manipulador robótico que es el volumen total barrido por el efector final a medida que el manipulador ejecuta todos los movimientos posibles. Este espacio de encuentra restringido a la geometría del manipulador y a los límites de los movimientos de las articulaciones.

La redundancia cinemática se produce cuando un manipulador robótico tiene más grados de libertad que los estrictamente necesarios para ejecutar una tarea determinada. Para una tarea espacial completamente general, que consiste en seguir una trayectoria de movimiento se requiere únicamente seis grados de libertad. En este sentido, el brazo de robot con siete o más articulaciones se considera un manipulador inherentemente redundante. Algunos ejemplos populares en el mercado se muestran en la Fig. 2.1. Sin embargo, incluso los brazos robóticos con menos grados de libertad, como los manipuladores industriales convencionales de seis articulaciones, pueden volverse cinemáticamente redundantes para tareas específicas, como el posicionamiento simple del efector final sin restricciones en la orientación.

El robot Sawyer[37] fabricado por Rethink Robotics y mostrado en la Fig. 2.2, cuenta con 7 grados de libertad se encuentra dirigido a la investigación al brindar las herramientas para modificar su control articular en posición, velocidad, torque o trayectoria, en comparación con fabricantes como KUKA Robotics (serie KR), ABB Robotics (serie IRB) entre otros.

Sawyer [37] se compone de siete articulaciones etiquetadas como J_0-J_6, desde su base hasta su articulación extrema, que permiten controlar la configuración mecánica del

(a) (b)

Figura 2.1: Modelo de robot redundantes. (a) Modelo Yumi de ABB Robotics [38]. (b) Modelo Titan de Kuka Robotics [39].

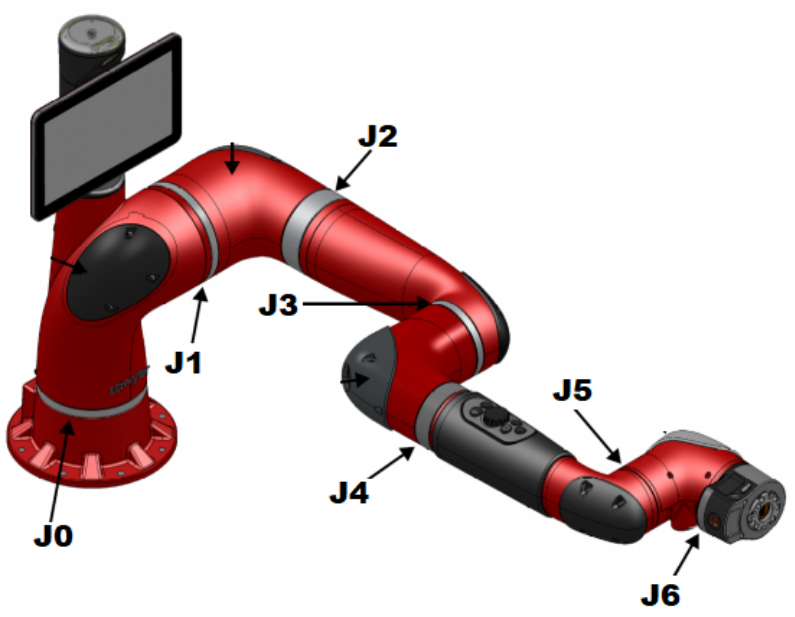


Figura 2.2: Definición de las articulaciones del Robot Sawyer [37].

robot. Los valores máximos que puede alcanzar cada articulación se limitan por el fabricante y se muestran en la Tabla 2.1 para la posición articular, la velocidad y la aceleración.

Tabla 2.1: Limites articulares para el robot Sawyer.[37]

Articulación	**Límite superior [rad]**	**Límite inferior [rad]**	**Máx. Velocidad (rad/s)**	**Máx. ACC (rad/s^2)**
J_0	3.0503	-3.0503	1.74	10.0
J_1	2.2736	-3.8095	1.328	8.0
J_2	3.0426	-3.0426	1.957	10.0
J_3	3.0439	-3.0439	1.957	10.0
J_4	2.9761	-2.9761	3.485	12.0
J_5	2.9761	-2.9761	3.485	12.0
J_6	3.14	-3.14	4.545	12.0

El problema de la cinemática directa para un manipulador busca determinar la posición cartesiana y orientación del efector final con respecto a la base dadas las posiciones de todas las articulaciones y sus parámetros geométricos. El problema de la cinemática directa es fundamental para desarrollar algoritmos de coordinación en los manipuladores, ya que las posiciones de las articulaciones se miden normalmente con sensores montados en los motores y es necesario calcular las posiciones de las articulaciones a fin de desplazar el efecto final al punto de referencia.

La cinemática inversa buscar resolver el problema opuesto al descrito. En un manipulador involucra encontrar los valores de las posiciones de las articulaciones dada una postura del efector final con respecto a la base y los valores de todos los parámetros geométricos. De acuerdo a los grados de libertad y a la geometría de cada robot, las condiciones para resolver dicha función inversa varían. Los métodos usados pueden dividirse en analíticos y numéricos. Los métodos analíticos encuentran la solución de manera cerrada para robots con un máximo de 6 grados de libertad por cadena serial, pero suelen ser complicados de obtener. Los métodos numéricos son una alternativa algorítmica genérica que se basa en iteraciones hasta la convergencia, y brindan diferentes soluciones dependiendo del punto inicial que se considere. Para el caso de robots redundantes, dado que existe un número infinito de soluciones a la cinemática inversa, solamente los métodos numéricos pueden ser utilizados complementado con las particularidades de la aplicación.

2.3 Control por trayectoria

Una tarea fundamental en robótica es la generación de movimientos para cuerpos complejos de un punto inicial a un punto deseado; este estudio es conocido como *motion planning* o planeamiento de la ruta geométrica. Los algoritmos de planeamiento de ruta emplean la cinemática inversa para generar las configuraciones necesarias del robot para un conjunto de posturas deseadas o *knots*. Con dicha información, se puede ejecutar el movimiento del robot por control de posición (conocido también como control por cinemática directa) que se basa en los parámetros del fabricante.

El control por trayectoria, a diferencia del control de posición, permite controlar un vector de posiciones para las articulaciones en correspondencia con un vector de tiempo continuo. Para ello, este control necesita como entrada las posiciones, velocidades y aceleración de cada una de las articulaciones. La propuesta de esta tesis implica que el robot desarrolle una trayectoria en base a información cartesiana de la articulación final entregada por una cámara externa.

2.4 Robot Operating System (ROS)

Para un robot manipulador de investigación, existen dos principales interfaces de programación; la primera, se basa en emplear el software del fabricante, el cual permite desarrollar las actividades predefinidas y modificadas de acuerdo a las condiciones particulares de la actividad; la segunda se basa en usar frameworks alternativos como *Robot Operating System* (ROS), el cual permite la extracción de la información de sus componentes y la escritura de los mismos. Esta sección se centrará en explicar el segundo método y su filosofía de programación.

2.4.1 Comunicación ROS

La red de comunicación de los procesos definidos en ROS se basa en procesar datos distribuidos por canales que son definidos por el fabricante del robot. Los conceptos básicos de ROS, representados en la Fig.2.3, se encuentran organizados como:

1. **Nodos**: Son procesos donde se realizan cálculos. El sistema de control de un robot se encuentra en un conjunto de nodos. Un nodo es declarado en un entorno y puede ser definido utilizando diversos lenguajes de programación, siendo C++ y Python los lenguajes por defecto.

2. **Mensajes**: La comunicación entre los nodos incluye mensajes. Un mensaje se define como una estructura de datos con un formato predefinido.

3. **Tópico**: Los mensajes se enrutan a través de un sistema de transporte con organización de publicación / suscripción. Un nodo envía un mensaje publicándolo en un determinado tópico el cual recibe un nombre que se emplea para identificar el contenido del mensaje.

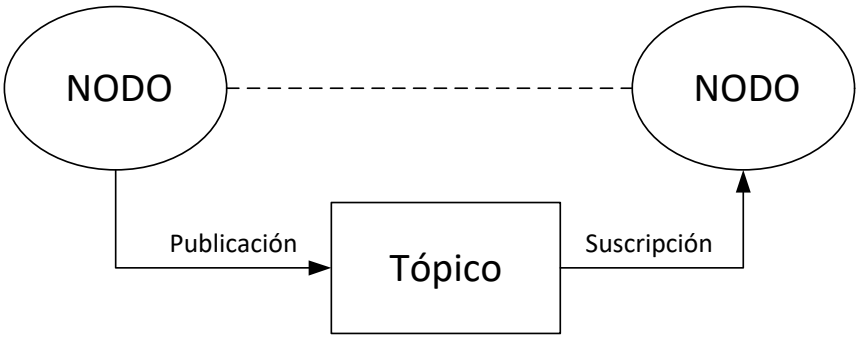


Figura 2.3: Modo resumido de operación en ROS.

ROS es empleado en diferentes robots industriales en el entorno Linux y sus versiones recomendadas varían en periodos de dos años. Una extensa bibliografía sobre su funcionamiento y programación se encuentra en [31, 40].

2.5 Retroalimentación por cámara

Existen distintas tecnologías que comprenden la computación visual que proporcionan un soporte valioso para la validación del proceso. La computación visual puede definirse como la totalidad del proceso de adquisición, análisis y síntesis de datos visuales por medio de computadoras. Entre sus aplicaciones se encuentra:

- Adaptación automática y flexible de la cadena de producción a los requisitos cambiantes. Es decir, permite cambiar su configuración de acuerdo a las condiciones en tiempo real.
- Seguimiento de piezas y productos y su comunicación con máquinas y otros productos.
- Mejoramiento de interacción hombre-máquina (HMI del término inglés *Human-Machine-Interface*), incluida la coexistencia con robots o formas radicalmente nuevas de interactuar y operar en fábricas. Es conocida también como *coworking*.
- Optimización de la producción por medio de empleo de la comunicación habilitada para Internet de las cosas (IoT) en fábricas inteligentes.
- Servicios y modelos comerciales que contribuyen a cambiar las formas de interacción en la cadena de valor.

OpenCV (de *Open Source Computer Vision*) [41] es una librería de software libre para visión artificial que más empleada en esta área.

2.5.1 Segmentación basada en HSV

Una representación alternativa del modelo de color RGB es el modelo HSV. La representación de HSV (*Hue-Saturation-Value*) modela la forma en que se mezclan las

pinturas de diferentes colores, con la dimensión de saturación que se asemeja a varios tonos de pintura de colores brillantes, y la dimensión de valor que se asemeja a la mezcla de esas pinturas con cantidades variables en valores blanco o negro. Es más fácil representar un color en el plano HSV que en el espacio de color RGB.

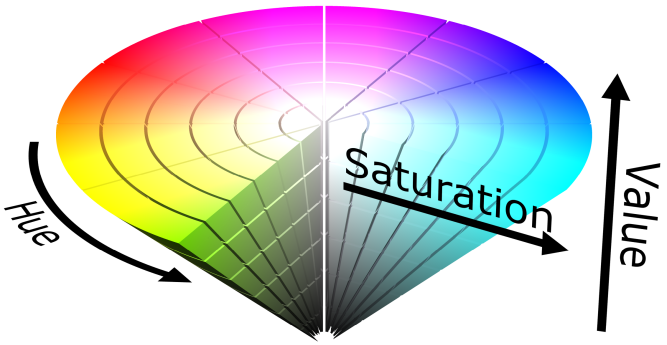


Figura 2.4: Representación RGB en HSV.

En caso el usuario defina ciertos rangos empleando HSV es posible identificar secciones de la imagen digital de interés, es decir, se segmenta la imagen a partir de una entrada de color determinada por el usuario.

2.5.2 Transformada de Hough para círculos

La transformada de Hough es una técnica básica utilizada en el procesamiento de imágenes digitales para detectar objetos circulares en una imagen digital. El propósito de la técnica es encontrar círculos en entradas de imagen imperfectas a partir de la descripción:

$$(x-a)^2 + (y-b)^2 = r^2,$$

donde (a, b) es el centro del círculo, y r es el radio. Si un punto 2D (x, y) es fijo, entonces los parámetros se pueden encontrar de acuerdo con dicha ecuación. Las imágenes digitales son obtenidas en formato 2D y transformadas por medio de librerías como por ejemplo *OpenCV*. Esta transforma es potente debido a los parámetros que considera para la identificación en tiempo real.

Figura 2.5: Ejemplo de aplicación de la transformada de Hough. En rojo se determina el circulo de la imagen a partir de la configuración empleando *OpenCV*.

2.6 Monitoreo en robótica

Los robots tienen la capacidad de procesar información utilizando múltiples procesadores integrados. Dichos robots pueden aprovechar el procesamiento remoto al descargar tareas de computación pesadas a computadoras externas. El servicio de nube o *cloud* permite la transmisión de información en tiempo real como tarea de monitoreo que incluye la creación de base de datos así como interfaces interés de un proceso.

2.6.1 Servicios de nube

La automatización tradicional implica mecanismos con tareas repetitivas de alta eficiencia y precisión sin la necesidad de cálculos significativos. Estas soluciones solo son rentables a gran escala donde se fabrican productos similares durante largos períodos de tiempo. Sin embargo, el futuro de la automatización radica en los pisos de fábrica flexibles y en los procesos de fabricación rápida, que pueden proporcionar productos complejos de ciclo de vida corta rápidamente [29]. Esto permitirá que los tipos de productos y suministros se actualicen con frecuencia, y que los procesos de fábrica se re-configuren mediante la interacción con operadores humanos. Esto requiere un nuevo nivel de adaptabilidad de la infraestructura de automatización, eficacia en configuraciones menos estructuradas y una coexistencia segura con las personas.

En este sentido la introducción de la nube se puede aprovechar en el cómputo paralelo bajo demanda para abordar rápidamente las necesidades de las operaciones de robot exigentes desde el punto de vista informático, como la planificación del movimiento. La nube puede incluso actuar como intermediario entre la instalación de automatización y los operadores humanos y puede acceder fácilmente a una vista global de las operaciones industriales. La nube también puede acomodar la interacción entre múltiples robots, dándoles la capacidad de compartir el conocimiento y aprender colectivamente de la experiencia en el tiempo. Los proveedores de servicios en la nube de dominio público son *Amazon Web Services*, *Google Cloud*, *IBM Cloud* y *Microsoft Azure* .

2.6.2 Métodos de comunicación

Estos mensajes se construyen en base a los parámetros de los tópicos en ROS para obtener datos o enviar datos en una sola acción. Los mensajes se representan como un conjunto de información de formato JSON. Dicho formato se optó por utilizar en el intercambio de datos porque es independiente de la plataforma y el formato de representación de datos es independiente del lenguaje [33]. MQTT (inicial de *Message Queuing Telemetry Transport*) [42] es el protocolo primario que utilizan los dispositivos y las aplicaciones para comunicarse con la plataforma *IoT Platform*. MQTT es un protocolo de transporte de mensajería de publicación y suscripción diseñado para el intercambio eficiente de datos en tiempo real entre el sensor y los dispositivos móviles. MQTT se ejecuta sobre TCP/IP, y mientras sea posible codificar directamente a TCP/IP, también se puede elegir utilizar una biblioteca que maneje los detalles del protocolo MQTT.

CAPÍTULO 3

METODOLOGÍA

Este capítulo se basa en describir la metodología para lograr los tres objetivos particulares en orden del flujo de información: en la sección 3.1, se describe el modo de retroalimentación empleando una cámara para identificar el objeto a clasificar. En la sección 3.2 se explica el desarrollo de los algoritmos de optimización de trayectoria. En la sección 3.3 se explican los procesos necesarios para ejecutar el monitoreo a distancia. Finalmente, en la sección 3.4 se describirá el proceso que engloba el funcionamiento de todos los objetivos particulares a fin de lograr el objetivo general.

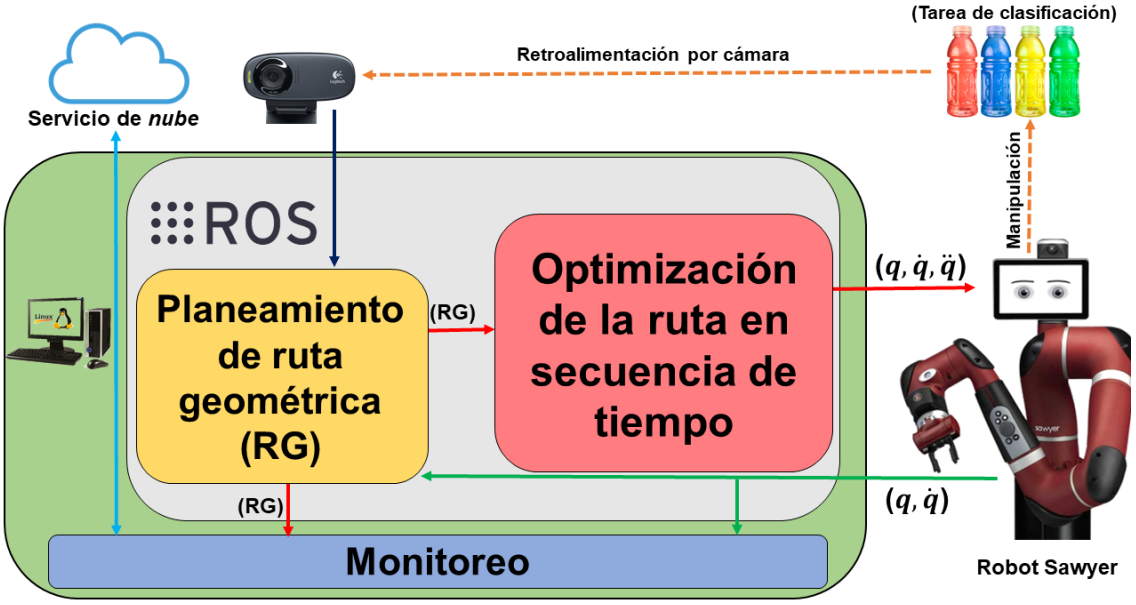


Figura 3.1: Diagrama general de la metodología propuesta.

El entorno de trabajo asume que existe una cámara que converge información con el computador que comanda al robot Sawyer. Dicha cámara no ha de ubicarse dentro del espacio de trabajo del robot. La tarea que se define incluye la clasificación de recipientes,

los cuales gozan de un color diferenciador respecto a la superficie donde se encuentran. El flujo de información representado en la Fig. 3.1 es el siguiente:

- **Etapa 1**: Se utiliza una cámara para obtener una inspección 2D del área de trabajo a fin de identificar todos elementos a clasificar. Empleando el algoritmo de *reconocimiento del objeto* (sección 3.1) se analiza dicha imagen para obtener la posición del objeto de interés, el cual debe ser transformando respecto a la posición de la base del robot.

- **Etapa 2**: La posición del objeto es recibida y se añade la orientación considerando las dimensiones de la garra o *gripper* del robot para obtener la postura deseada. En consecuencia, agrupando la postura de inicio, deseada, final y restricciones de espacio se envían al *planeamiento de ruta*, para generar la ruta geométrica que el robot deberá de seguir. Dicha ruta es enviada a la *optimización de ruta* la cual generará los valores de posición, velocidad y aceleración para cada una de las articulaciones. Ambas acciones son explicadas en *Optimización de trayectoria* (sección 3.2).

- **Etapa 3**: Los valores reales de posición y velocidad serán leídos en todo el proceso y enviados a un servicio de nube como acción de *monitoreo* (sección3.3). Las etapas 1 y 2 se repiten hasta que no existan elementos a clasificar o el usuario desee concluir con la tarea. La lógica empleada es descrita en la sección 3.4.

3.1 Reconocimiento del objeto de interés

Trabajos previos han demostrado la factibilidad de realizar la clasificación de objetos en otro robot de la familia de *Rethink Robotics* [28] empleando segmentación de imágenes usando HSV (iniciales de *hue, saturation, value*; interpretado como tono, saturación, valor). A partir del análisis del espacio de trabajo, el objeto cuenta con las siguientes características:

- Se tiene un recipiente cilíndrico con una superficie superior circular de color entero, el cual mantiene alto contraste con el color de la base.
- El espaciado entre dos objetos es mayor al diámetro circular de la superficie del objeto.

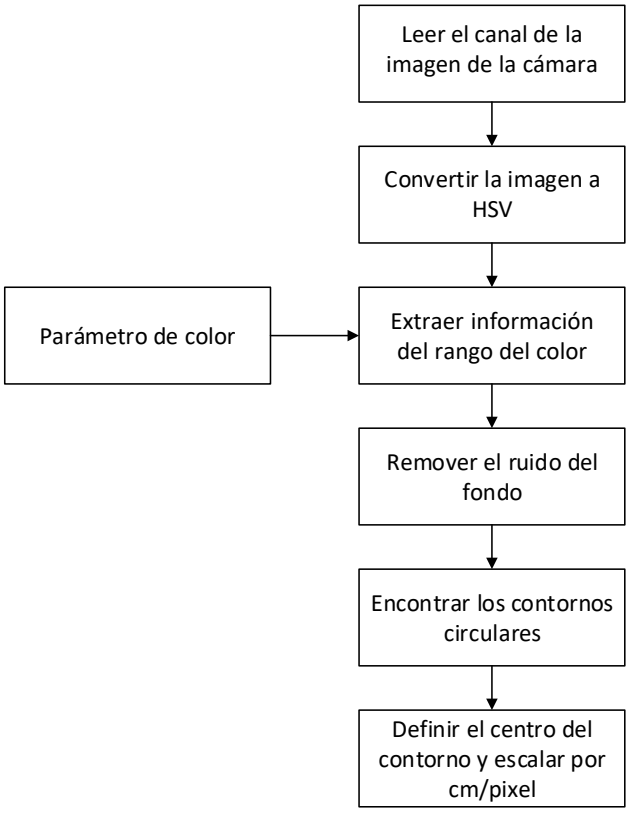


Figura 3.2: Algoritmo propuesto para el reconocimiento del objeto.

El entorno de trabajo se basa en la librería OpenCV, la cual cuenta con las funciones para poder desarrollar las acciones detalladas en la Fig.3.2. La misma se describe a continuación:

- Se define el canal de la cámara y se obtiene la imagen en formato RGB.
- Debido a que se desea la segmentación basada en color, se opta por transformar la imagen de RGB a HSV.

- Definido el rango de parámetros HSV de la superficie del objeto a clasificar, se extrae solo el elemento de interés.
- Se remueve el ruido que no puede formar un cuerpo y que no fue capaz de ser eliminado por el paso anterior.
- Se emplea la transformada de Hough a fin de identificar la superficie circular y determinar el punto centro de la superficie del objeto.
- Se define el contorno de la superficie del objeto y por medio de emplear la densidad pixel/mm se determina la posición del centro del objetivo.
- Se define la posición y orientación cartesiana de la cámara respecto a la base del robot para luego poder obtener la posición del objeto con respecto al sistema de referencia de la base del robot.

El lenguaje de implementación es *python* y se detalla en el Anexo A 1.2.

3.2 Optimización de trayectoria

La optimización de trayectoria, que constituye la etapa 2 de la metodología propuesta, se divide en dos secciones. En la primera, se genera la ruta geométrica que el robot debe de seguir a partir de las posturas recibidas de la Etapa 1. En la segunda, se describe la obtención de los tiempos optimizados en base a una ecuación de costo propuesta.

3.2.1 Planeamiento de la ruta geométrica

Asumiendo que se tiene un conjunto de posturas deseadas y que se encuentran dentro del área de trabajo, es posible determinar las configuraciones articulares para cada una de ellas. Existen diversos métodos que incluyen básicamente el empleo de la cinemática inversa. Sin embargo, lograr la continuidad entre ellas o el menor desplazamiento

entre las misma es un proceso de optimización que incrementa la complejidad de esta tarea. En vista de ello, la comunidad robótica, así como los fabricantes han colocado a disposición de los usuario e investigadores distintas librerías y software que permiten obtener los valores previo a su ejecución en el robot real. En este trabajo, se ha optado por *MoveIt!*[43] por su extenso reconocimiento en el campo del planeamiento de ruta.

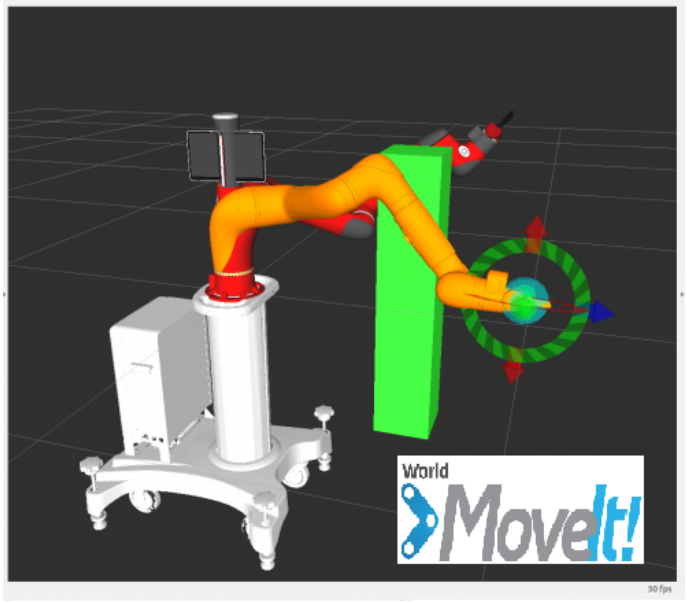


Figura 3.3: Ejemplo de movimiento evitando un objeto empleando *MoveIt!*. En naranja se tiene la posición deseada, en verde el obstáculo que el planeamiento debe de considerar para el movimiento.

MoveIt! Motion Planning Framework [43] es un software empleado para la manipulación móvil, planificación de movimiento, manipulación, percepción 3D, cinemática, control y navegación. Es un software de código abierto utilizado en más de 65 robots entre los que se encuentra el robot Sawyer y Baxter, ambos de la familia de Rethink Robotics. El mismo introduce los limites mecánicos descritos en la Tabla 2.1.

Su interface es gráfica con los recursos de ROS y RVIZ. Sin embargo, para evitar la intervención constante del usuario es necesario emplear su librería teniendo la interfaz como background. Al utilizar *python* como lenguaje de programación obtenemos los valores que serán empleado para ejecutarlos en el robot Sawyer.

Tabla 3.1: Nomenclatura de términos para la optimización.

N_q	Número de articulaciones del robot
N_k	Número de *knots*
h_i	Intervalo de tiempo entre dos *knots* consecutivos
H_i	Tiempo acumulado desde el *knot* inicial hasta el *knot* i-ésimo
T	Tiempo total de ejecución
α	Peso proporcional empleado en la ecuación de costo
f	Frequencia para la ejecución del robot
$Q_j(t)$	Posición de la articulación en el j-ésimo *knot*
V_j	Límite de velocidad de la articulación j-ésimo
$q_j(t)$	Posición del j-ésimo articulación
$\dot{q}_j(t)$	Velocidad de la j-ésimo articulación
$\ddot{q}_j(t)$	Aceleración de la j-ésimo articulación
$\dddot{q}_j(t)$	Sobreaceleración de la j-ésimo articulación

3.2.2 Generación de la ruta discreta

En las acciones mecánicas del robot, los valores de tiempo y sobreaceleración están relacionados de forma inversa. Para lograr ejercer el mínimo tiempo, la sobreaceleración se incrementará, y viceversa. En base a los modelos de ecuaciones de costo en [4, 6], se definen dos objetivos de minimización:

- **Tiempo acumulado**. Definido como el tiempo total de ejecución multiplicado por el número de articulaciones.
- ***Jerk* acumulado**. Definido como la raíz cuadrada de la suma de los cuadrados de la contribución de la sobreaceleración de cada articulación a través del tiempo.

Empleando ambos objetivos, y basados en [3–7, 21, 44],definimos la ecuación de costo C como:

$$C = N_q \sum_{i=1}^{N_k-1} h_i + \sqrt{\sum_{i=1}^{N_q} \int_0^T [\dddot{q}_i(t)]^2 \, \mathrm{d}t} \tag{3.1}$$

Cabe resaltar que un robot con más articulaciones tomará más tiempo acumulado en su desplazamiento, debido a ello, el factor N_q multiplicará al tiempo total. Además, los valores de sobreaceleración pueden ser tanto positivos como negativos por lo cual es necesario elevarlos al cuadrado para realizar su comparación.

Las configuraciones recibidas del planificador de ruta son asociadas a un tiempo; este conjunto se conoce como *knots*. Siguiendo la nomenclatura de la Tabla 3.1, el tiempo entre cada *knot* se denota h_i y como tiempo acumulado H_i, que para $k - simo$ elemento se representa:

$$H_k = \sum_{i=1}^{k-1} h_i, \quad \text{con} \quad H_1 = 0. \tag{3.2}$$

Para realizar la interpolación de estos puntos es necesario definir un vector de tiempos que se representa como $\vec{\boldsymbol{H}}^{\,t} = [H_1 \;\; H_2 \;\; H_3 \;\cdots\; H_{N_k}]^T$. En el caso del control por trayectoria para el robot Sawyer se requiere enviar la información con una frecuencia recomendada de 100 *Hz*. Por lo tanto habrá que construir los valores discretos de la posición, velocidad, aceleración con dicho muestreo de envío. A partir de ello, la ecuación de costo puede ser representada como:

$$C = N_q H_{N_k} + \sqrt{\sum_{j=0}^{(N_q-1)} \sum_{n=1}^{w} (\ddot{q}_j[n])^2} \tag{3.3}$$

El tiempo para la sobreaceleración ha sido transformada a una señal discreta usando los valores de n con el valor entero de $w = fH_{N_k}$.

Entre las consideraciones de la generación de la trayectoria discreta se busca que seguir algunas buenas prácticas de los movimientos mecánicos. Esto incluye, por ejemplo, definir los valores iniciales y finales de la velocidad en 0 y si es posible en las siguientes derivadas, lo cual es además, necesario para evitar movimientos bruscos al partir o terminar en el reposo.

Este trabajo propone emplear splines cúbicos debido a su bajo costo computacional y mayor fidelidad comparado con los B-spline. Entre los spline cúbico, el *spline* cúbico natural se caracteriza por formularse con condiciones de aceleración inicial y final igual 0; por lo cual procedemos a construir un spline con condiciones iniciales y finales nulas para la velocidad.

El *spline* a emplear puede ser representado por una función de posición $Q(t)$ para un tiempo $[p, q]$ y un conjunto de *knots* tal que $p = H_1 < H_2 < ... < H_{N_k} = q$. La interpolación S para Q se compone de ecuaciones polinómicas continuas de grado 3 que cambian de coeficiente entre cada *knot* singular. Describimos dicho polinomio S_j en $[H_j, H_{j+1}],\ \forall j = 1, 2, ..., N_k - 1$ donde

$$S(t) = \begin{cases} a_1 + b_1(t - H_1) + c_1(t - H_1)^2 + d_1(t - H_1)^3, & \text{si} \quad H_1 \leq t \leq H_2 \\ a_2 + b_2(t - H_2) + c_2(t - H_2)^2 + d_2(t - H_2)^3, & \text{si} \quad H_2 \leq t \leq H_3 \\ ... & \\ a_{_{N_k-1}} + b_{_{N_k-1}}(t - H_{_{N_k-1}}) + c_{_{N_k-1}}(t - H_{_{N_k-1}})^2 + & \\ d_{_{N_k-1}}(t - H_{_{N_k-1}})^3, & \text{si} \quad H_{_{N_k-1}} \leq t \leq H_{_{N_k}} \end{cases} \tag{3.4}$$

,

donde a_i, b_i, c_i, d_i son parámetros que necesitan ser hallados en base la información de los *knots* y al vector de tiempo H_i. Reformulando la condición inicial en p es $f'(p)$ de tal forma que $S'(p) = S'_1(p) = f'(p) = b_1$, entonces:

$$S'_1(p) = \frac{1}{h_1}(a_2 - a_1) - \frac{h_1}{3}(2c_1 + c_2)$$

de donde, despejando, se obtiene

$$h_1(2c_1 + c_2) = \frac{3}{h_1}(a_2 - a_1) - 3f'(p) \tag{3.5}$$

Replicando para la condición final en q se tiene $f'(q)$ de tal forma que $S'(q) = S'_{N_k}(p) = f'(p) = b_{N_k}$, se tiene:

$$\begin{aligned} b_{N_k} &= b_{N_k-1} + h_{N_k-1}(c_{N_k-1} + c_{N_k}) \\ &= \frac{1}{h_{N_k-1}}(a_{N_k} - a_{N_k-1}) - \frac{h_{N_k-1}}{3}(2c_{N_k-1} + c_{N_k}) + h_{N_k-1}(c_{N_k-1} + 2c_{N_k}) \\ &= \frac{1}{h_{N_k-1}}(a_{N_k} - a_{N_k-1}) + \frac{h_{N_k-1}}{3}(c_{N_k-1} + 2c_{N_k}), \end{aligned}$$

lo que es equivalente a:

$$h_{N_k-1}(c_{N_k-1} + 2c_{N_k}) = 3f'(q) - \frac{3}{h_{N_k-1}}(a_{N_k} - a_{N_k-1}) \tag{3.6}$$

A partir de (3.5) y (3.6) se propone la matriz tridiagonal A:

$$A = \begin{bmatrix} 2h_1 & h_1 & 0 & 0 & \dots & 0 \\ h_1 & 2(h_1+h_2) & h_1 & 0 & \dots & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & h_{N_k-2} & 2(h_{N_k-2}+h_{N_k-1}) & h_{N_k-1} \\ 0 & 0 & 0 & \dots & h_{N_k-1} & 2h_{N_k-1} \end{bmatrix}$$

De esta forma se tiene:

$$A \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{N_k-1} \\ c_{N_k} \end{bmatrix} = \begin{bmatrix} \frac{3}{h_1}(a_2 - a_1) - 3f'(p) \\ \frac{3}{h_2}(a_3 - a_2) - \frac{3}{h_1}(a_2 - a_1) \\ \frac{3}{h_3}(a_4 - a_3) - \frac{3}{h_2}(a_3 - a_2) \\ \vdots \\ \frac{3}{h_{N_k-1}}(a_{N_k} - a_{N_k-1}) - \frac{3}{h_{N_k-2}}(a_{N_k-1} - a_{N_k-2}) \\ 3f'(q) - \frac{3}{h_{N_k-1}}(a_{N_k} - a_{N_k-1}) \end{bmatrix}$$

A partir de los *knots* se determina los valores de a_i. Por lo tanto, al resolver el sistema

$Ax = b$ se podrá determinar c_i considerando los valores de $p = f'(p) = f'(q) = 0$. En consecuencia determinamos b_i y d_i $\forall i = 1, 2, ..., N_k - 1$ como:

$$b_i = \frac{1}{h_i}(a_{i+1} - a_i) - \frac{h_i}{3}(c_{i+1} + 2c_i)$$
$$d_i = \frac{1}{3h_i}(c_{i+1} - c_i)$$

Resultados previos son mostrados en la Fig. 4.2, donde se detalla que no se cumple con todos los valores iniciales y finales en cero. A partir de ello, se busca disminuir los valores de vibración extendiendo los valores nulos al inicio y final de la aceleración y sobreaceleración. Para ello se definen las condiciones para reemplazar los polinomios S_1 y S_{N_k-1} (definidos en la ecuación (3.4)) por S_u y S_v cumpliendo las siguientes ecuaciones de continuidad:

$$S_u(H_1) = a_1 \qquad S_u(H_2) = S_1(H_2) = S_2(H_2) = a_2$$
$$\dot{S}_u(H_1) = 0 \qquad \dot{S}_u(H_2) = \dot{S}_1(H_2) = \dot{S}_2(H_2) = b_2$$
$$\ddot{S}_u(H_1) = 0 \qquad \ddot{S}_u(H_2) = \ddot{S}_1(H_2) = \ddot{S}_2(H_2) = 2c_2$$
$$\dddot{S}_u(H_1) = 0 \qquad \dddot{S}_u(H_2) = \ddot{S}_1(H_2) = \ddot{S}_2(H_2) = 6d_2$$

$$S_v(H_{N_k-1}) = S_{N_k-1}(H_{N_k-1}) = a_{N_k-1} \qquad S_v(H_{N_k}) = a_{N_k}$$
$$\dot{S}_v(H_{N_k-1}) = \dot{S}_{N_k-1}(H_{N_k-1}) = b_{N_k-1} \qquad \dot{S}_v(H_{N_k}) = 0$$
$$\ddot{S}_v(H_{N_k-1}) = \ddot{S}_{N_k-1}(H_{N_k-1}) = 2c_{N_k-1} \qquad \ddot{S}_v(H_{N_k}) = 0$$
$$\dddot{S}_v(H_{N_k-1}) = \dddot{S}_{N_k-1}(H_{N_k-1}) = 6d_{N_k-1} \qquad \dddot{S}_v(H_{N_k}) = 0$$

Al tener ocho puntos de referencias, se construye las ecuaciones polinómicas de séptimo grado con la siguiente forma:

$$\begin{aligned} S_u(t) =& a_{1u} + a_{2u}(t - H_1) + a_{3u}(t - H_1)^2 + a_{4u}(t - H_1)^3 + a_{5u}(t - H_1)^4 + \\ & a_{6u}(t - H_1)^5 + a_{7u}(t - H_1)^6 + a_{8u}(t - H_1)^7 \end{aligned} \tag{3.7}$$

$$\begin{aligned} S_v(t) =& a_{1v} + a_{2v}(t - H_{N_k-1}) + a_{3v}(t - H_{N_k-1})^2 + a_{4v}(t - H_{N_k-1})^3 + a_{5v}(t - H_{N_k-1})^4 + \\ & a_{6v}(t - H_{N_k-1})^5 + a_{7v}(t - H_{N_k-1})^6 + a_{8v}(t - H_{N_k-1})^7 \end{aligned} \tag{3.8}$$

De las condiciones iniciales se deduce:

$$\begin{array}{llll} a_{1u} = a_1 & a_{2u} = 0 & a_{3u} = 0 & a_{4u} = 0 \\ a_{1v} = a_{N_k-1} & a_{2v} = b_{N_k-1} & a_{3v} = 2c_{N_k-1} & a_{4v} = 6d_{N_k-1} \end{array}$$

Los puntos determinados anteriormente pueden ser resumidos en las siguientes matrices:

$$\begin{bmatrix} h_1^7 & h_1^6 & h_1^5 & h_1^4 \\ 7h_1^6 & 6h_1^5 & 5h_1^4 & 4h_1^3 \\ 42h_1^5 & 30h_1^4 & 20h_1^3 & 12h_1^2 \\ 210h_1^4 & 120h_1^3 & 60h_1 & 24h_1 \end{bmatrix} \begin{bmatrix} a_{8u} \\ a_{7u} \\ a_{6u} \\ a_{5u} \end{bmatrix} = \begin{bmatrix} a_2 - a_1 \\ b_2 \\ 2c_2 \\ 6d_2 \end{bmatrix}$$

$$\begin{bmatrix} h_{N_k-1}^7 & h_{N_k-1}^6 & h_{N_k-1}^5 & h_{N_k-1}^4 \\ 7h_{N_k-1}^6 & 6h_{N_k-1}^5 & 5h_{N_k-1}^4 & 4h_{N_k-1}^3 \\ 42h_{N_k-1}^5 & 30h_{N_k-1}^4 & 20h_{N_k-1}^3 & 12h_{N_k-1}^2 \\ 210h_{N_k-1}^4 & 120h_{N_k-1}^3 & 60h_{N_k-1}^2 & 24h_{N_k-1} \end{bmatrix} \begin{bmatrix} a_{8v} \\ a_{7v} \\ a_{6v} \\ a_{5v} \end{bmatrix} = \begin{bmatrix} a_{N_k} - h_{N_k-1}^3 a_{4v} - h_{N_k-1}^2 a_{3v} - h_{N_k-1} a_{2v} - a_{1v} \\ 0 - 3h_{N_k-1}^2 a_{4v} - 2h_{N_k-1} a_{3v} - a_{2v} \\ 0 - 6h_{N_k-1} a_{4v} - 2a_{3v} \\ 0 - 6_{N_k-1} \end{bmatrix}$$

En balance, definiendo los valores h_i se podrá generar el spline cubico modificado.

3.2.3 Optimización del tiempo

La ecuación de costo de manera discreta quedó definida en (3.3). La optimización deberá de generar los valores de h_i de tal forma que se pueda generar la ruta. En este sentido, es necesario identificar el procedimiento que se puede ser empleado para resolver el problema de minimización de dicha ecuación de costo. Previo a ello es necesario identificar las restricciones mecánicas a considerar.

En la Tabla 2.1 se definieron los límites articulares que el robot Sawyer tiene en cada articulación y que a pesar de poder ser enviados a los controladores, no podrían ser ejecutados debido a que su sistema interno de seguridad provocaría la finalización de la tarea o los ignoraría. Las restricciones de velocidad, de acuerdo con la Tabla 3.1, se definen como $V_j,\ \forall j = 0 : N_q - 1$. Igualmente, la diferencia de posición entre dos *knots* consecutivos será definida como:

$$q_j^{i+1} - q_j^i = \Delta q_j^i, \quad \forall i = 1 : N_k - 1$$

La variación de desplazamiento alcanzará su tiempo mínimo cuando se ejerce una velocidad constante igual a la máxima velocidad posible para la articulación. Sin embargo, entre cada *knot* actúan N_k articulación es con diferentes tiempos mínimos; por tanto, el máximo valor entre ellos será el tiempo mínimo que puede asumir h_i entre dos *knots* consecutivos, representado en:

$$h_i^{\text{mín}} = \underset{j=0,\ldots,N_q-1}{\text{máx}} \left\{ \frac{\Delta q_j^i}{V_j} \right\}. \tag{3.9}$$

Para N_k *knots* empleando (3.2) definimos el vector de tiempo $\boldsymbol{H}_{min}^t$, el cual representa el punto de inicio para el proceso de optimización pues reúne las restricciones como una condición de frontera. de tal forma que el tiempo de ejecución del robot se resumen en el vector:

$$\vec{\boldsymbol{H}}^{t}_{\text{mín}} = \begin{bmatrix} 0 & H_2^{\text{mín}} & H_3^{\text{mín}} & \cdots & H_{N_k}^{\text{mín}} \end{bmatrix}^T.$$

Esta tesis propone dos métodos de optimización que parten de la interacción con $\boldsymbol{H}^{t}_{min}$ el cual busca disminuir el costo computacional al resolver el problema de optimización. Se comienza asumiendo que $\vec{\boldsymbol{H}}^{t}_{\text{mín}}$ representa un valor proporcional respecto al valor óptimo, por lo cual definimos que el valor óptimo será la multiplicación de un escalar r sujeto a $r > 1$:

$$\vec{\boldsymbol{H}}^{t}_{\text{opt}} = r \cdot \vec{\boldsymbol{H}}^{t}_{\text{mín}},$$

de tal forma que se tiene la ecuación de costo en (3.3) como:

$$\min_{r} \left\{ \alpha r N_q H_{N_k}^{\text{mín}} + (1-\alpha) \sqrt{\sum_{j=0}^{N_q-1} \sum_{n=1}^{(afH_{N_k}^{\text{mín}})} (\dddot{q}_j[n])^2} \right\} \tag{3.10}$$

A fin de mejorar la solución de la optimización, se replantea el factor a multiplicar elemento a elemento (producto de *Hadamards*) por el vector de tiempo mínimo $\vec{\boldsymbol{H}}^{t}_{\text{mín}}$. En este caso se busca independizar las sección de tiempo h_i de cada *knot* al multiplicarse por un valor r_i del vector R:

$$\boldsymbol{R} = [r_1 \; r_2 \; ... \; r_{N_k-1}] \; , \; \forall \boldsymbol{r_i} > 1$$

$$\vec{\boldsymbol{H}}^{t}_{\text{opt}} = \boldsymbol{R} \circ \vec{\boldsymbol{H}}^{t}_{\text{mín}},$$

donde $\circ$ representa el producto de Hadamards. Redefiniendo nuestra ecuación de costo (3.3), al discretizarla empleando una frecuencia f será:

$$\min_{\boldsymbol{R}} \left\{ \alpha r_{N_k} N_q H_{N_k}^{\text{mín}} + (1-\alpha) \sqrt{\sum_{j=0}^{N_q-1} \sum_{n=1}^{(r_{N_k} f H_{N_k}^{\text{mín}})} (\dddot{q}_j[n])^2} \right\} \tag{3.11}$$

Para ambos métodos el paso final incluye emplear un solucionador. La bibliografía menciona reiteradas veces el empleo de SQP. Al emplear las restricciones en (3.11), y la variable α es posible emplear el método LBFGS-B (*Limited-memory Broyden Fletcher Goldfarb Shanno*). La implementación en python se detalla en el Anexo A 1.1.

3.3 Monitoreo

El monitoreo tiene como finalidad enviar la información adquirida desde los tópicos obtenidos por ROS del robot Sawyer a una interface web la cual puede ser accedida desde cualquier parte del mundo con acceso a internet. Para ejecutar el envío de información se emplea el servicio IoT de IBM Cloud.

3.3.1 IBM Cloud

IBM Cloud es una propuesta de IBM que reúne los recursos de cloud en una única plataforma bajo un plan de pago o gratis con recursos limitados que permite la convergencia de múltiples datos, escalar sistemas e incorporar servicios cognitivos para identificar y enfocar el valor de un proyecto de forma rápida y económica comparado a múltiples servicios que se debería de contratar. Su principal ventaja es el ahorro de recursos en el diseño de una infraestructura y la reducción total del empleo de hardware, así como su mantenimiento para el desarrollo de aplicaciones.

IBM Watson^{TM} IoT Platform [45] para IBM Cloud es un kit de utilidades versátiles que incluye traspaso de información de dispositivos, gestión de dispositivos y acceso a las aplicaciones por enlaces MQTT. Empleando este servicio se puede recopilar datos de distintos dispositivos conectados y la posibilidad de realizar e identificar patrones en los datos en tiempo real desde el robot. Para emplear este servicio la implementación requiere los siguiente pasos previos:

- Cuenta de IBM Cloud
- Interfaz de línea de comandos de Cloud Foundry (CLI)
- Git

Se emplea este servicio sobre los otros expuestos en la sección 2.6.1 debido a que se goza de una cuenta educativa que permite utilizar el servicio sin costo alguno por ser estudiante. A continuación, se explica el procedimiento propuesto, y empleado, para la construcción del servicio del monitoreo.

Procedimiento

Desde la línea de comandos, se define el punto final del API ejecutando el mandato cf api para la región sudamericana. Se creó el servicio de Watson IoT Platform en IBM Cloud con el nombre *IIOT-UTEC* para la versión lite disponible para estudiantes. Luego se empleó un modelo por defecto el cual adaptamos para el envío de la información desde una programa en python ya que acceder a la información del robot es nativo en este lenguaje. Los siguientes son los comandos utilizados:

```
$ cf api https://api.ng.bluemix.net
$ cf create-service iotf-service iotf-service-free IIOT-UTEC
$ git clone https://github.com/ibm-watson-iot/guide-conveyor-rasp-
    pi
```

Previamente, en la plataforma se define la información del dispositivo robótico, el cual debe ser configurado de forma local en el archivo *define.conf*. En este archivo se define la información de organización, tipo, id, método de comunicación y *token*.

El mensaje a enviar se define con un formato JSON, el cual se construye al leer los tópicos de posición/velocidad del robot. Para esto se crea la función *SendIOT* la cual

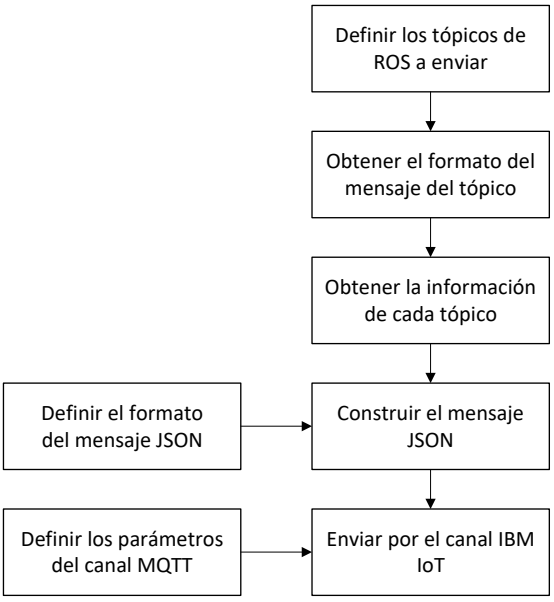


Figura 3.4: Algoritmo propuesto para el envío de datos desde los tópicos de ROS.

tiene un periodo de envío de 0.1 seg. El código a emplear se detalla en el Anexo A en 1.3. Con esto se dispone el envío de la información a la plataforma que nos ofrece el servicio IBM Watson IoT. Sin embargo, para que dicha información sea vista por usuarios no propietarios del servicio, es necesario crear una interface web. IBM Cloud trabaja sobre una plataforma NodeJS. El procedimiento es explicado a continuación.

3.3.2 NodeJS

Es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome que permite acceder información desde un servicio y tratarla para la generación de acciones o visualización. Se emplea un modelo base que se extrae desde GitHub de la aplicación de muestra *Widget Library Monitoring*.

```
1 $ git clone https://github.com/ibm-watson-iot/guide-conveyor-ui-html
2 $ npm install
```

La interfaz es programada en formato HTML usando la librería *widget*. Para cada representación de variable del estado del robot se debe de llamar la variable definida en el mensaje JSON. Para el caso de $righ\_j0$ se tendría:

```
WIoTPWidget.CreateGauge("chart_01","joint_angle", "robot-utec",
"sawyer-intera", "right_j0" ,{
  label: {
    format: function(value, ratio) {
      return value;
    },
    show: true // to turn off the min/max labels.
  },
  min: -180.0,
  max: 180.0,
  units: 'Grados',
},['#FF0000', '#F97600', '#F6C600','#60B044']);
```

HTML brinda flexibilidad para mejorar el aspecto visual de la interface usando, por ejemplo, CSS. Una vez construido la página de forma local, es necesario subir el servicio de NodeJS para obtener un enlace web. En el archivo *manifest.yml* se describe las capacidades de esta aplicación y si tendrá alguna conexión. En este caso se realiza la conexión empleando servicio de IBM Watson *IIOT-UTEC*.

```
declared-services:
  IIOT-UTEC:
    plan: iotf-service-free
applications:
- path: .
  memory: 128M
  instances: 1
  domain: mybluemix.net
  disk_quota: 1024M
services:
- IIOT-UTEC
```

Por último, se eleva la aplicación con el comando $cfpush * **$, definiendo el nombre del dominio como deseemos y con ello el link web.

3.4 Integración

En esta sección se busca concretar la ejecución del objetivo general al diseñar un sistema que permita al robot manipulador realizar tareas de clasificación flexible optimizando los valores de tiempo y sobreaceleración con monitoreo remoto. En la Fig. 3.5, se detalla el algoritmo que se busca implementar con dicho objetivo. A continuación, se presenta los parámetros a tomar en cuenta en cada subtarea:

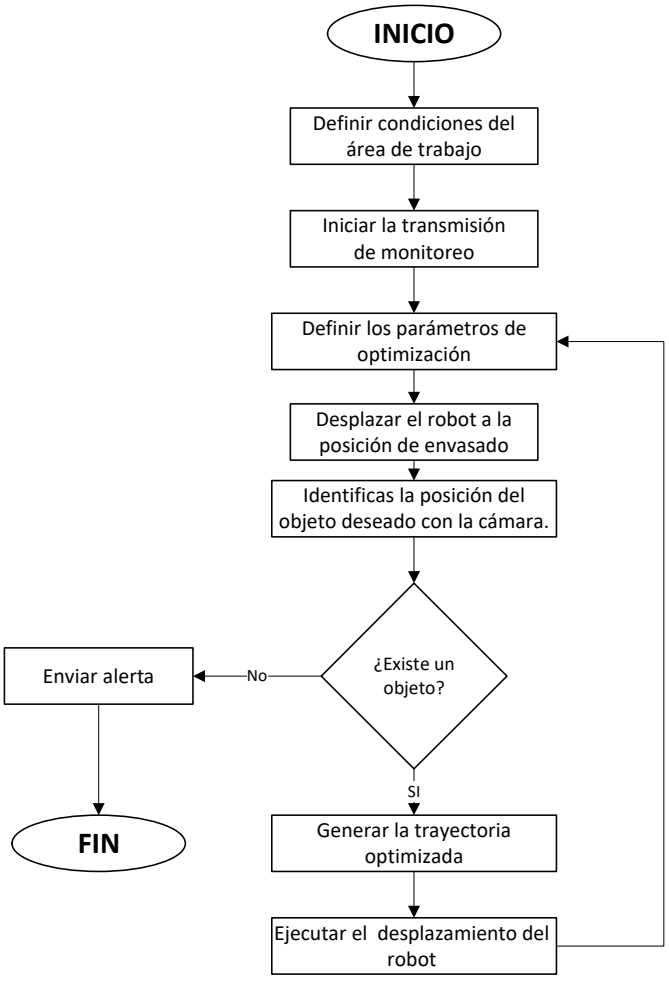


Figura 3.5: Algoritmo para ejecutar el funcionamiento de la tarea final.

- En la primera etapa se tiene que identificar el área de trabajo. En la Fig. 4.18 se observa que el área de trabajo se complementa con una superficie donde se encuentran los elemento a clasificar. En este sentido, es necesario que el robot identifique cuál es el rango de posiciones cartesianas que puede ejecutar a fin de evitar una colisión con el entorno. Para ello, es necesario redefinir la mesa como un obstáculo que se define como 8 puntos de un octaedro, el cual es incorporado al emplear *MoveIt!*.

- Luego se inicia con el envío de información como tarea de monitoreo. En ella se debe abrir los canales MQTT que permita el traslado del mensaje con el formato JSON. Asimismo, dependiendo de las condiciones de red, se deberá de modificar la frecuencia de envío de la información.

- El parámetro α será publicado en un tópico para que pueda ser variado independientemente de la ejecución de la tarea. Entonces, cada vez que el robot culmine con clasificar un objeto, se actualizará dicho valor.

- Previo a la tarea de clasificación el robot se moverá a una posición inicial a fin de garantizar posiciones redundantes que obliguen realizar movimientos innecesarios.

- Se emplea el algoritmo que nos permite identificar el objetivo a clasificar. Previamente se realiza la calibración para determinar qué característica deberá de tener dicho elemento para ser desplazado.

- A partir de ello, se realiza un bucle de clasificación hasta que se concluye con todos los elementos que se ubican en el área de trabajo.

La ejecución de la integración implica el uso simultáneo de la ejecución de los tres objetivos. Para ello, la implementación ya tiene en consideración parámetros de compatibilidad entre los canales, a fin de reducir tiempos muertos en el traspaso de la información. Cabe resaltar que los formatos usados se basan en garantizar los valores mínimos de tiempo muerto. Los detalles de programción se ubican en el Anexo A 1.4.

CAPÍTULO 4

RESULTADOS Y DISCUSIÓN

En este capítulo presenta los resultados empleando la metodología propuesta en el capítulo anterior. Además, se expone los resultados previos obtenidos y se discute las razones de su modificación. La organización de este capítulo se encuentra en relación a los objetivos específicos por medio de las tres primeras secciones. Por último, en la sección 4.4 se muestra cómo la integración de los resultados de los objetivos particulares cumple con el objetivo general planteado. El sistema operativo por defecto en las aplicaciones del fabricante del robot Sawyer es Ubuntu Kinetic 16.04. Para lograr la mayor compatibilidad posible se presentan los resultados utilizando Python, debido a que es el lenguaje que utiliza por defecto el robot para la mayoría de nodos. Así mismo, la formulación de estos resultados han sido expuestos en un paper-conference en [46].

4.1 Implementación de la optimización de trayectoria

En esta sección se exponen los resultados previos y el procedimiento específico para la implementación en el robot Sawyer.

4.1.1 Control por medio de tópicos

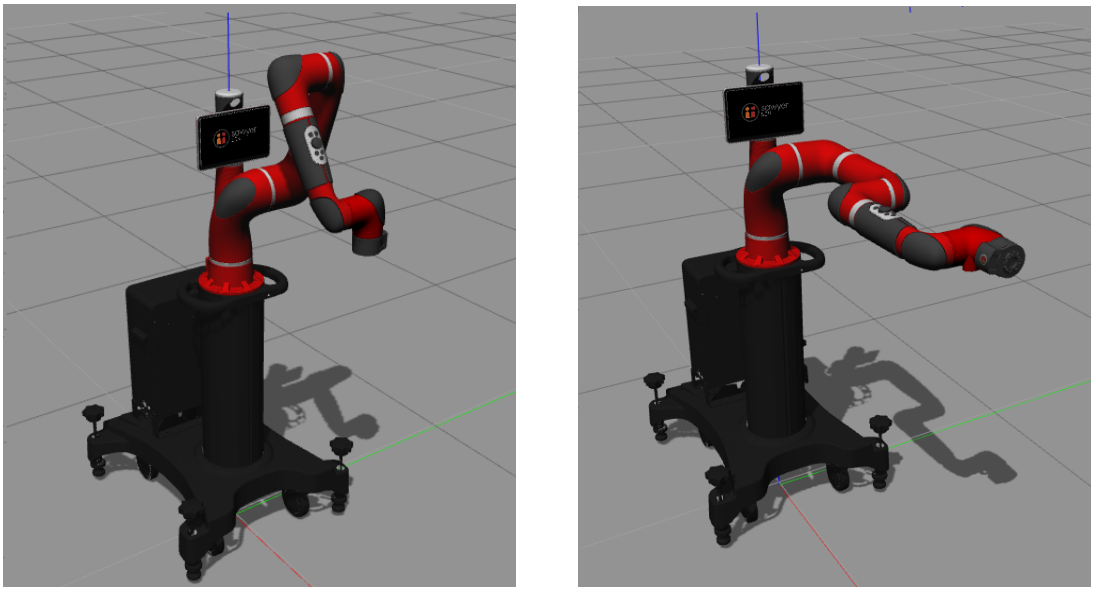
Para realizar el control del robot Sawyer, la librería del fabricante nos permite realizar movimientos en el robot sin la necesidad de emplear los tópicos del entorno de ROS. Sin embargo, para ejecutar el control por trayectoria es necesario emplear los tópicos debido al retraso en el envío de los parámetros producto de validaciones de seguridad empleadas por las funciones de dicha librería.

De acuerdo a [37], para ejercer un control directo en dicho manipulador, se tiene que publicar los valores deseados en el tópico $/robot/limb/right/joint\_command$ con una frecuencia recomendada de 100 Hz.Al enviar las señales de control en el tópico, ya sea para realizar un control de posición o trayectoria, se evaden la mayoría de elementos de seguridad en los motores de cada articulación. Recordemos que el robot tiene 7 articulaciones y la vibración de cada articulación afecta en secuencia a las demás. En tanto, es posible que este tipo de publicaciones produzca que el robot genere aún mayores vibraciones si es que el envío de valores genera picos en velocidad o aceleración. Debido a ello, en las primeras etapas de las pruebas se realizarán simulaciones en el entorno de *Gazebo*, el cual es un simulador dinámico ampliamente utilizado en robótica.

4.1.2 Generación de trayectoria cúbica

A fin de ejecutar un ejemplo de control de trayectoria, es necesario generar ciertos casos de prueba y con ello los valores de posición, de velocidad y aceleración a partir de los *knots* y de un vector de tiempo. Como primer caso, se propone que el desplazamiento deseado, denominado *tarea A1*, el cual deberá de seguir la configuración desde la *posicion_neutra* (en Fig. 4.1a), que es definida por la librería del fabricante, a la *posición_cero* (en Fig. 4.1b) que es la posición en que todas las articulaciones en valor cero y volver nuevamente a la *posicion_neutra*.

Con solo 3 *knots* (a-b-a) es insuficiente la cantidad de puntos para lograr una interpolación cúbica; por ello, es necesario introducir valores intermedios. Se agregan un total de 8 *knots* internos con forma lineal respecto a los *knots* principales, siendo estos proporcionales al vector de tiempo $\vec{\boldsymbol{H}}^{\,t} = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]s$. Entonces, definidos los puntos para las articulaciones q_i y $\vec{\boldsymbol{H}}^{\,t}$ se inicia con el proceso de interpolación.



(a) (b)

Figura 4.1: Movimiento de *tarea A1*. El robot se desplazará de siguiendo la configuración (a)-(b)-(a) empleado *splines* simulados en el entorno de Gazebo.

El primer método analizado fue el *spline natural*. Dicho *spline* logra construir las ecuaciones polinómicas de grado tres cumpliendo con desplazarse por los puntos deseados y una aceleración final e inicial igual a 0, es decir, $\ddot{S}(0) = 0$ y $\ddot{S}(H_{N_k}) = 0$. Para dicho método existe una extensa cantidad de librerías y algoritmos que facilitan su obtención a muy bajo costo computacional. Dicha trayectoria generada se muestra en la Fig.4.2a; sin embargo, posteriormente la solución habrá que descartarla al no garantizar que la velocidad inicial es 0 puesto que el hecho de que no inicie en 0 obliga al controlador interno del robot a ejercer la máxima velocidad posible para lograr en el menor tiempo llegar a dicha velocidad inicial que propone la trayectoria teórica. La simulación, mostrada en la Fig. 4.3, muestra dicho escenario donde el comportamiento de la velocidad al iniciar y finalizar el movimiento implica un desplazamiento abrupto hasta su estabilización.

Por otro lado, el *clamped spline* se caracteriza por garantizar la velocidad inicial y final en una constante. De acuerdo a lo mencionado, dicha constante deberá de ser 0 y se introduce en las condiciones como $\dot{S}(0) = 0$ y $\dot{S}(H_{N_k}) = 0$. A partir de la ecuación 3.4, se

realiza la demostración de cómo obtener los valores del *spline* $S(t)$ que son interpretados como posición (q_i), velocidad ($\dot{q}_i$) y aceleración ($\ddot{q}_i$) para cada una de las articulaciones i. Los resultados se observan en la Fig.4.2b, donde los indicadores del gráfico representan los *knots* definidos como puntos de desplazamiento deseado. A pesar de que los valores picos de aceleración para el método *clamped* es mucho menor comparado con el método *natural*, su problema radica en que su aceleración no inicia ni culmina en 0, y aunque los pico en aceleración son menores al anterior caso, de igual manera vibración al iniciar y concluir el movimiento en el robot real.

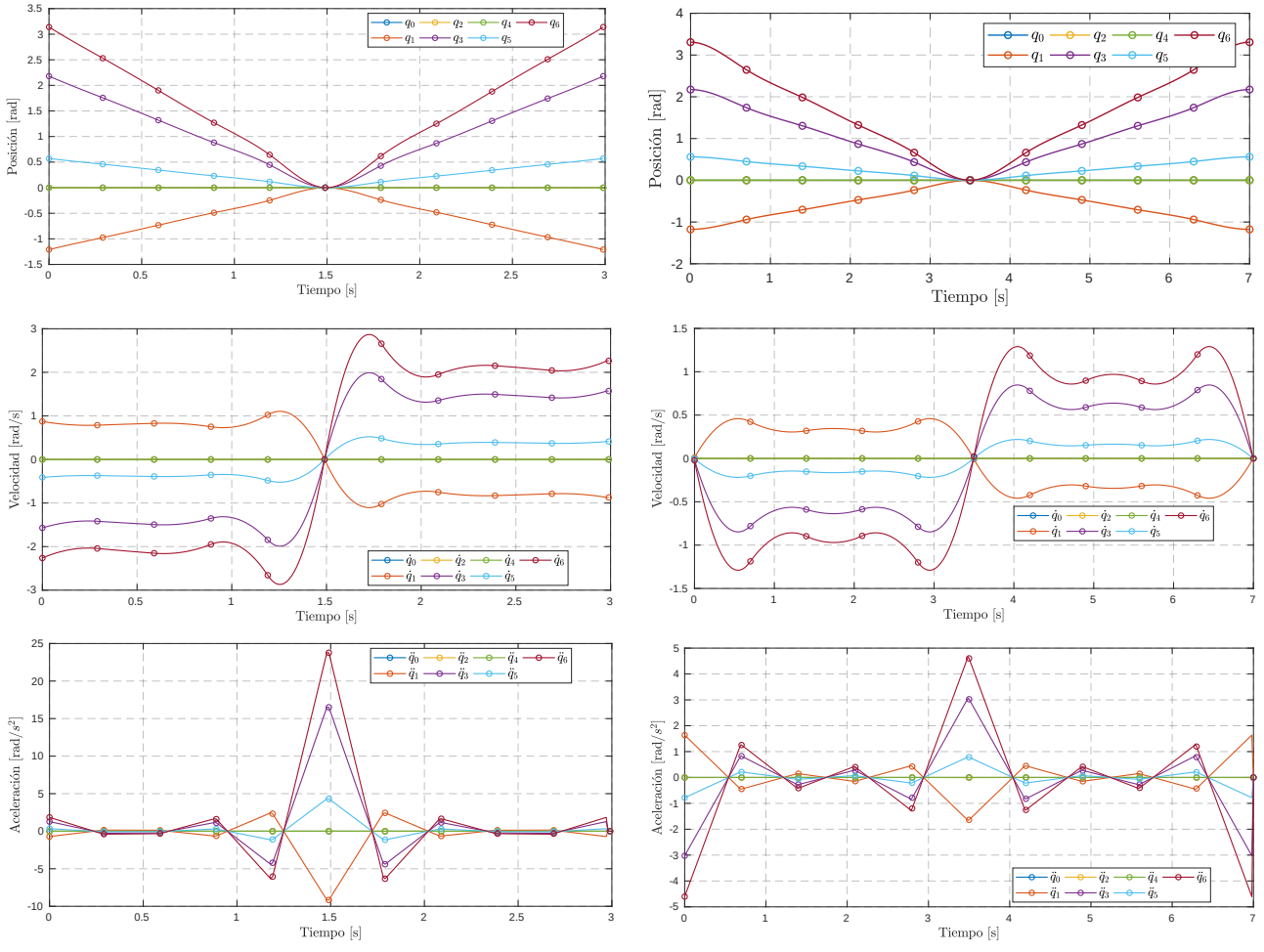


Figura 4.2: Trayectoria generada para la *tarea A1*.(a) Valores generados empleando *Natural spline*. (b) Valores generados empleando *Clamped spline*.

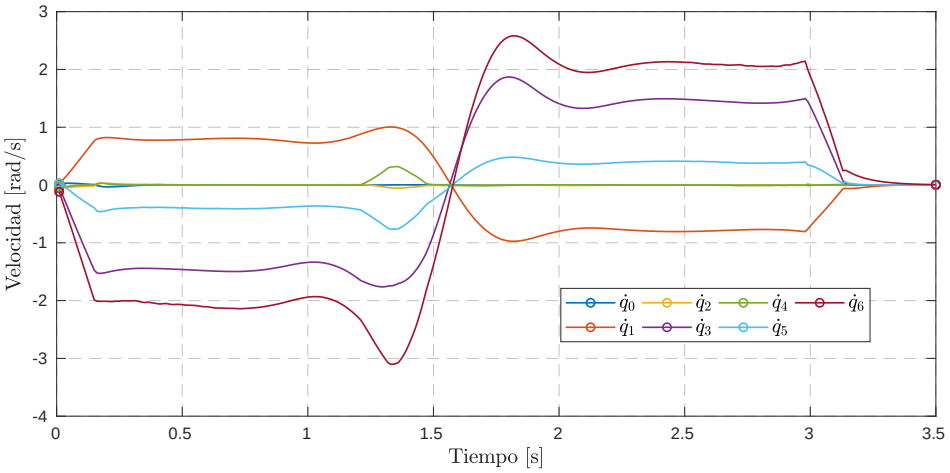


Figura 4.3: Velocidad resultante de la simulación de la *tarea A1* empleando un *spline natural* para la interpolación.

En vista de lo anterior, la metodología propuesta valora los resultados del método *clamped* sobre el *natural* debido a que el objetivo busca reducir dicha vibración inicial y final del desplazamiento. Para esta tesis, se propone no solo garantizar un inicio de velocidad en cero sino también los otro componentes de interés. Para ello, habrá que reemplazar las ecuaciones S_1 y S_{N_k-1} del *spline* por ecuaciones de séptimo grado que tendrá como raíces las condiciones de posición, ademas de velocidad, aceleración y sobreaceleración con valores igual a 0.

El control del fabricante, emplea el control de posición del robot o también denominado método de cinemática directa el cual tiene como única entrada los *knots* definidos. En la *tarea A1* se comparan con el método del splines de nuestra metodología. Al emplear el control cinemático, no se puede controlar el tiempo de ejecución. Las pruebas en el robot real, empleando las condiciones del fabricante, ejecutaron la *tarea A1* en 7.5 seg. por lo que se propone que el control de trayectoria se realice en 7 seg. Los resultados se comparan en la Fig.4.4. De la comparación, a nivel de velocidad, el control cinemático emplea una velocidad constante que se respalda en el controlador interno PID que se explica en la bibliografía del fabricante. El hecho de variar entre valores de velocidad constante produce picos de aceleración con valores mayores al control por trayectoria.

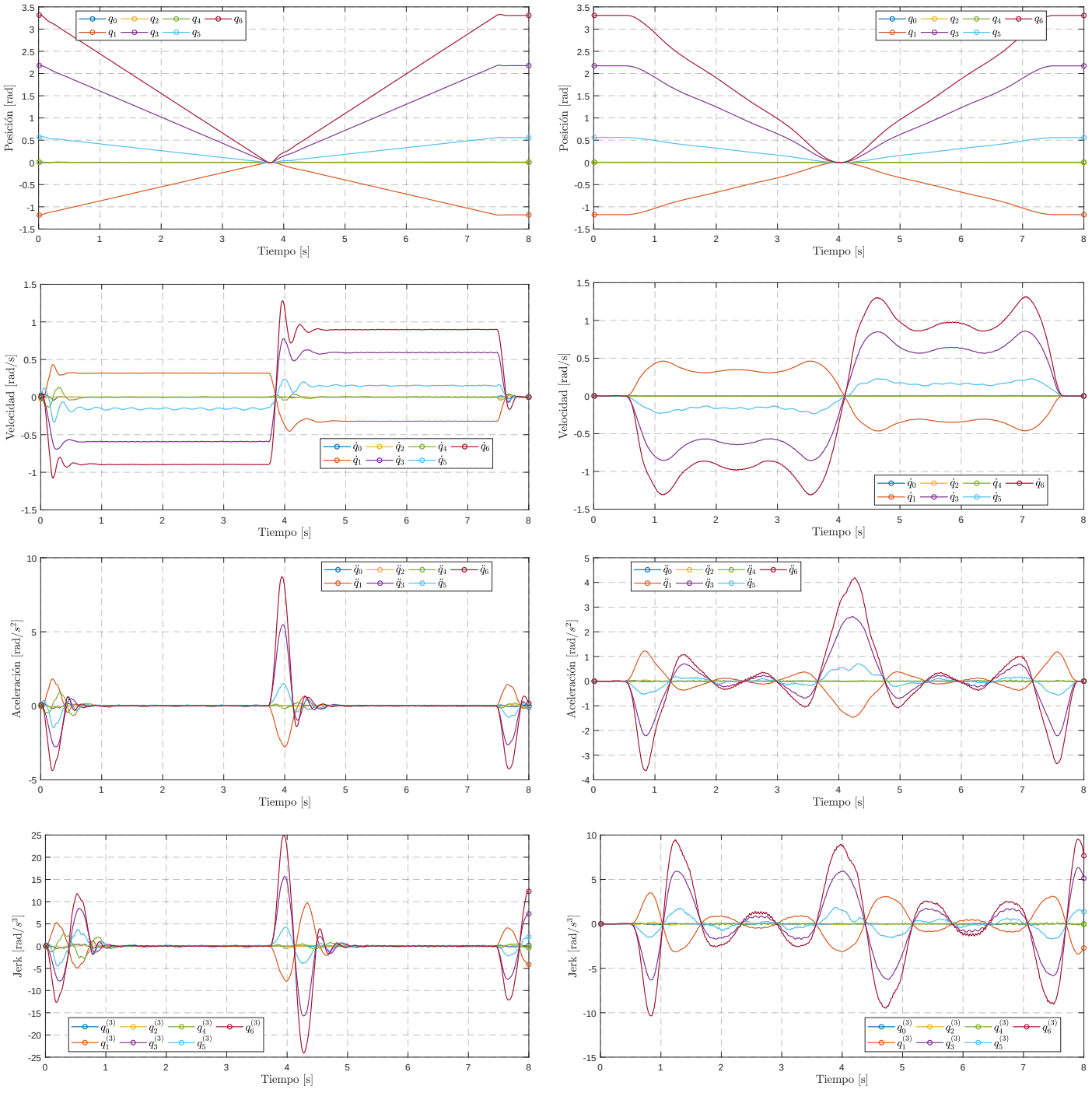


Figura 4.4: Comparación del desplazamiento para la *tarea A1*. (a) Desplazamiento con cinemática directa. (b)Desplazamiento con *Clamped spline*.

Ello, en consecuencia, genera los momentos de vibración en el robot. En el caso del *knot* ubicado alrededor de 4seg. los valores de nuestra propuesta representa prácticamente la mitad del valor pico. Finalmente, a nivel de sobreaceleración o *jerk*, los picos generados tienen poco más del doble del valor que el generado por el control de trayectoria. Este efecto es proporcional a las vibraciones, por lo tanto, se puede decir que la vibración se disminuye empleando nuestro método de optimización de trayectoria.

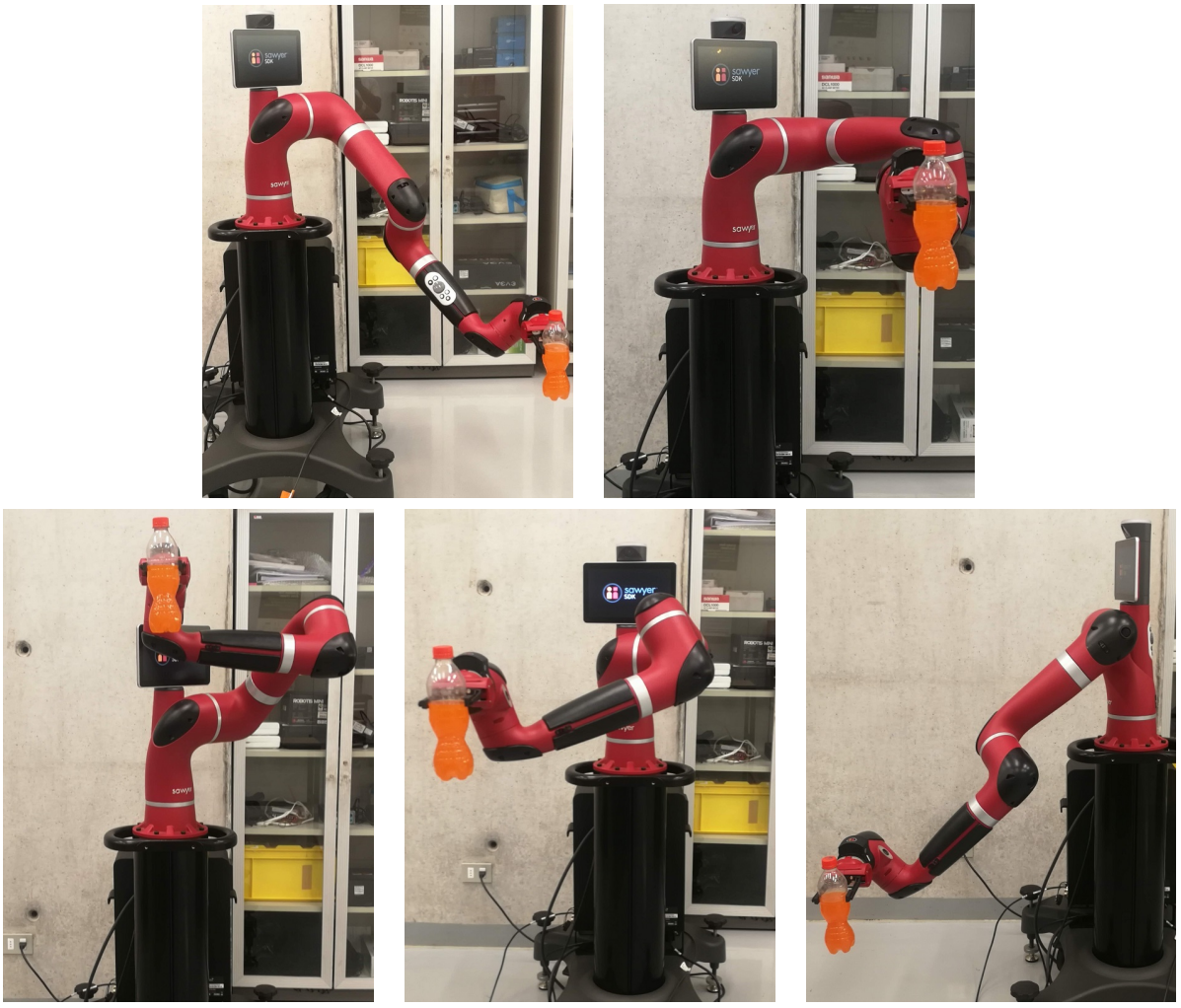


Figura 4.5: Comparación del desplazamiento para la *tarea A2* con *Clamped spline* y cinemática directa.

Las pruebas anteriores han demostrado que con una actividad básica como la definida cómo *tarea A1* el método propuesto, en similar tiempo de ejecución, tiene mejores resultados respecto a la vibración en su desplazamiento. Sin embargo, es necesario identificar como optimizar dicho movimiento pues, hasta el momento, $\vec{\boldsymbol{H}}^t$ ha sido definido por el usuario y la idea principal para una tarea autónoma es que ésta sea obtenida por medio del sistema que se propone.

Por este motivo, se define la *tarea A2*, la cual busca escenificar el movimiento tradicional de una tarea *pick-up*, es decir, es una tarea en la cual un robot manipulador es empleado para el desplazamiento de un objeto a fin de colocarlo un recipiente que luego se empleará para empacarlo. Dicha tarea se resume en el desplazamiento de 5 configuraciones deseadas o *knots* que se observan en la Fig.4.5. Estas posiciones son determinadas a partir de los valores cartesianos de la articulación final. Empleando la interfaz de *MoveIt!* es posible obtener los valores de la posición para cada articulación mediante su función de cinemática inversa. Además, se agregan nodos de apoyo que permiten al robot aumentar la fidelidad del movimiento.

Empleando la optimización define en la sección 3.2.3, se propone la generación de trayectoria para distintos valores de α. A continuación, se exponen dos casos: en el primero, donde se tiene prioridad en reducir los valores de sobreaceleración (y con ello la vibración) y en el segundo caso se prioriza la ejecución en un tiempo reducido. Ambos casos son producto del valor que se asigne a α. Para garantizar un caso con baja sobre aceleración se propone $\alpha = 0.2$, el cual la ruta generada se observa en la Fig.4.6a. Mientras que para generar un caso donde la relevancia se encuentra el tiempo, se construye la trayectoria con $\alpha = 0.8$ expuesto en Fig.4.6b.

En la trayectoria generada para $\alpha = 0.2$ se observan los valores para las siete articulaciones del robot Sawyer. En el plano de la velocidad, se puede afirmar que se logró garantizar que su valor inicial y final sean 0, además de una continuidad en sus valores que se respalda en la eliminación de picos de su siguiente derivada. En este sentido, la

aceleración evidencia valores que no tienen la misma naturaleza debido al reemplazo de las ecuaciones iniciales y finales por las de séptimo grado S_u y S_v (definidas en (3.7) y

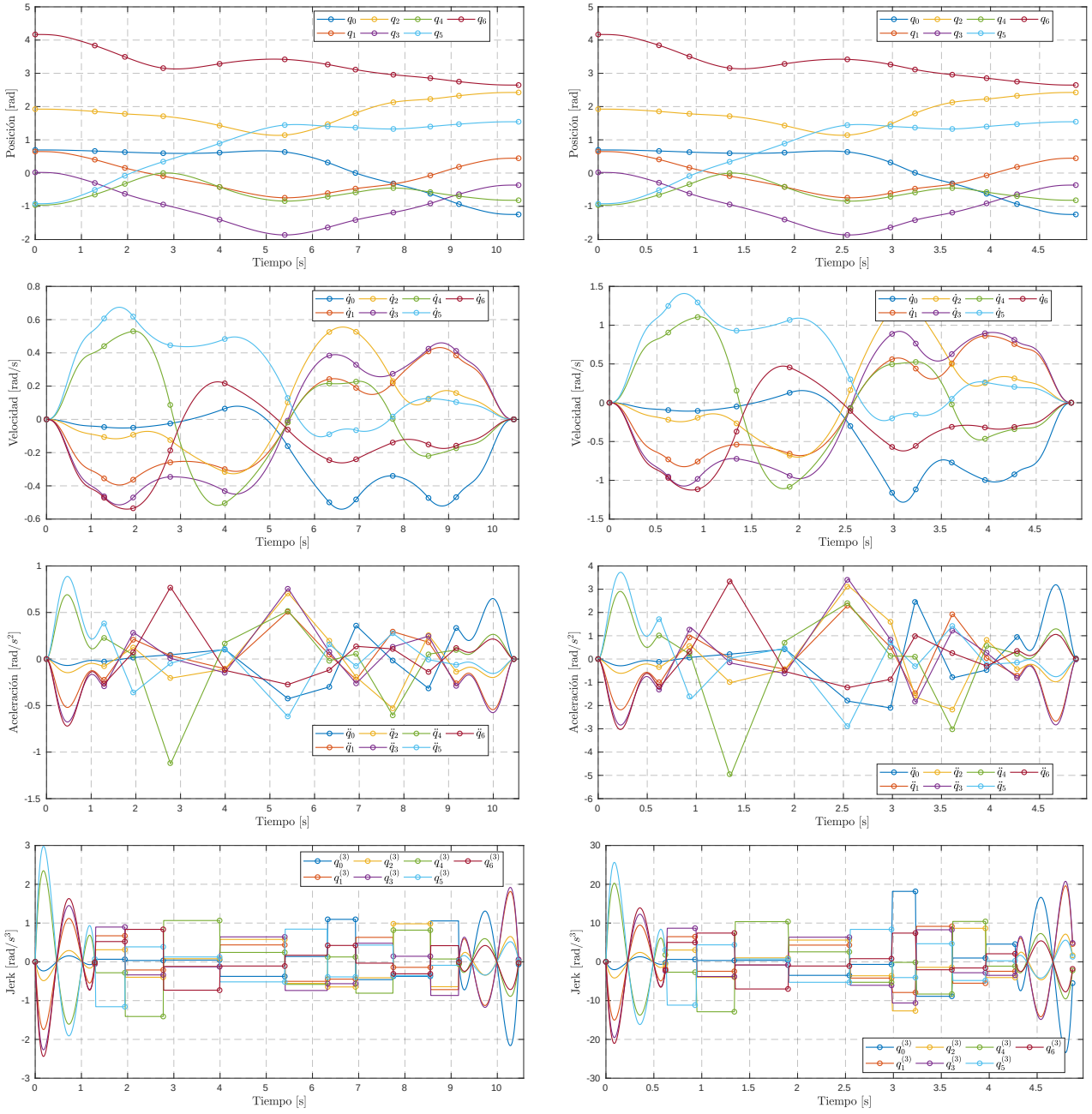


Figura 4.6: Comparación de trayectoria generada para la *tarea A2*. (a) Trayectoria generada con $\alpha = 0.2$. (b) Trayectoria generada con $\alpha = 0.8$.

(3.8)); exceptuando de ello, el hecho de aplicar un *spline* polinómico de grado tres, su

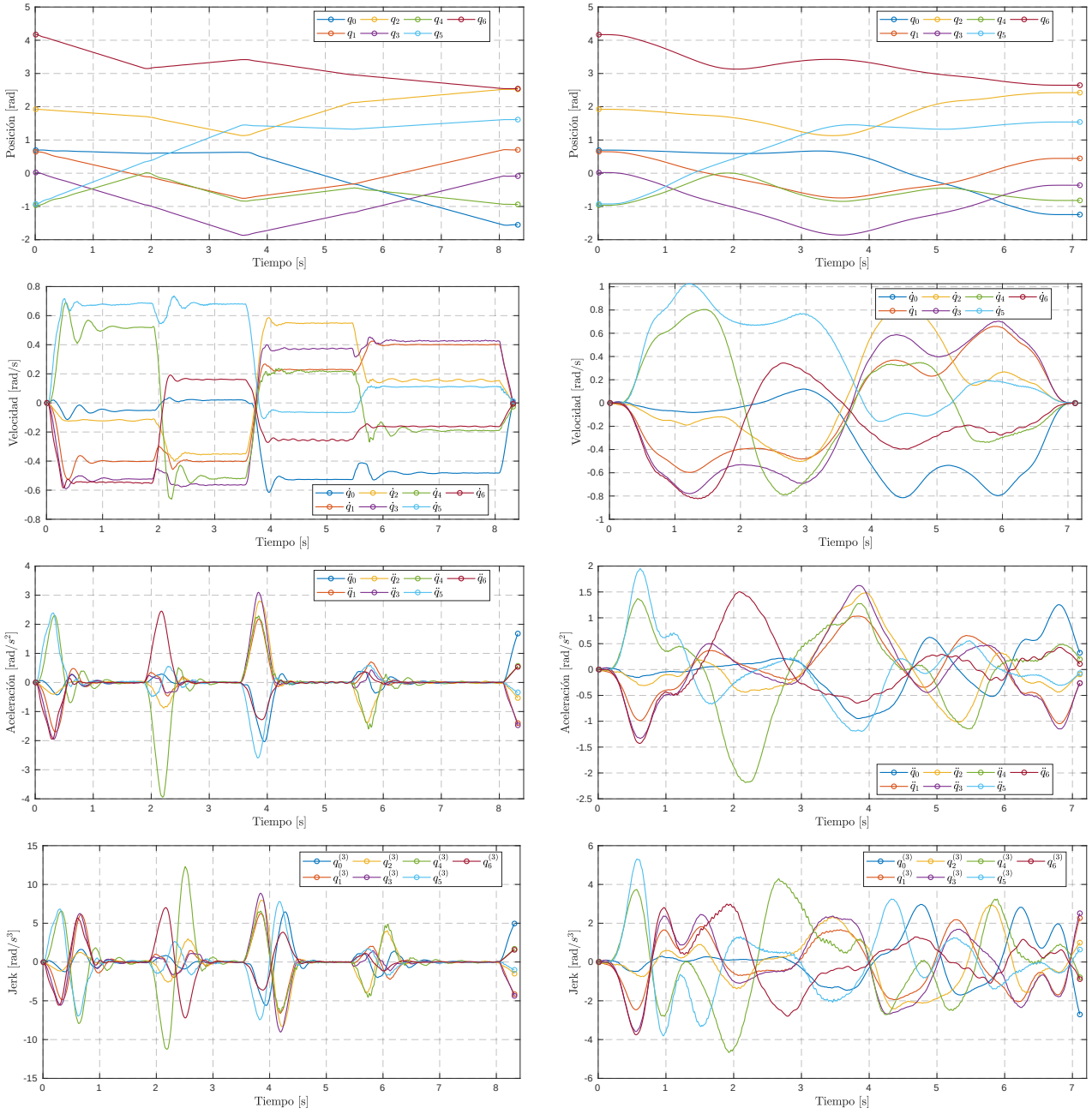


Figura 4.7: Comparación del movimiento por (a) cinemática directa con (b) la trayectoria obtenida para la *tarea A2* con $\alpha = 0.5$.

aceleración será un conjunto de ecuaciones lineales como se aprecia en la figura. Sin embargo, mencionar que existe una completa continuidad de valores entre dichas secciones. Siguiendo la lógica del spline, su sobreaceleración contendrá valores constantes donde los extremos garantizan los valores de 0. Este comportamiento también se refleja en la posición generada para $\alpha = 0.8$; la diferencia sustancial se debe al menor tiempo de ejecución que obliga a incrementar los valores a nivel velocidad, aceleración y sobreaceleración. No obstante, en ambos casos se cumple con el desplazamiento atraviesa los *knots* definidos.

Al comparar ambos escenarios, donde el primero se ha tomado un tiempo alrededor de 10.5 seg. en comparación del segundo caso de poco menos de 5 seg., se observa que los valores picos en velocidad, aceleración y sobreaceleración son mucho mayores en $\alpha = 0.2$ en comparación de $\alpha = 0.8$. Al nivel de aceleración, es muy sencillo identificar que al inicio y al final el efecto de las ecuaciones de séptimo grado permiten que se cumplan con las condiciones iniciales definidas. En este sentido, se han comparado dos casos extremos sobre la influencia del valor de α en la ecuación de costo. En Anexos, se expone los resultados de las trayectorias generadas por los demás valores de α que continúan con la lógica de que al incrementar α, el tiempo se reducirá y los niveles de vibración aumentarán.

El caso neutral de optimización se encuentra al emplear el valor de $\alpha = 0.5$ donde diversas publicaciones se han centrado en generar una única solución óptima ya que consideran el factor de tiempo y de sobreaceleración con igual ponderación. Este caso se ejecuta en el robot Sawyer siguiendo las posiciones de la *tarea A2* definidas en la Fig.4.5 y se compara con los resultados de la cinemática directa en la Fig.4.7.

En el desplazamiento de la posición empleando la cinemática directa se observa como cada articulación se desplaza de forma lineal entre los puntos de referencia introducidos. Se puede afirmar que su continuidad son muchos menores a la ejecutada por la optimización de trayectoria, donde ambos están cumpliendo por desplazarse por las referencias. A nivel de velocidad, se observa que el controlador por defecto del robot Sawyer

buscar la ejecución de una velocidad lineal entre sus articulaciones mientras que en el caso de la trayectoria sus valores son más oblicuos siendo los valores de ambos casos entre -1 y 1 *rad/s*. La diferencia sustancia y es donde se reúne las ventajas de la optimización de trayectoria es a nivel de aceleración. En este apartado se puede observar claramente los picos generados por el controlador de la cinemática directa; previamente, se ha explicado como dichos picos están relacionados con la vibración en el efector final del robot, así como un daño permanente en el robot. Los valores de picos de aceleración duplican los valores de la aceleración generados por la optimización de trayectoria, lo que representa una disminución de vibración. Similares efectos se interpretan en su siguiente derivada, en la sobreaceleración los valores máximos en el caso del controlar por defecto son más del doble del generado por el control de trayectoria.

Se debe recordar que el tiempo de ejecución empleando la optimización de trayectoria no es un factor se pueda definir con precisión pero si posible incrementarlo o disminuirlo en valor de la componente $alpha$; similar en el caso de la cinemática directa donde el tiempo puede incrementar o disminuir por medio de su configuración interna. En esta comparación, el tiempo del controlador propuesto es menor en 15 % del tiempo de la cinemática por lo que se esperaba que los picos en la aceleración fueran aún mayores y se incrementara los efectos de la vibración. Sin embargo, el efecto fue contrario, el desplazamiento fue más fluido y con menos efectos en la articulación final; esto demuestra que existe una clara ventaja en el uso del controlador basado en optimización de trayectoria.

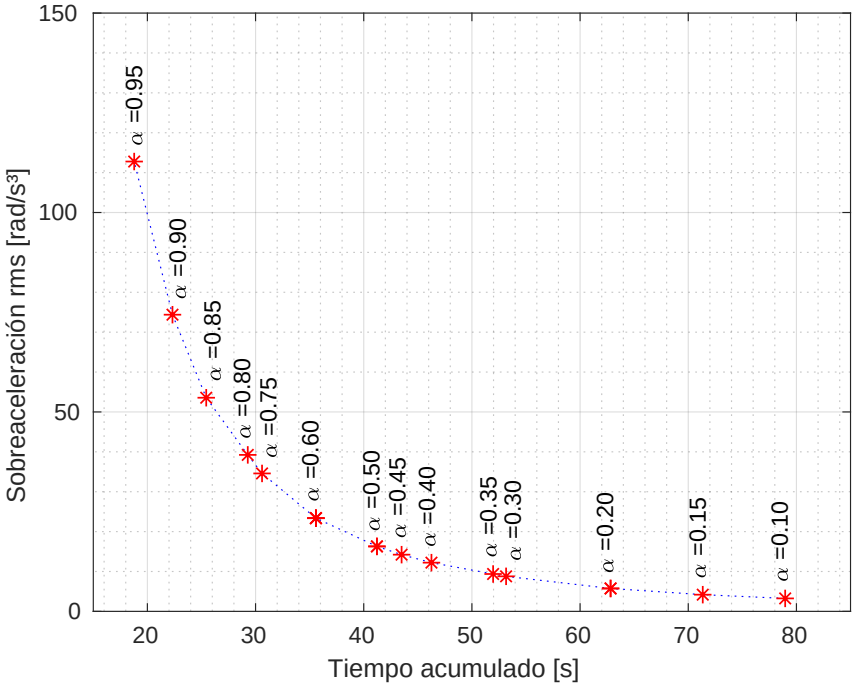
A fin de demostrar las ventajas de un factor modificable en la ecuación de costo, se ejecutaron todos los casos de α entre 0.5 y 0.95 con pasos de incremento de 0.05 y se resumieron los casos más relevantes en la Fig.4.8. En Fig.4.8a se muestra el comportamiento de dicho factor a nivel de simulación, asumiendo que la reproducción será exacta comparando el tiempo de ejecución frente a los valores de sobreaceleración rms, el cual es un indicador de vibración. Del gráfico se puede observar una tendencia entre dichos valores que permite estimar la influencia de α sobre el tiempo de ejecución. Este nivel de

simulación permitirá al usuario final identificar cual es la opción más adecuada a emplear de acuerdo a la aplicación. Esto representa una excelente ventaja sobre modelos descritos en [5, 6, 13, 19] donde la solución representa un único caso de optimización demostrando poca flexibilidad para el empleo del usuario.

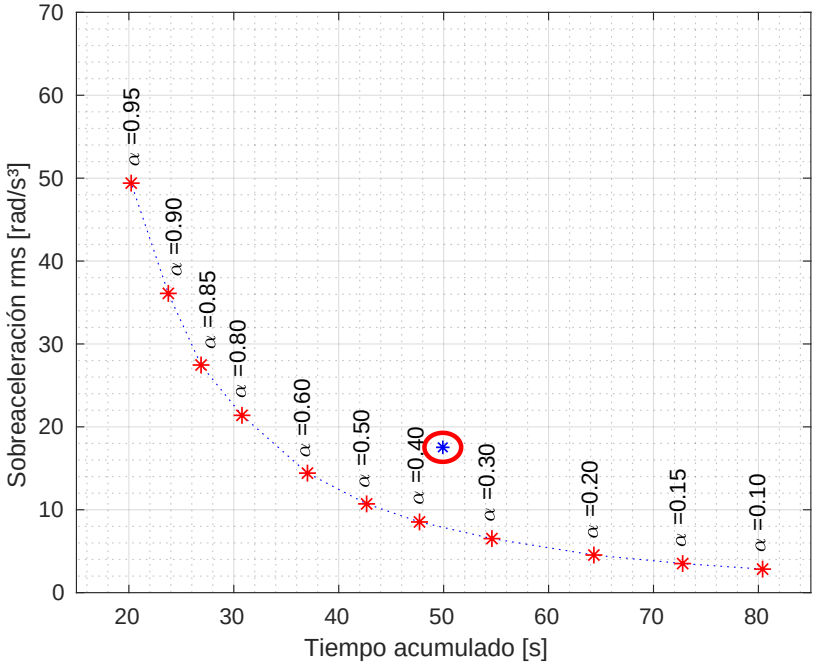
Previo a la optimización, analizando el caso de desplazamiento de tiempo mínimo teórico, los niveles de sobreaceleración media se encuentran sobre 5000 rad/s^3. Esto representa altos niveles de vibración en el desplazamiento, y a partir de pruebas previas realizadas en el robot, los niveles aceptables que se recomienda son en caso de una media menor a 400 rad/s^3. Luego de identificar que los niveles de vibración estén dentro de los límites, se ha procedido a ejecutar cada caso de α en el robot Sawyer a fin de comprobar si se cumple la tendencia y el correcto funcionamiento. Los resultados se resumen en la Fig.4.8b.

De los resultados obtenidos, los tiempos de ejecución mantienen un margen de reproducción entre 2 % a 3 % respecto a lo simulado. Por otro lado, los niveles de sobreaceleración no logran ser los mismos, pero se mantiene la proporción y la línea de tendencia identificada en la simulación. Esto se debe a los filtros de ventana aplicados para obtener los valores de ruido luego de aplicar el método de Euler, pues el entorno ROS solo permite obtener los valores de posición y velocidad.

Los resultados de la implementación permiten compararlo con la cinemática directa (representado como azul en Fig.4.8b). A partir de ella, podemos identificar para la misma cantidad de tiempo, el controlador por optimización de trayectoria genera menor niveles de vibración. En este sentido, se demuestra que se ha implementado con éxito el controlador con los resultados esperados en la disminución de picos de aceleración.



(a)



(b)

Figura 4.8: Comportamiento α para el tiempo acumulado y sobreaceleración para la tarea $A2$. (a) Resultados simulados del efecto de α. (b) Resultado de implementación donde el punto azul representa el caso de la cinemática directa a condiciones de fabricante.

4.2 Implementación del reconocimiento del objecto

Para la ejecución del reconocimiento del objeto, la cámara de interés envía la información por un canal a través del puerto usb y ésta es analizada desde la interface de python el cual recolecta la librería de OpenCV. Independiente de leer la imagen, lo que se tiene es una nueva imagen en una determinada frecuencia. La tarea propuesta $B1$ busca identificar el objeto de interés luego de aplicar el algoritmo descrito en la figura 3.2 en un escenario de simulación. En dicha tarea, se asume que la cámara se encontrará sobre un área uniforme que luego identificará la sección superficial del objeto.

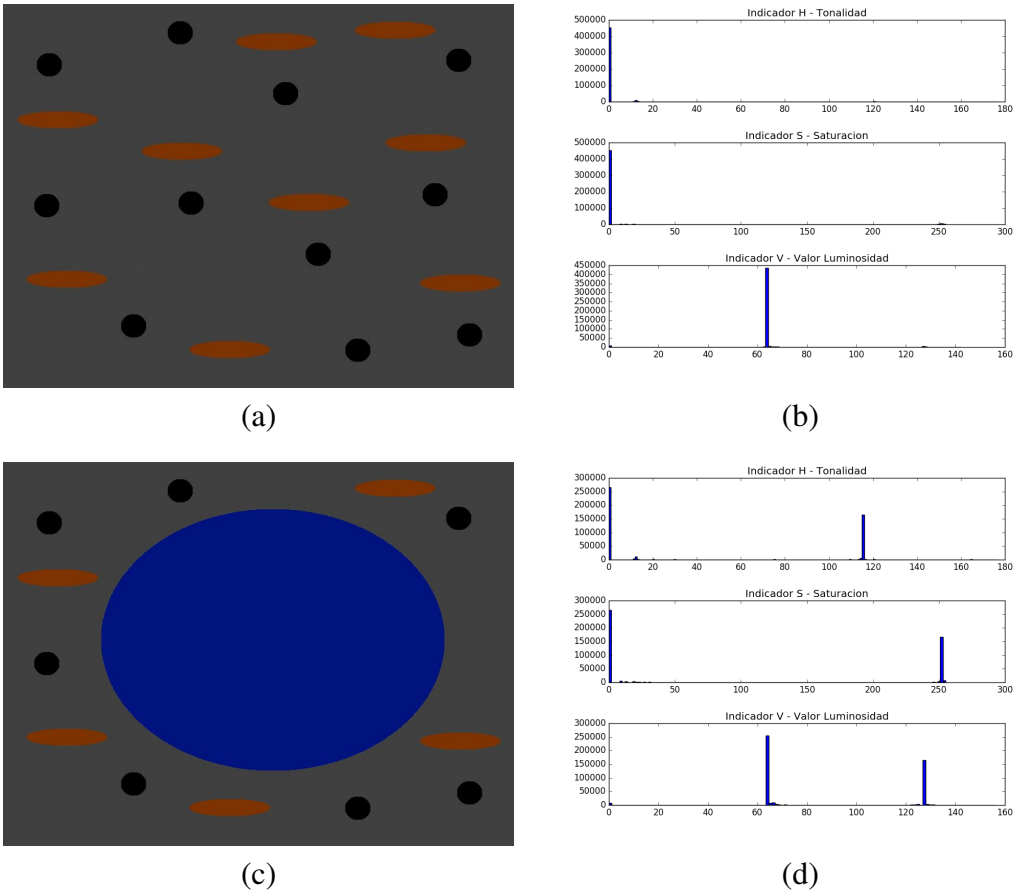
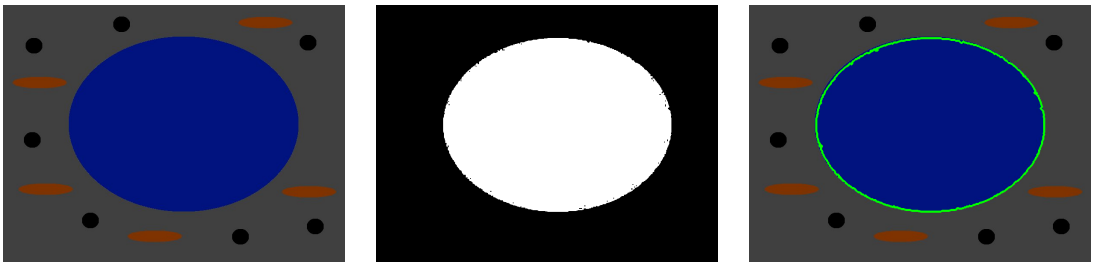


Figura 4.9: Resumen de la tarea *B1*. En ella se expone dos situaciones ideales en la que se busca identificación de la superficie del objeto a clasificar (color azul). A la derecha se muestra los valores de los indicadores HSV.

De acuerdo a la metodología propuesta, la imagen es transformada a un formato HSV para luego extraer la información de acuerdo a los parámetros que el usuario defina. Dichos parámetros, de acuerdo a la tarea, se establecen por conocimiento general sobre los colores; es decir, se tienen parámetros parcialmente definidos para grupos de colores como los primarios. Sin embargo, en este caso se propone realizar un análisis dimensional de los parámetros HSV. Para ello se propone dos situaciones. En la Fig.4.9a, se asume que la cámara se encuentra frontal al área sin ningún objeto con un área de color no uniforme del cual se desea, mientras que en 4.9c se encontraría, sobre dicha área, el objeto de interés que deseamos identificar su posición cartesiana.

Empleando los valores HSV, se puede obtener las intensidades para cada indicador Tonalidad (H), Saturación (S), Luminosidad (V). Por lo tanto a partir de las dos situaciones propuestas se generan los valores en las figuras 4.9b y 4.9d. A partir de ellas, se puede inferir que hay secciones de colores común y las que no, corresponde al objeto que se ha añadido al escenario; es decir, se identifica de forma rápida el rango HSV del objeto. Si bien este proceso resulta empírico para este análisis en la definición del parámetro de color, este proceso puede automatizarse.



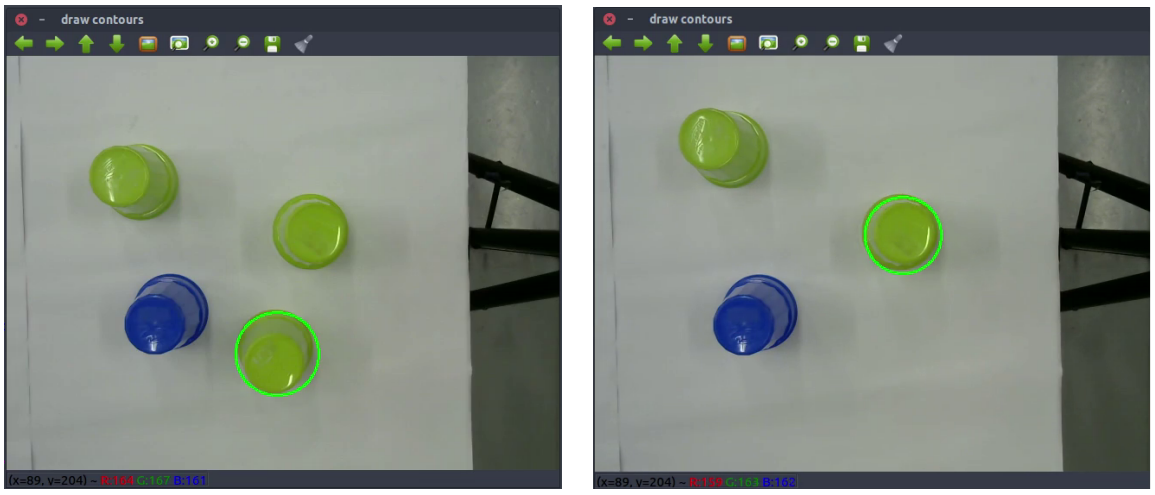
(a) (b) (c)

Figura 4.10: Representación de un caso ideal de reconocimiento del objeto. En (a) se tiene se observa la superficie del objeto (color azul). En (b) se filtra dicha sección de interés. En (c) se demuestra que se puede identificar la imagen de interés.

A partir de lo anterior, se procede con la reducción del ruido y con la definición de contornos. En la Fig. 4.10, se expone las 3 etapas principales de dicho algoritmo. En la Fig. 4.10a, se logra realizar la segmentación del objeto de interés con los parámetros

previamente definidos para crear el área potencial donde se encuentra el objeto reflejado en la Fig. 4.10b. Luego de ello se aplica la transforma de Hough, la cual nos permite identificar el punto cartesiano y el radio del objeto a clasificar. Lo mencionado se observa en la Fig.4.11.

Debido a que se conoce el radio de los objetos a clasificar, se programa un radio de rango $\pm 10\,\%$. A fin de evitar un falso positivo o ruido en la posición, se toman 10 muestras y se promedian para obtener el valor cartesiano de (x, y). El valor de la abscisa z será una constante pues se conoce la altura del objeto a clasificar y de la mesa. Por tanto, se construye un tópico en ROS que llama al grupo *geometry_msgs* y se construye un mensaje tipo *Point* que contiene los valores (x, y, z) con formato *float64*.



(a) (b)

Figura 4.11: Ejemplo de identificación del objeto a clasificar.

4.3 Implementación de monitoreo

Para la implementación del monitoreo se construye el mensaje utilizando Python. Se emplea la librería del fabricante para extraer la información actual del robot empleando una sentencia *while* con una frecuencia que se determina en la ejecución. Es posible disminuir dicho valor de frecuencia de acuerdo a la prioridad de la aplicación por parte del usuario sin dificultad alguna. Sin embargo, a altas frecuencias las condiciones de conexión a internet generan un retraso produciendo un efecto *buffer* generando que la información llegue a la nube pero se muestra con un retardo que se va incrementando con el número de datos acumulados. La información que se recolecta son los valores de posición y velocidad para las articulaciones *j0-j6*. Además de ello se extrae la información sobre el efector final. El efector final resume de forma competente el estado del elemento a desplazar, para ello se obtiene su posición respecto a la base del robot además de la velocidad. En la Lista 4.1 se muestra la configuración del mensaje.

```
limb = Limb()
j_p=limb.joint_angles(), j_v=limb.joint_velocities()
p_p=limb.endpoint_pose(), v_p=limb.endpoint_velocity()
mesage={
    joint_p = {
      'right_j0': j_p['right_j0'],
      'right_j1': j_p['right_j1'],
      'right_j2': j_p['right_j2'],
      'right_j3': j_p['right_j3'],
      'right_j4': j_p['right_j4'],
      'right_j5': j_p['right_j5'],
      'right_j6': j_p['right_j6']
      },
    joint_v = {
      'right_j0': j_v['right_j0'],
      'right_j1': j_v['right_j1'],
      'right_j2': j_v['right_j2'],
      'right_j3': j_v['right_j3'],
      'right_j4': j_v['right_j4'],
      'right_j5': j_v['right_j5'],
      'right_j6': j_v['right_j6']
      }
    }


```

Lista 4.1: Mensaje JSON de monitoreo.

Empleando la estructura de *NodeJS* definida en la sección 3.3.2 se procedió a enviar el mensaje definido en la lista 4.1. Se empleó una frecuencia de 10 Hz con una conexión de envío de información de 10 Mbps. En 4.12 se presenta la plataforma principal del servicio de IBM IOT, donde se configura el formato de la información que se debe aceptar.

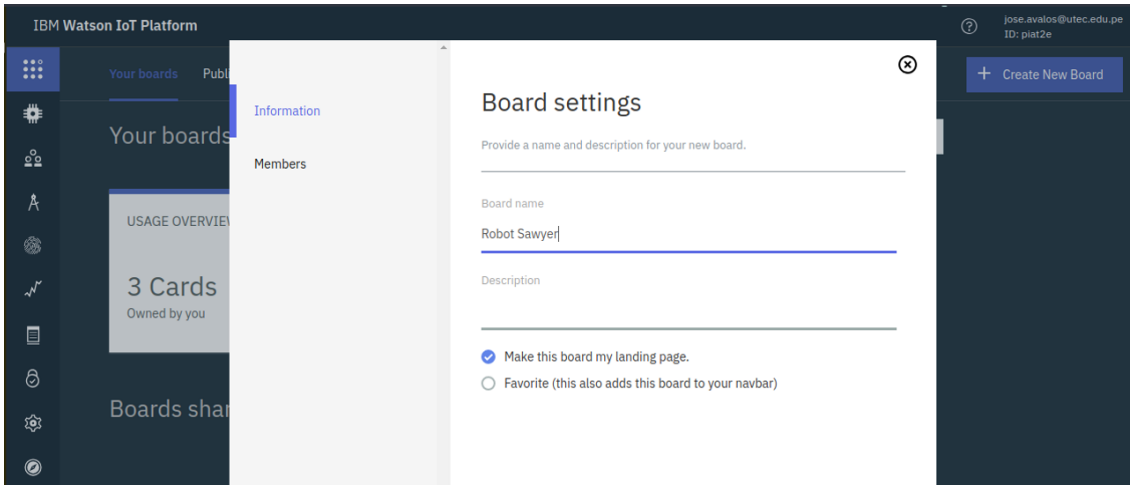
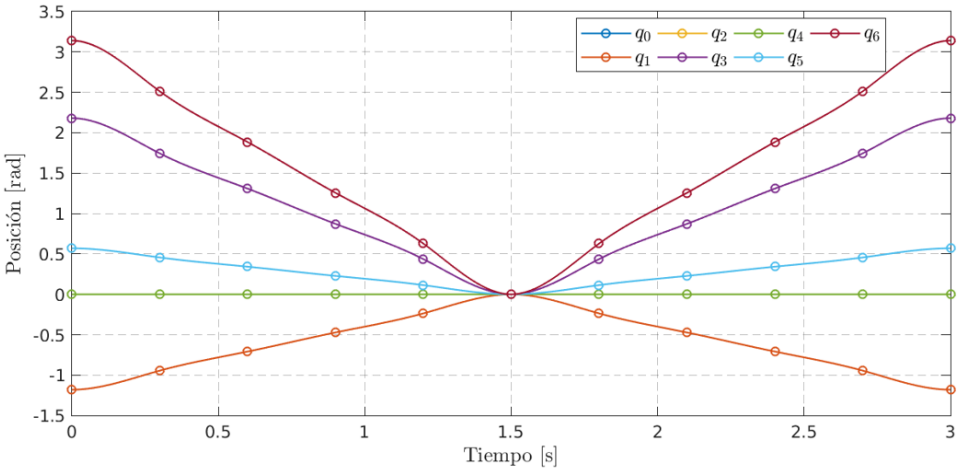


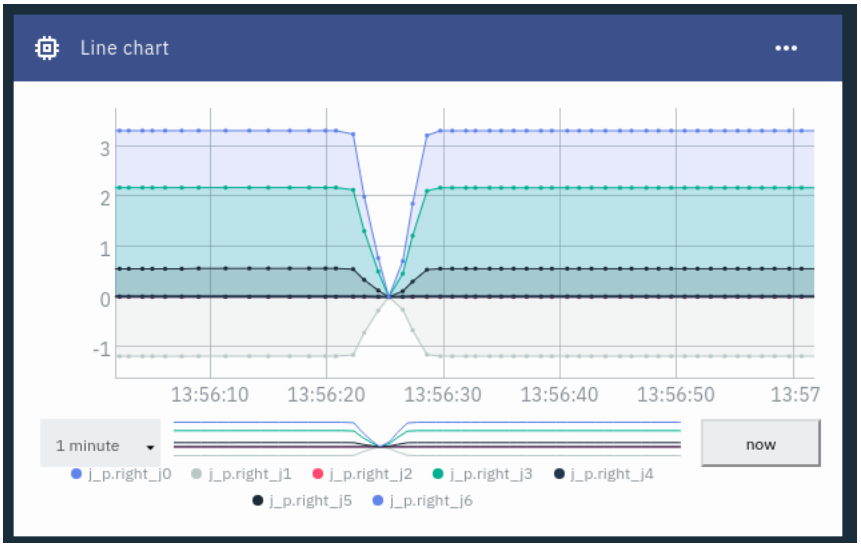
Figura 4.12: Plataforma IoT del servicio de IBM Cloud. Si bien es una plataforma de rápida generación de gráficos de monitoreo, solo puede ser accedido por un único usuario.

En esta plataforma se realizan las pruebas básicas que buscan demostrar si existe una continuidad en la información enviada sin retardos u otros efectos. En la Fig. 4.13 se muestra los resultados gráficos de la transmisión de ejecución de la *Tarea A1* a fin de comprobar si la información enviada puede ser interpretada por el usuario principal que empleará el monitoreo. Sin embargo, en caso de un usuario tercero que se dedicará íntegramente a ser informado del estado del robot no puede acceder a esta Plataforma debido a que las credenciales de este servicio están dirigidas a un único usuario maestro.

Por ello, resulta necesario la creación de una interfaz que permita representar la información del robot mediante un servicio web sin la necesidad de emplear las credenciales de mayor rango. Siguiendo la metodología, el entorno de *NodeJs* es empleado para albergar a nuestra web diseñada en base a HTML y *CSS*. Se empleará la información de



(a)

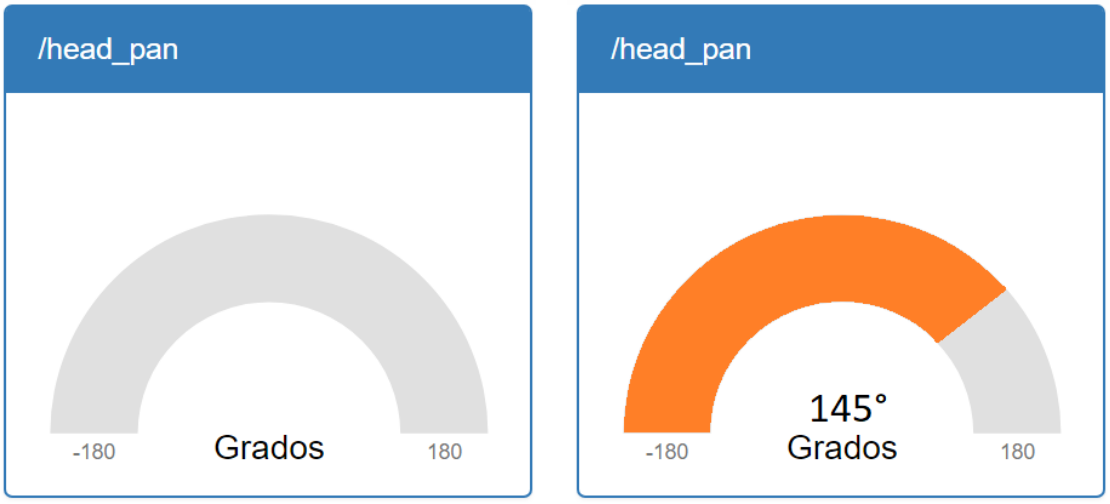


(b)

Figura 4.13: Ejemplo de monitoreo en tiempo real a 10 Hz. (a) Data registrada de manera local. (b) Data mostrada en la plataforma desarrollada.

obtenida en radianes por el entorno de ROS, la cual serán escalados a grados sexagesimales para ser mostrada de manera más didáctica. El contenedor visual se muestra en la Fig.4.14a, donde se crea un gráfico por articulación del robot. Cuando se reciba la información al emplear el servicio se mostrará el valor que se está reproduciendo en tiempo real, tal es el caso en la Fig.4.14b. La librería gráfica nos permite modificar los colores

del gráfico de acuerdo al rango de los valores. Esto nos permite demostrar alertas como un color rojo cuando la articulación se encuentre en valores extremos.



(a) (b)

Figura 4.14: Modelo de *Widget* en HTML para un tópico. (a) Gráfica en modelo inactivo. (b) Demostración de información en tiempo real.

Al emplear el servicio de IBM Cloud, una ventaja es la obtención de una dirección web certificada empleado el dominio *mybluemix.net*. Este dominio es configurado a través de la compilación local de la carpeta *NodeJS*, eligiendo *sawyerapp-3.mybluemix.net* mostrado en la Fig.4.15.

En la Fig.4.16, se observa una comparación entre el robot y la interfaz que observa un usuario externo en tiempo real.

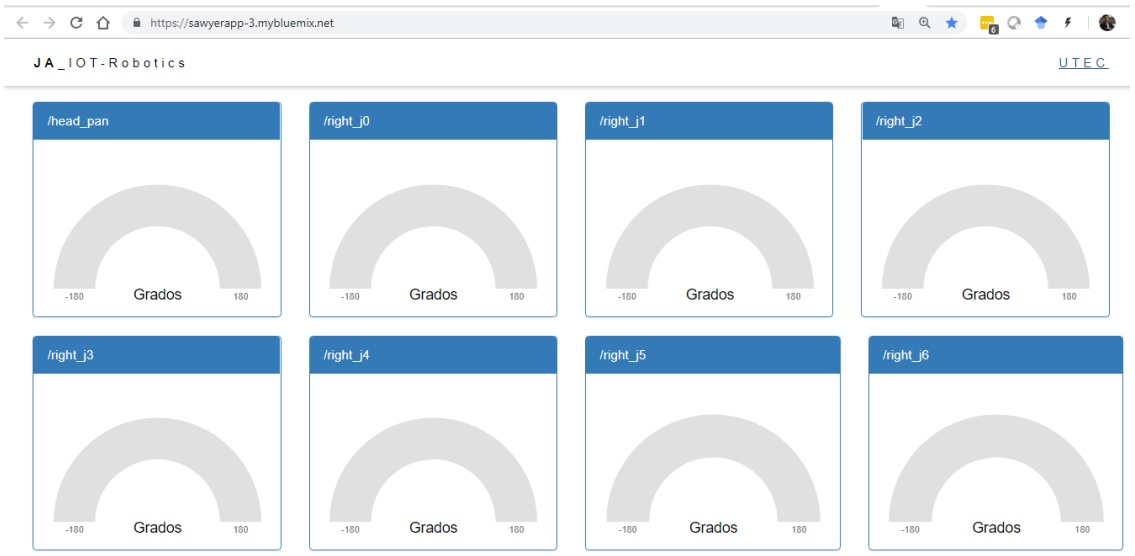


Figura 4.15: Modelo completo de la web IOT-UTEC para el monitoreo del robot Sawyer.

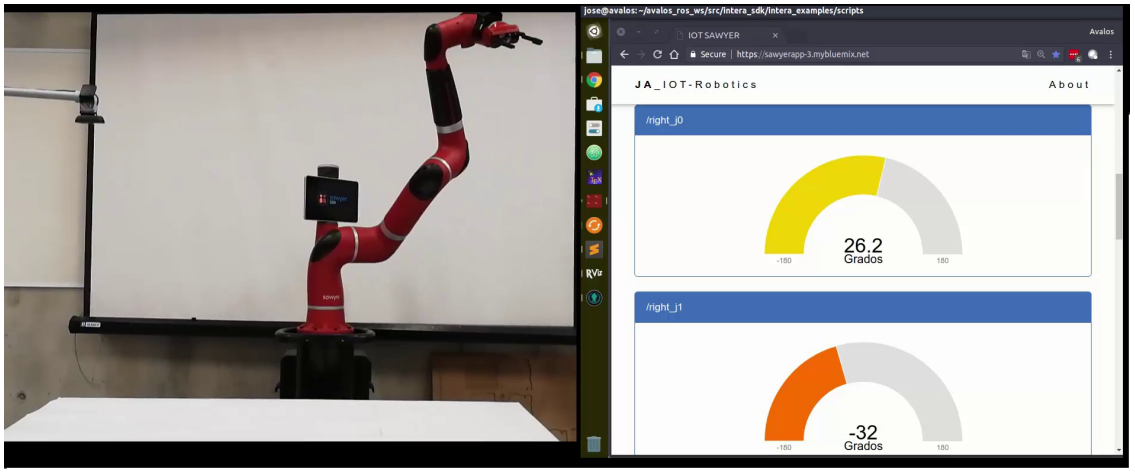


Figura 4.16: Modelo de la web IOT-UTEC en tiempo real.

4.4 Integración

En esta sección se busca describir la conexión del robot con el entorno de red a fin de lograr un eficiente traspaso de información entre las etapas descritas anteriormente. La conexión operativa se resume en la Fig.4.17. El robot Sawyer ha sido desarrollado para

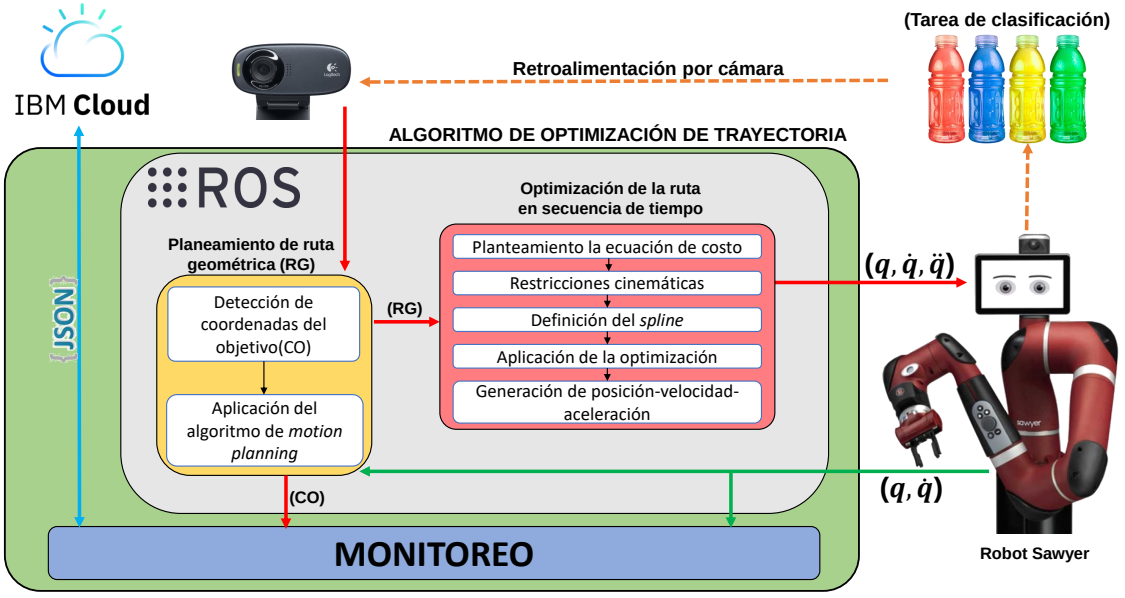


Figura 4.17: Representación de la metodología para la tarea de clasificación del robot Sawyer.

trabajar con el entorno ROS y nos permite obtener información de los tópicos por medio de una interface de terminal. Sin embargo, para hacer una tarea automática es necesario introducirla en un lenguaje de programación a fin de aplicar un conjunto de tareas repetitivas. Normalmente, ROS trabaja con lenguajes populares como *C++* y Python. En el caso del Sawyer, su fabricante, *Rethink Robotics*, ha desarrollado sus librerías principales en la interfaz de *Python*, es por ello, que se opta por emplear dicho lenguaje como el principal para las tareas ha realizar. Realizada la instalación nuestro entorno de trabajo se define en el directorio *avalos-ros-ws* donde el archivo *intera.h* resumirá la conexión con el robot, el cual es el paso principal para la ejecución de las tareas. La identificación del robot es con el ID $robot\_hostname = \text{“}utec.local\text{”}$.

4.4.1 Conexión al robot Sawyer

Los modos de conexión entre el robot y el *workstation* (computador o estación de trabajo) pueden ser de forma inalámbrica o alámbrica. El empleo de cada uno es dependiente de la aplicación a realizarse. Para esta tesis, se realiza una conexión directa con el robot a fin de eliminar cualquier retraso en el envío de información. Este punto es de vital importancia para identificar las condiciones de retraso de información ya que puede afectar la ejecución de la tarea.

4.4.2 Resultados de la tarea de clasificación

A partir de la demostración del cumplimiento de cada objetivo específico, en esta sección se expone el desarrollo de la tarea final. En el área de trabajo, expuesta en la Fig.4.18, se observan los 3 elementos necesarios para la ejecución: el robot Sawyer, una cámara sobre los elementos a clasificar y una conexión a internet.

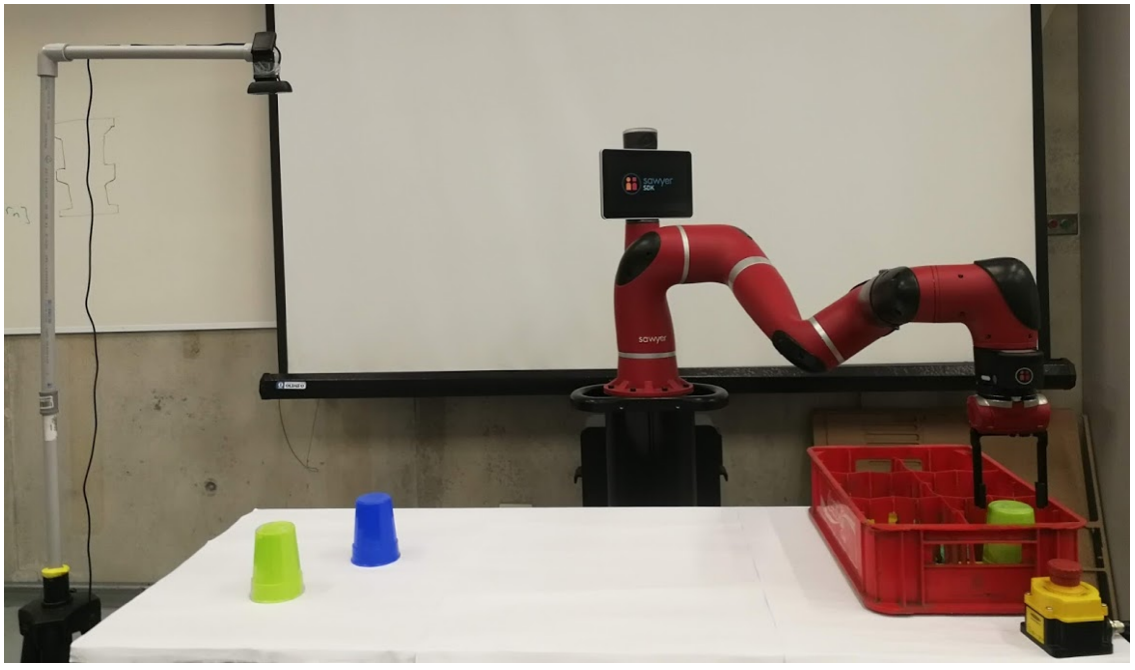


Figura 4.18: Área de trabajo para la tarea de clasificación del robot Sawyer.

La tarea incluyó que el robot clasifique, de forma autónoma, los dos objetos de color verde discriminando el objeto de color azul. Para analizar dichos resultados, se toma la media del tiempo en el desplazamiento desde que el *gripper* coge el objeto hasta que lo ubica en el contenedor. Se desarrolla la prueba con múltiples valores de α, los cuales son resumidos en la Tabla 4.2, donde se describe los valores del tiempo medio de desplazamiento, la sobreaceleración media y el valor pico máximo de aceleración. A partir de ello, se puede deducir cómo el valor de α influye en los valores de tiempo medio de forma inversamente proporcional; y directamente proporcional a la sobreaceleración. Además, se observa que los picos aceleración tiene una relación directa con el valor de α, el cual se encuentra dentro de los parámetros descritos en la Tabla 2.1. En este sentido, el usuario, modificando el valor de α, mantiene la libertad de escoger la solución más adecuada de acuerdo a la aplicación a realizar.

Tabla 4.1: Resultados empleando cinemática directa.

Tiempo medio $[s]$	Sobreaceleración media$[rad/s^3]$	Máx. aceleración $[rad/s^2]$
7.23	3.25	7.40

Tabla 4.2: Resultados de la aplicación de la optimización de trayectoria empleado la ecuación de costo propuesta.

α	Ecuación de costo - Avalos		
	Tiempo medio $[s]$	Sobreaceleración media $[rad/s^3]$	Máx. aceleración $[rad/s^2]$
0.10	13.00	0.40	0.70
0.20	10.45	0.78	1.05
0.30	9.15	1.16	1.42
0.40	7.70	1.19	1.90
0.50	7.04	2.62	2.49
0.60	5.95	4.31	3.38
0.70	5.05	6.99	4.62
0.80	4.01	13.63	7.58
0.90	3.55	20.84	10.05

De lo expuesto, se evidencia que los resultados de la implementación de la metodología propuesta cumplen con el alcance definido. A modo de concluir con este capítulo,

se compara los resultados con otras ecuaciones de costo definidos en el marco teórico. La ecuación de costo de esta tesis ,descrita en (3.1), es un nuevo modelo frente a los presentados en [3, 4]. Los resultado de aplicar la misma tarea de clasificación empleando las otras ecuaciones se resumen en la Tabla 4.3.

Tabla 4.3: Resultados de la aplicación de la optimización de trayectoria empleando la ecuación de costo en [3, 4].

α	Ecuación de costo Huang[4]			Ecuación de costo Gasparetto[3]		
	Tiempo [s]	Sobreaceleración $[rad/s^3]$	Máx. aceleración $[rad/s^2]$	Tiempo $[s]$	Sobreaceleración $[rad/s^3]$	Máx. aceleración $[rad/s^2]$
0.1	30.04	0.03	0.14	18.95	0.14	0.36
0.2	24.18	0.06	0.22	16.55	0.21	0.48
0.3	19.70	0.11	0.33	14.85	0.30	0.57
0.4	16.75	0.20	0.47	13.59	0.39	0.70
0.5	14.6	0.30	0.62	11.90	0.58	0.90
0.6	12.95	0.42	0.78	10.75	0.78	1.12
0.7	11.25	0.67	1.06	10.15	0.91	1.23
0.8	9.6	1.11	1.43	9.60	1.09	1.39
0.9	8.44	1.56	1.83	9.40	1.15	1.50

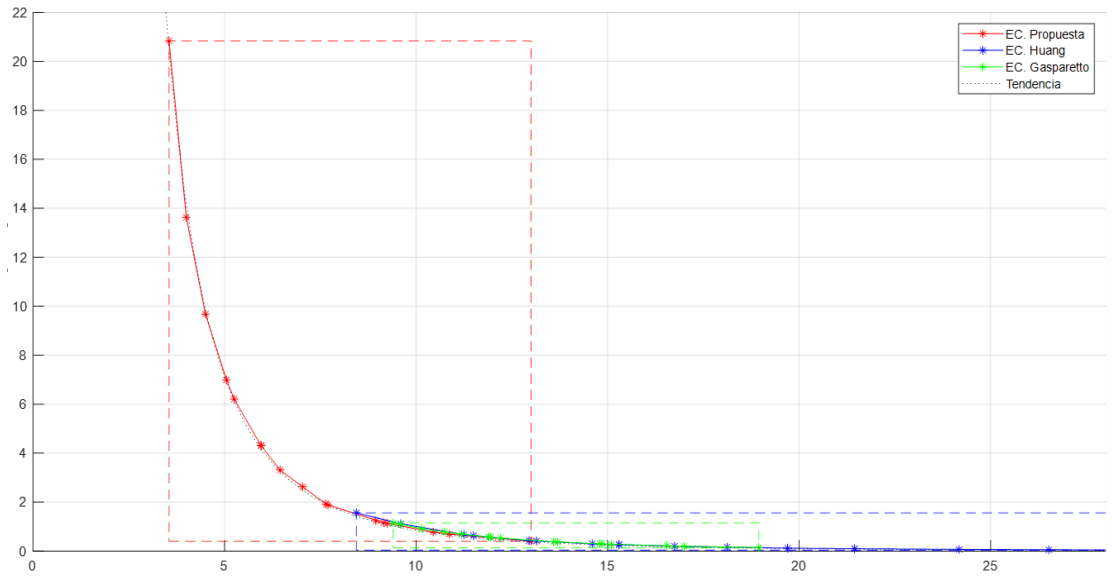


Figura 4.19: Resultados de comparación con otras ecuaciones de costo.

De los resultado, se observa un comportamiento proporcional entre las variables de similares características a la metodología desarrollada. Es decir, se evidencia la influencia del valor de α a fin de modificar el comportamiento de la trayectoria generada. A fin de lograr mostrar las diferencias, se reúnen todos los resultados en la Fig.4.19. En ella se puede observar las soluciones obtenidas y su comportamiento frente a un variación de α.

La ecuación de costo de [4], minimiza al máximo los valores de sobreaceleración; sin embargo, para ello incrementa el tiempo de desplazamiento haciéndolo impráctico para soluciones industriales cuyo tiempo de ejecución en su solución de mínima vibración representa 4 veces el tiempo empleando el por el controlador del fabricante en la Tabla 4.1. En el caso de [3], su mínimo tiempo no logra superar el tiempo de ejecución del controlador, por lo cual, no representa una ecuación viable que buscar brindar más opciones al momento de ejecutar la tarea.

En este sentido, ambas ecuaciones son poco recomendadas de introducir un factor de ponderación que busca brindar múltiples opciones sobre las condiciones de ejecución de las tareas. Esto se debe principalmente a que las ecuaciones planteadas se han concentrado principalmente en determinar la solución con mínima sobreaceleración sin considerar cómo el tiempo puede incrementar a niveles muy poco prácticos para un robot manipulador aplicado en industria. Por tanto, se demuestra que la metodología desarrollada en esta tesis resulta ser la más adecuada a fin de brindar flexibilidad al usuario sobre la ejecución de una trayectoria optimizada para un robot manipulador en industria.

CONCLUSIONES

En esta tesis se propuso un algoritmo de optimización de trayectoria, el cual fue implementado en el robot Sawyer de *Rethink Robotics*, mostrando la ejecución de una tarea de clasificación flexible con retroalimentación por cámara, además de una interfaz que emplea los servicios de la nube para un monitoreo remoto.

De la metodología descrita en el Capítulo 3, para el objetivo (a), se expusieron dos casos de desplazamiento para el robot Sawyer descritos en las Fig. 4.1 y Fig. 4.5. Mediante ambos escenarios, se expusieron las desventajas de los modelos actuales y las mejoras que puede ofrecer esta tesis, la cuál encuentra una solución óptima empleando la ecuación de costo planteada en 3.1. Dicha solución será una trayectoria, que se conforma de splines de séptimo grado en los nodos extremos y de spline cúbico en los nodos intermedios. Los resultados demuestran que la introducción de factor α en la ecuación de costo permite al usuario adaptar el tiempo y el grado de vibración en la aplicación siendo los margenes de trabajo 0.1-0.9 ya que el comportamiento de la curva. Así mismo, se realiza la comparación con las ecuaciones de costo expuestas en el estado del arte, donde nuestra propuesta es la que engloba mayor área de trabajo; es decir, permite al usuario poder variar las condiciones de trabajo lo cual fue la motivación de este trabajo en la industria.

Luego, se procedió a desarrollar los objetivos (b) y (c), donde la premisa de la tesis radica en la arquitectura propuesta; y que los métodos empleados en el procesamiento de imágenes y transmisión de data demuestran ser las más adecuadas para la tarea final definida en 4.4.2. Los resultados de implementar los 3 objetivos en paralelo permitió clasificar los objetos sin mayores contramedidas implementando un algoritmo repetitivo que permita realizar la tarea de forma indefinida y que puede ser adaptado a las condiciones particulares de múltiples escenarios.

Finalmente, partir de la metodología propuesta, y en base a los resultados obtenidos, se concluye que:

- El algoritmo propuesto muestra mejores resultados en comparación a la cinemática directa que emplea el fabricante, reduciendo los efectos de vibración teniendo como referencia los valores de sobreaceleración empleando el mismo tiempo de desplazamiento.

- La ecuación de costo implementada mantiene una mayor amplitud de tiempo y sobreaceleración en comparación a las descritas del marco teórico permitiendo al usuario modificar el comportamiento del robot entre tiempo y vibración de acuerdo a las circunstancias.

- Se implementó el monitoreo empleando una plataforma web con un envío de información de 10 Hz. En este caso, el método propuesto fue capaz de monitorear la posición cartesiana del objeto respecto al robot.

- El reconocimiento del objeto se realizó en tiempo real empleando *OpenCV*, permitiendo identificar la posición cartesiana del centro del objeto a clasificar.

- Se logró la implementación total del objetivo general propuesto de ejecutar una tarea de clasificación empleando el robot Sawyer con optimización de trayectoria. Asimismo, el algoritmo puede ser aplicado a cualquier robot manipulador a fin de reemplazar el controlador por defecto, garantizando menor vibración en su desplazamiento.

RECOMENDACIONES

Los resultados han sido exitosos dentro del alcance y objetivos propuestos al inicio de la tesis. Asimismo, la conclusión de esta tesis ha creado nuevas oportunidades de mejora que se enlistan a continuación:

- Los resultados previos han demostrado un tiempo de procesamiento aceptable con velocidad de procesamiento de 0.1 seg/knot por cada construcción de trayectoria empleando *Python*. A fin de reducir dicho tiempo se recomienda emplear un lenguaje de más bajo nivel como *C++*.
- A fin de no depender del costo computacional de la PC de ejecución se recomienda evaluar desplazar el algoritmo de optimización a un servicio de nube que permita emplear una computadora virtual de gran poder de procesamiento.
- A nivel de actuadores se demostró que el robot Sawyer es capaz de desplazar un único objeto por cada tarea de clasificación. A fin de extender estas capacidades se recomienda reemplazar el *gripper* por un modelo de mayor capacidad a fin de efectuar tareas de mayor alcance.
- La plataforma web puede transmitir la información en tiempo real. Sin embargo, dicha información podría resultar susceptible dañando la privacidad del usuario; por lo cual, se recomienda agregar un *login* a fin de evitar infiltración de terceros.

Se espera que esta tesis motive nuevas líneas de investigación gracias a la aparición de nuevos recursos de procesamiento y a la creciente necesidad de la robótica en las tareas de rubro industrial.

REFERENCIAS BIBLIOGRÁFICAS

[1] E. G. Moreno, *Automatización de procesos industriales*. Alfaomega Valencia, 2001.

[2] Z.-H. Luo, "Direct strain feedback control of flexible robot arms: new theoretical and experimental results," *IEEE Transactions on Automatic Control*, vol. 38, no. 11, pp. 1610–1622, 1993.

[3] A. Gasparetto and V. Zanotto, "Optimal trajectory planning for industrial robots," *Advances in Engineering Software*, vol. 41, no. 4, pp. 548–556, 2010.

[4] J. Huang, P. Hu, K. Wu, and M. Zeng, "Optimal time-jerk trajectory planning for industrial robots," *Mechanism and Machine Theory*, vol. 121, pp. 530–544, 2018.

[5] F. Rubio, C. Llopis-Albert, F. Valero, and J. L. Suñer, "Industrial robot efficient trajectory generation without collision through the evolution of the optimal trajectory," *Robotics and Autonomous Systems*, vol. 86, pp. 106–112, 2016.

[6] A. Gasparetto and V. Zanotto, "A new method for smooth trajectory planning of robot manipulators," *Mechanism and machine theory*, vol. 42, no. 4, pp. 455–471, 2007.

[7] S. Kucuk, "Optimal trajectory generation algorithm for serial and parallel manipulators," *Robotics and Computer-Integrated Manufacturing*, vol. 48, pp. 219–232, 2017.

[8] J. Y. Luh, M. W. Walker, and R. P. Paul, "On-line computational scheme for mechanical manipulators," *Journal of Dynamic Systems, Measurement, and Control*, vol. 102, no. 2, pp. 69–76, 1980.

[9] B. Siciliano, "Kinematic control of redundant robot manipulators: A tutorial," *Journal of intelligent and robotic systems*, vol. 3, no. 3, pp. 201–212, 1990.

[10] A. Visioli, “Trajectory planning of robot manipulators by using algebraic and trigonometric splines,” *Robotica*, vol. 18, no. 6, p. 611–631, 2000.

[11] A. Piazzi and A. Visioli, “Global minimum-jerk trajectory planning of robot manipulators,” *IEEE transactions on industrial electronics*, vol. 47, no. 1, pp. 140–149, 2000.

[12] E. Dyllong and A. Visioli, “Planning and real-time modifications of a trajectory using spline techniques,” *Robotica*, vol. 21, no. 5, p. 475, 2003.

[13] T. Chettibi, H. Lehtihet, M. Haddad, and S. Hanchi, “Minimum cost trajectory planning for industrial robots,” *European Journal of Mechanics-A/Solids*, vol. 23, no. 4, pp. 703–715, 2004.

[14] M. Egerstedt and C. F. Martin, “Optimal trajectory planning and smoothing splines,” *Automatica*, vol. 37, no. 7, pp. 1057–1064, 2001.

[15] J. Han, Y. Jiang, X. Tian, F. Chen, C. Lu, and L. Xia, “A local smoothing interpolation method for short line segments to realize continuous motion of tool axis acceleration,” *The International Journal of Advanced Manufacturing Technology*, vol. 95, no. 5, pp. 1729–1742, Mar 2018. [Online]. Available: https://doi.org/10.1007/s00170-017-1264-3

[16] S. Macfarlane and E. A. Croft, “Jerk-bounded manipulator trajectory planning: design for real-time applications,” *IEEE Transactions on Robotics and Automation*, vol. 19, no. 1, pp. 42–52, Feb 2003.

[17] C. Lin, P. Chang, and J. Luh, “Formulation and optimization of cubic polynomial joint trajectories for industrial robots,” *IEEE Transactions on automatic control*, vol. 28, no. 12, pp. 1066–1074, 1983.

[18] W. Aribowo and K. Terashima, "Cubic spline trajectory planning and vibration suppression of semiconductor wafer transfer robot arm." *IJAT*, vol. 8, no. 2, pp. 265–274, 2014.

[19] B. Tondu and S. A. Bazaz, "The three-cubic method: an optimal online robot joint trajectory generator under velocity, acceleration, and wandering constraints," *The International Journal of Robotics Research*, vol. 18, no. 9, pp. 893–901, 1999.

[20] J. Z. Kolter and A. Y. Ng, "Task-space trajectories via cubic spline optimization," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 1675–1682.

[21] A. Gasparetto and V. Zanotto, "A technique for time-jerk optimal planning of robot trajectories," *Robotics and Computer-Integrated Manufacturing*, vol. 24, no. 3, pp. 415–426, 2008.

[22] B. Demeulenaere, G. Pipeleers, J. De Caigny, J. Swevers, J. De Schutter, and L. Vandenberghe, "Optimal splines for rigid motion systems: a convex programming framework," *Journal of Mechanical Design*, vol. 131, no. 10, p. 101004, 2009.

[23] A. Ess, B. Leibe, K. Schindler, and L. Van Gool, "Moving obstacle detection in highly dynamic scenes," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 56–63.

[24] M. Quigley, S. Batra, S. Gould, E. Klingbeil, Q. Le, A. Wellman, and A. Y. Ng, "High-accuracy 3d sensing for mobile manipulation: Improving object detection and door opening," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 2816–2822.

[25] C. Theis, I. Iossifidis, and A. Steinhage, "Image processing methods for interactive robot control," in *Robot and Human Interactive Communication, 2001. Proceedings. 10th IEEE International Workshop on*. IEEE, 2001, pp. 424–429.

[26] J. R. Sanchez-Lopez, A. Marin-Hernandez, and E. R. Palacios-Hernandez, "Visual detection, tracking and pose estimation of a robotic arm end effector," in *Proc. Robotics Summer Meeting (Veracruz, Mexico, 27–28 June 2011)*, 2011, pp. 41–8.

[27] P. Vijayalaxmi, R. Putta, G. Shinde, and P. Lohani, "Object detection using image processing for an industrial robot," in *Proceedings of Fifth IRAJ International Conference, 15th September*, 2013.

[28] J. Avalos and O. E. Ramos, "Flexible visually-driven object classification using the baxter robot," in *Proc. Electrical Engineering and Computing (INTERCON) 2017 IEEE XXIV Int. Conf. Electronics*, Aug. 2017, pp. 1–4.

[29] K. Bekris, R. Shome, A. Krontiris, and A. Dobson, "Cloud automation: Precomputing roadmaps for flexible manipulation," *IEEE Robotics & Automation Magazine*, vol. 22, no. 2, pp. 41–50, 2015.

[30] B. Qureshi and A. Koubâa, "Five traits of performance enhancement using cloud robotics: A survey," *Procedia Computer Science*, vol. 37, pp. 220–227, 2014.

[31] L. Joseph and J. Cacace, *Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System*. Packt Publishing Ltd, 2018.

[32] A. Koubaa, M. Alajlan, and B. Qureshi, "Roslink: Bridging ros with the internet-of-things for cloud robotics," in *Robot Operating System (ROS)*. Springer, 2017, pp. 265–283.

[33] T. Bray, "The javascript object notation (json) data interchange format," 2017.

[34] T. C. Smith, "An investigation on a mobile robot in a ros enabled cloud robotics environment," 2016.

[35] J. Wallén, *The history of the industrial robot*. Linköping University Electronic Press, 2008.

[36] I. Standard, "8373: 1994," *Manipulating industrial robots–Vocabulary*, 1994.

[37] E. Guizzo, "Sawyer: Rethink robotics unveils new robot," *Spectrum, IEEE*, 2015.

[38] M. Kajko-Mattsson, S. Forssander, and G. Andersson, "Software problem reporting and resolution process at abb robotics ab: state of practice," *Journal of Software Maintenance: Research and Practice*, vol. 12, no. 5, pp. 255–285, 2000.

[39] J. Braumann and S. Brell-Cokcan, "Parametric robot control: integrated cad/cam for architectural design," 2011.

[40] M. Quigley, B. Gerky, and W. Smart, "Programming robots with ros, a practical introduction to the robot operating system (first. o'reilly media, inc)," 2015.

[41] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. .º'Reilly Media, Inc.", 2008.

[42] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-s—a publish/subscribe protocol for wireless sensor networks," in *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*. IEEE, 2008, pp. 791–798.

[43] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[44] H. Liu, X. Lai, and W. Wu, "Time-optimal and jerk-continuous trajectory planning for robot manipulators with kinematic constraints," *Robotics and Computer-Integrated Manufacturing*, vol. 29, no. 2, pp. 309–317, 2013.

[45] IBM. (2018) IBM Cloud watson iot platform. [Online]. Available: https://www.ibm.com/cloud/watson-iot-platform

[46] J. Avalos and O. E. Ramos, "Optimal time-jerk trajectory generation for robot manipulators," in *2018 IEEE 2nd Colombian Conference on Robotics and Automation (CCRA)*. IEEE, 2018, pp. 1–6.

ANEXOS

ANEXO A

CÓDIGOS

Toda la información mostrada representa el código principal en esta tesis. De igual manera se deja el repositorio con la información de intalación para el robot Sawyer en: https://github.com/javalosv/avalos_intera

1.1 Código Robot- Objetivo (a)

```
# ***************************************************************************
# Copyright (c) 2018 Jose Avalos
# ***************************************************************************
#!/usr/bin/env python
from __future__ import division
import argparse
import rospy
import intera_interface
import intera_external_devices
import time
import numpy as np
from math import *
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy
from scipy.optimize import minimize
from intera_core_msgs.msg import JointCommand
from scipy import interpolate
from scipy.interpolate import InterpolatedUnivariateSpline
from scipy.interpolate import interp1d
from geometry_msgs.msg import (
  PoseStamped,
  Pose,
  Point,
  Quaternion,
)
from std_msgs.msg import Header
from sensor_msgs.msg import JointState
from intera_core_msgs.srv import (
  SolvePositionIK,
  SolvePositionIKRequest,
)

```

```
# Funcion para obtener la integral de una funcion empleando metodo del poligono
def get_area_cuadrada(_vector,_f):
  _v=np.power(_vector,2)
  k=np.sum(_v)-0.5*(_v[0]+_v[-1])
  area=k/float(_f)
  return area

# Funcion para guardar la data y luego analizar
def save_matrix(_j,_name,_f):
  file2write=open(_name,'w')
  l=len(_j[0][:])
  time=np.linspace(0, (l-1)/float(_f), num=l)
  for n in range(l):
    file2write.write(str(time[n])+' , '+str(_j[0][n])+' , '+str(_j[1][n])+' , '+str(_j[2][n])+' , '+str(_j[3][
        n])+' , '+ str(_j[4][n])+' , '+str(_j[5][n])+' , '+str(_j[6][n])+'\n')
  file2write.close()
  print "save data en",_name
  return True

# Funcion que nos permitira construir la trayectoria de una articulacionteniendo la informacion de
      los tiempos de los nodos
def path_simple_cub(_point,_time,_f):
  x=_time
  a=_point
  f=_f
  FPO=0.0
  FPN=0.0

  n=len(a)-1;
  l=np.zeros(n+1, dtype=np.float_)
  u=np.zeros(n, dtype=np.float_)
  z=np.zeros(n+1, dtype=np.float_)
  h=np.zeros(n, dtype=np.float_)
  alfa=np.zeros(n+1, dtype=np.float_)
  c=np.zeros(n+1, dtype=np.float_)
  b=np.zeros(n, dtype=np.float_)
  d=np.zeros(n, dtype=np.float_)

  for i in range(n):
    h[i]=x[i+1]-x[i]
  sA = np.zeros(shape=(n+1,n+1), dtype=np.float_)
  for i in range(n-1) :
    for j in range(n-1) :
      if i is j:
        sA[i+1][i+1]=2*(h[i]+h[i+1])
        sA[i+1][i]=h[i]
        sA[i][i+1]=h[i]
  sA[0][0]=2*h[0]
  sA[-1][-1]=2*h[-1]
  sA[-1][-2]=h[-1]
  sA[-2][-1]=h[-1]

  sb = np.zeros(shape=(n+1,1), dtype=np.float_)

  for i in range(1,n) :
    sb[i]=(3.0*(a[i+1]-a[i])/h[i]) - (3.0*(a[i]-a[i-1])/h[i-1])

  sb[0]=((3.0*(a[1]-a[0]))/h[0])-3.0*FPO
```

```
sb[-1]=3.0*FPN-(3.0*(a[n]-a[n-1])/h[n-1])

_b=np.arange(n, dtype=np.float_)
_c=np.linalg.solve(sA, sb)
_d=np.arange(n, dtype=np.float_)

for j in reversed(range(n)):
  _b[j]=((x|[j+1]-a[j])/h[j] )-(h[j]*(_c[j+1]+2*_c[j])/3.0)
  _d[j]=(_c[j+1]-_c[j])/(3.0*h[j])

# Graphic
t_out=np.linspace(x[0], x[-1], int((x[-1]-x[0])*f)+1)
tl=len(t_out)
p =np.zeros(tl, dtype=np.float_)
v =np.zeros(tl, dtype=np.float_)
ac =np.zeros(tl, dtype=np.float_)
y =np.zeros(tl, dtype=np.float_)

for i in range(n):
  for j in range(tl):
    if(t_out[j]>=x[i] and t_out[j]<x[i+1]):
      p[j]=( a[i]+_b[i]*(t_out[j]-x[i])+_c[i]*(t_out[j]-x[i])**2+_d[i]*(t_out[j]-x[i])**3)
      v[j]=_b[i]+2*_c[i]*(t_out[j]-x[i])+3*_d[i]*(t_out[j]-x[i])**2
      ac[j]=2*_c[i]+6*_d[i]*(t_out[j]-x[i])
      y[j]=6*_d[i]
k1=int((h[0])*f)
k2=int((h[-1])*f+1)

s_v=v[k1]
s_ac=ac[k1]
s_y=y[k1]
h_0=h[0]
h_e=h[-1]

# Spline 7 grade begin
a00 =(10*s_v)/h_0**6 - (2*s_ac)/h_0**5 + s_y/(6*h_0**4) + (20*(a[0]-a[1]))/h_0**7
a01 =(13*s_ac)/(2*h_0**4) - (34*s_v)/h_0**5 - s_y/(2*h_0**3) - (70*(a[0]-a[1]))/h_0**6
a02 =(39*s_v)/h_0**4 - (7*s_ac)/h_0**3 + s_y/(2*h_0**2) + (84*(a[0]-a[1]))/h_0**5
a03 =(5*s_ac)/(2*h_0**2) - (15*s_v)/h_0**3 - s_y/(6*h_0) - (35*(a[0]-a[1]))/h_0**4
for j in range(k1):
   p[j]=a00*(j*0.01)**7+a01*(j*0.01)**6+a02*(j*0.01)**5+a03*(j*0.01)**4+a[0]
   v[j]=7*a00*(j*0.01)**6+6*a01*(j*0.01)**5+5*a02*(j*0.01)**4+4*a03*(j*0.01)**3
   ac[j]=42*a00*(j*0.01)**5+30*a01*(j*0.01)**4+20*a02*(j*0.01)**3+12*a03*(j*0.01)**2
   y[j]=210*a00*(j*0.01)**4+120*a01*(j*0.01)**3+60*a02*(j*0.01)**2+24*a03*(j*0.01)**1

#  Spline 7 grade end
a7=a[-2]
a6=v[-k2]
a5= ac[-k2]/2
a4= -y[-k2]/6
h_e=h[-1]
p_e=a[-1]

ak0 = (2*(2*a5 + 6*a4*h_e))/h_e**5 - a4/h_e**4 - (10*(3*a4*h_e**2 + 2*a5*h_e + a6))/h_e
    **6 + (20*(a4*h_e**3 + a5*h_e**2 + a6*h_e + a7 - p_e))/h_e**7
ak1 = (3*a4)/h_e**3 - (13*(2*a5 + 6*a4*h_e))/(2*h_e**4) + (34*(3*a4*h_e**2 + 2*a5*h_e + a6
    ))/h_e**5 - (70*(a4*h_e**3 + a5*h_e**2 + a6*h_e + a7 - p_e))/h_e**6
```

```
146   ak2 = (7*(2*a5 + 6*a4*h_e))/h_e**3 - (3*a4)/h_e**2 - (39*(3*a4*h_e**2 + 2*a5*h_e + a6))/
          h_e**4 + (84*(a4*h_e**3 + a5*h_e**2 + a6*h_e + a7 - p_e))/h_e**5
147   ak3 = a4/h_e - (5*(2*a5 + 6*a4*h_e))/(2*h_e**2) + (15*(3*a4*h_e**2 + 2*a5*h_e + a6))/h_e
          **3 - (35*(a4*h_e**3 + a5*h_e**2 + a6*h_e + a7 - p_e))/h_e**4
148
149   for j in range(k2):
150     p[-k2+j]=ak0*(j*0.01)**7+ak1*(j*0.01)**6+ak2*(j*0.01)**5+ak3*(j*0.01)**4+a4*(j*0.01)
          **3+a5*(j*0.01)**2+a6*(j*0.01)+a7
151     v[-k2+j]=7*ak0*(j*0.01)**6+6*ak1*(j*0.01)**5+5*ak2*(j*0.01)**4+4*ak3*(j*0.01)**3+3*
          a4*(j*0.01)**2+2*a5*(j*0.01)+a6
152     ac[-k2+j]=42*ak0*(j*0.01)**5+30*ak1*(j*0.01)**4+20*ak2*(j*0.01)**3+12*ak3*(j*0.01)
          **2+6*a4*(j*0.01)+2*a5
153     y[-k2+j]=210*ak0*(j*0.01)**4+120*ak1*(j*0.01)**3+60*ak2*(j*0.01)**2+24*ak3*(j*0.01)
          **1+6*a4
154
155   return p,v,ac,y,tl
156
157 # Funcion que nos permitira obtener el jerk de acuerdo a los informacion de los tiempos de los nodos
        para una articulacion. En esta funcion se reduce el costo cumputacional ya que la Clase
        optimizacion llamara recurrente esta funcion hasta obtener la optima. Hallada la optima, se
        construye el trayectoria con la funcion path_simple_cub()
158 def path_simple_cub_get_jerk(_point,_time,_f):
159   x=_time
160   a=_point
161   f=_f
162   FPO=0.0
163   FPN=0.0
164
165   n=len(a)-1;
166   l=np.zeros(n+1, dtype=np.float_)
167   u=np.zeros(n, dtype=np.float_)
168   z=np.zeros(n+1, dtype=np.float_)
169   h=np.zeros(n, dtype=np.float_)
170   alfa=np.zeros(n+1, dtype=np.float_)
171   c=np.zeros(n+1, dtype=np.float_)
172   b=np.zeros(n, dtype=np.float_)
173   d=np.zeros(n, dtype=np.float_)
174
175   for i in range(n):
176     h[i]=x[i+1]-x[i]
177   sA = np.zeros(shape=(n+1,n+1), dtype=np.float_)
178   for i in range(n-1) :
179     for j in range(n-1) :
180       if i is j:
181         sA[i+1][i+1]=2*(h[i]+h[i+1])
182         sA[i+1][i]=h[i]
183         sA[i][i+1]=h[i]
184   sA[0][0]=2*h[0]
185   sA[-1][-1]=2*h[-1]
186   sA[-1][-2]=h[-1]
187   sA[-2][-1]=h[-1]
188   sb = np.zeros(shape=(n+1,1), dtype=np.float_)
189   for i in range(1,n) :
190     sb[i]=(3.0*(a[i+1]-a[i])/h[i]) - (3.0*(a[i]-a[i-1])/h[i-1])
191
192   sb[0]=((3.0*(a[1]-a[0]))/h[0])-3.0*FPO
193   sb[-1]=3.0*FPN-(3.0*(a[n]-a[n-1])/h[n-1])
194   _b=np.arange(n, dtype=np.float_)
```

```
  _c=np.linalg.solve(sA, sb)
  _d=np.arange(n, dtype=np.float_)
  for j in reversed(range(n)):
    _b[j]=((a[j+1]-a[j])/h[j] )-(h[j]*(_c[j+1]+2*_c[j])/3.0)
    _d[j]=(_c[j+1]-_c[j])/(3.0*h[j])

  t_out=np.linspace(x[0], x[-1], int((x[-1]-x[0])*f)+1)
  tl=len(t_out)
  y =np.zeros(tl, dtype=np.float_)
  for i in range(n):
    for j in range(tl):
      if(t_out[j]>=x[i] and t_out[j]<x[i+1]):
        y[j]=6*_d[i]
  k1=int((h[0])*f)
  k2=int((h[-1])*f+1)
  s_v=_b[1]
  s_ac=2*_c[1]
  s_y=6*_d[1]
  h_e=h[-1]
  # Spline 7 grade begin
  a00 =(10*s_v)/h[0]**6 - (2*s_ac)/h[0]**5 + s_y/(6*h[0]**4) + (20*(a[0]-a[1]))/h[0]**7
  a01 =(13*s_ac)/(2*h[0]**4) - (34*s_v)/h[0]**5 - s_y/(2*h[0]**3) - (70*(a[0]-a[1]))/h[0]**6
  a02 =(39*s_v)/h[0]**4 - (7*s_ac)/h[0]**3 + s_y/(2*h[0]**2) + (84*(a[0]-a[1]))/h[0]**5
  a03 =(5*s_ac)/(2*h[0]**2) - (15*s_v)/h[0]**3 - s_y/(6*h[0]) - (35*(a[0]-a[1]))/h[0]**4
  for j in range(k1):
     y[j]=210*a00*(j*0.01)**4+120*a01*(j*0.01)**3+60*a02*(j*0.01)**2+24*a03*(j*0.01)**1
  #  Spline 7 grade end
  a7=a[-2]
  a6=_b[-1]
  a5= _c[i]
  a4= -_d[i]
  h_e=h[-1]
  p_e=a[-1]
  ak0 = (2*(2*a5 + 6*a4*h_e))/h_e**5 - a4/h_e**4 - (10*(3*a4*h_e**2 + 2*a5*h_e + a6))/h_e
      **6 + (20*(a4*h_e**3 + a5*h_e**2 + a6*h_e + a7 - p_e))/h_e**7
  ak1 = (3*a4)/h_e**3 - (13*(2*a5 + 6*a4*h_e))/(2*h_e**4) + (34*(3*a4*h_e**2 + 2*a5*h_e + a6
      ))/h_e**5 - (70*(a4*h_e**3 + a5*h_e**2 + a6*h_e + a7 - p_e))/h_e**6
  ak2 = (7*(2*a5 + 6*a4*h_e))/h_e**3 - (3*a4)/h_e**2 - (39*(3*a4*h_e**2 + 2*a5*h_e + a6))/
      h_e**4 + (84*(a4*h_e**3 + a5*h_e**2 + a6*h_e + a7 - p_e))/h_e**5
  ak3 = a4/h_e - (5*(2*a5 + 6*a4*h_e))/(2*h_e**2) + (15*(3*a4*h_e**2 + 2*a5*h_e + a6))/h_e
      **3 - (35*(a4*h_e**3 + a5*h_e**2 + a6*h_e + a7 - p_e))/h_e**4
  for j in range(k2):
     y[-k2+j]=210*ak0*(j*0.01)**4+120*ak1*(j*0.01)**3+60*ak2*(j*0.01)**2+24*ak3*(j*0.01)
       **1+6*a4
  return y

# Funcion que nos permite construir una matriz de todas las articulaciones luego de determinar la
      solucion optima
def generate_path_cub(_points,_time,_f,p=True):
  [q0,v0,a0,y0,l]=path_simple_cub(_points[0],_time,_f)
  [q1,v1,a1,y1,l]=path_simple_cub(_points[1],_time,_f)
  [q2,v2,a2,y2,l]=path_simple_cub(_points[2],_time,_f)
  [q3,v3,a3,y3,l]=path_simple_cub(_points[3],_time,_f)
  [q4,v4,a4,y4,l]=path_simple_cub(_points[4],_time,_f)
  [q5,v5,a5,y5,l]=path_simple_cub(_points[5],_time,_f)
  [q6,v6,a6,y6,l]=path_simple_cub(_points[6],_time,_f)
  q= np.array([q0,q1,q2,q3,q4,q5,q6])
```

```
246   v= np.array([v0,v1,v2,v3,v4,v5,v6])
247   a= np.array([a0,a1,a2,a3,a4,a5,a6])
248   y= np.array([y0,y1,y2,y3,y4,y5,y6])
249   ext = l
250   if(p):
251     print "Knots en posicion generados.",ext
252   return q,v,a,y
253
254 # Se determina el minimo tiempo para ejecutar el movimiento considerando los limites del
        fabricante
255 def min_time(_q):
256   vel_lim=[1.74, 1.328, 1.957, 1.957, 3.485, 3.485, 4.545]
257   v_factor=0.9 # Es un concepto de seguridad para las pruebas.Es decir, se utiliza el 90 % del limite
258   N=len(vel_lim)
259   k=len(_q[0])
260   t_min=np.zeros(k, dtype=np.float_)
261   t_tmp=np.zeros(N, dtype=np.float_)
262   for i in range (k-1):
263     for j in range (N):
264       t_tmp[j]= abs((_q[j,i+1]-_q[j,i])/((v_factor)*vel_lim[j]))
265     w=np.amax(t_tmp)# Se asume t[0]=0
266     t_min[i+1]=w+t_min[i]
267   return t_min, t_min[-1]
268
269 class Opt_avalos():
270   def __init__(self,_q,_f,_alfa):
271     self.q=_q
272     self.f=_f/5 # Para optimizar reducimos la frecuencia de trabajo
273     self.alfa=_alfa
274     [self.min_time,self.t_rec]=min_time(self.q)
275     self.l=len(self.min_time)-1
276     self.delta_t=np.ones(self.l)
277     self.tmp=np.zeros(self.l)
278     bnds = ()
279     for i in range(self.l):
280       bnds+=((1,None),)
281       self.delta_t[i]=self.min_time[i+1]-self.min_time[i]
282     x0 = np.ones(self.l)
283     print "Working in solution alfa=",str(_alfa)
284     # Aqui se desarrola el porblema de oprimizacion
285     myfactr = 5e-4
286     self.res = minimize(self.costo, x0,method='L-BFGS-B', bounds=bnds ,options={'ftol' : myfactr
          ,'disp': False, 'eps': 1e-8})
287     self.tmp=self.res.x*self.delta_t
288     self.v_time=np.append([0],self.tmp.cumsum())
289   def costo(self,k):
290     k=k*self.delta_t
291     # Funcion Costo
292     [jk,ext]=self.value_sum_jerk(self.q,np.append([0],k.cumsum()),self.f)
293     value_jk=sqrt(jk) # Avalos
294     #value_jk=jk #Pareto
295     #value_jk=jk #Huang
296     value_t=round(7*(ext/float(self.f)),2) #Avalos
297     #value_t=round((ext/float(self.f)),2) #Huang
298     ecu=self.alfa*value_t+(1-self.alfa)*value_jk
299     return ecu
300   def value_sum_jerk(self,_points,_time,_f):
301     jk0=path_simple_cub_get_jerk(_points[0],_time,_f)
```

```
    jk1=path_simple_cub_get_jerk(_points[1],_time,_f)
    jk2=path_simple_cub_get_jerk(_points[2],_time,_f)
    jk3=path_simple_cub_get_jerk(_points[3],_time,_f)
    jk4=path_simple_cub_get_jerk(_points[4],_time,_f)
    jk5=path_simple_cub_get_jerk(_points[5],_time,_f)
    jk6=path_simple_cub_get_jerk(_points[6],_time,_f)
    ext= len(jk0)
    a_jk0=get_area_cuadrada(jk0,_f)
    a_jk1=get_area_cuadrada(jk1,_f)
    a_jk2=get_area_cuadrada(jk2,_f)
    a_jk3=get_area_cuadrada(jk3,_f)
    a_jk4=get_area_cuadrada(jk4,_f)
    a_jk5=get_area_cuadrada(jk5,_f)
    a_jk6=get_area_cuadrada(jk6,_f)
    #value_jk=a_jk0+a_jk1+a_jk2+a_jk3+a_jk4+a_jk5+a_jk6
    value_jk=sqrt(a_jk0)+sqrt(a_jk1)+sqrt(a_jk2)+sqrt(a_jk3)+sqrt(a_jk4)+sqrt(a_jk5)+sqrt(a_jk6)
    # SQRT
    # ind=sqrt(value_jk/float(6.0))
    return value_jk,ext
  def full_time(self):
    return self.v_time

# Esta clase nos permite grabar la data del robot y exportarlo en un txt
class Data():
  def __init__(self):
    self.write=False
    rospy.Subscriber("/robot/joint_states", JointState, self.talker)
    print("Init bridge")
    rate = rospy.Rate(100) # 10hz
  def talker(self,data):
    if(data.name[0]=="head_pan"):
      self.position=data.position[1:7]# extrae solo 7
      if(self.write):
        _file=open(self.file,"a")
        _file.write(str(data.position[1])+","+str(data.position[2])+","+str(data.position[3])+","+str(data.
        position[4])+","+str(data.position[5])+","+str(data.position[6])+","+str(data.position[7])+","+str
        (data.velocity[1])+","+str(data.velocity[2])+","+str(data.velocity[3])+","+str(data.velocity[4])+"
        ,"+str(data.velocity[5])+","+str(data.velocity[6])+","+str(data.velocity[7])+"\n")
        _file.close()
  def actual_joint_position(self):
    return self.position
  def writeon(self,_text):
    self.write=True
    self.file=_text
    file=open(_text,"w")
    file.close()
    return True
  def writeoff(self):
    self.write=False
    return True
```

1.2 Código Imágenes- Objetivo (b)

```
# ********************************************************************************
# Copyright (c) 2018 Jose Avalos
# ********************************************************************************
#!/usr/bin/env python
import numpy as np
import cv2
import time as t
from scipy.cluster.vq import vq, kmeans
from matplotlib import pyplot as plt

def talker():
    # Definir el topico de acuerdo a las caracteristicas de la camra
    cv_image = cv2.imread('topico_camara',1)
    rospy.init_node('talker', anonymous=True)
    # Obtenemos los datos de formato de la cmara
    height, width , depth  = cv_image.shape
    hsv = cv2.cvtColor(cv_image, cv2.COLOR_BGR2HSV)
    thresh = 0
    obj_color = 0
    # De acuerdo al color de clasificacion se definen estos parametros
    if obj_color == 0:
        low__h  = 10
        low__s  = 0
        low__v  = 100
        high__h = 30
        high__s = 200
        high__v = 250
        hue, sat, val = hsv[:,:,0], hsv[:,:,1], hsv[:,:,2]
        thresh = cv2.inRange(hsv, np.array([low__h, low__s, low__v]), np.array([high__h, high__s,
        high__v]))
        plt.figure(figsize=(10,8))
        plt.subplot(311)                    #Posicion del grafico
        plt.subplots_adjust(hspace=.5)
        plt.title("H")
        plt.hist(np.ndarray.flatten(hue), bins=180)
        plt.subplot(312)                    #Posicion del grafico
        plt.title("S")
        plt.hist(np.ndarray.flatten(sat), bins=128)
        plt.subplot(313)                    #Posicion del grafico
        plt.title("V")
        plt.hist(np.ndarray.flatten(val), bins=128)
        plt.show()
        #Apertura morfologica (eliminar objetos pequenos del primer plano)
        thresh = cv2.erode(thresh, np.ones((2,2), np.uint8), iterations=1)
        thresh = cv2.dilate(thresh, np.ones((2,2), np.uint8), iterations=1)
        #Cierre morfologico (rellene pequenos agujeros en primer plano)
        thresh = cv2.dilate(thresh, np.ones((2,2), np.uint8), iterations=1)
        thresh = cv2.erode(thresh, np.ones((2,2), np.uint8), iterations=1)

        ret,thresh = cv2.threshold(thresh,255,255,0)
        contours= cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
        cv2.drawContours(cv_image, contours[1], -1, (0,255,0), 3)
        numobj = len(contours) # Numero de objetos encontrados
        if numobj > 0:
```

```
54            momentos = cv2.moments(contours[0])
55            if momentos['m00']>40:
56                cx = int(momentos['m10']/momentos['m00'])
57                cy = int(momentos['m01']/momentos['m00'])
58                # Parametros de acuerdo al area de trabajo.
59                xb = -(cy - (height/2))*.0025*.428 + .578
60                yb = -(cx - (width/2))*.0025*.428 + .40  +0.06
61                print("xb: ",xb,"/n")
62                print("yb: ",yb,"/n")
63            obj_found = True
64        else:
65            print "No puedo encontrar ningun objeto."
66        cv2.imshow("Thresh", thresh)
67    if __name__ == '__main__':
68        try:
69            talker()
70        except KeyboardInterrupt:
71            pass
```

1.3 Código Cloud - Objetivo (c)

```
# ******************************************************************************
# Copyright (c) 2018 Jose Avalos
# ******************************************************************************
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
import getopt
import time
import sys
import uuid
import threading
import time
import signal
import os
import random
import ibmiotf.device
import math

import rospy
import argparse
from intera_interface import Limb
import rospy
from std_msgs.msg import String


running_status = True

def my_on_publish_callback():
    print("Confirmed received by WatsonIoTP")

def sendIOT():
    #rospy.init_node('avalos_limb_py')

    while not rospy.is_shutdown():
        limb = Limb()
        j_p=limb.joint_angles()
        j_v=limb.joint_velocities()
        p_p=limb.endpoint_pose()
        v_p=limb.endpoint_velocity()

        if running_status:
            joint_p = {'right_j0': j_p['right_j0'],
                    'right_j1': j_p['right_j1'],
                    'right_j2': j_p['right_j2'],
                    'right_j3': j_p['right_j3'],
                    'right_j4': j_p['right_j4'],
                    'right_j5': j_p['right_j5'],
                    'right_j6': j_p['right_j6']}
            joint_v = {'right_j0': j_v['right_j0'],
                    'right_j1': j_v['right_j1'],
                    'right_j2': j_v['right_j2'],
                    'right_j3': j_v['right_j3'],
                    'right_j4': j_v['right_j4'],
                    'right_j5': j_v['right_j5'],
                    'right_j6': j_v['right_j6']}
```

```
            success = device_client.publishEvent(
                "joint_angle",
                "json",
                {'j_p': joint_p,'j_v': joint_v},
                qos=0,
                on_publish=my_on_publish_callback())
            #print data
            time.sleep(0.1)
            if not success:
                print("Not connected to WatsonIoTP")


if __name__ == "__main__":
    try:
        device_file = "device.conf"
        device_options = ibmiotf.device.ParseConfigFile(device_file)
        device_client = ibmiotf.device.Client(device_options)
        device_client.connect()
        rospy.init_node('listener', anonymous=True)
        sendIOT()
    except Exception as e:
        print("Caught exception connecting device:  %s"  % str(e))
        sys.exit()
```

1.4 Código Implementación

```
# ********************************************************************************
# Copyright (c) 2018 Jose Avalos
# ********************************************************************************
#!/usr/bin/env python
import rospy
import argparse
from intera_interface import Limb
from intera_interface import CHECK_VERSION
from intera_core_msgs.msg import JointCommand
import numpy as np
import scipy as sp
import math
import time
from intera_avalos import *
import intera_interface
import intera_external_devices
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from math import pi
from std_msgs.msg import String
from moveit_commander.conversions import pose_to_list
from trajectory_msgs.msg import JointTrajectoryPoint
from moveit_msgs.msg import RobotState
# Pasos para iniciar la programacion
# rosrun intera_interface enable_robot.py -e
# roslaunch sawyer_moveit_config sawyer_moveit.launch electric_gripper:=true
# python opt_caso2.py joint_states:=/robot/joint_states
# rostopic echo /robot/limb/right/endpoint_state/pose
# Python 2.9
def main():
    try:
        moveit_commander.roscpp_initialize(sys.argv)
        rospy.init_node('avalos_limb_py',anonymous=True)
        robot = moveit_commander.RobotCommander()
        scene = moveit_commander.PlanningSceneInterface()
        group_name = 'right_arm'
        group = moveit_commander.MoveGroupCommander(group_name)
        #frecuency for Sawyer robot
        f=100
        alfa=0.9
        rate = rospy.Rate(f)
        # Anadimos el gripper
        gripper = intera_interface.Gripper('right_gripper')
        gripper.calibrate()
        gripper.open()
        # Cargamos el modelo del robot
        moveit_robot_state = RobotState()
        joint_state = JointState()
        joint_state.header = Header()
        joint_state.header.stamp = rospy.Time.now()
```

```
55      joint_state.name = ['right_j0', 'right_j1', 'right_j2', 'right_j3', 'right_j4', 'right_j5', 'right_j6']
56      #Denifimos los topicos
57      pub = rospy.Publisher('/robot/limb/right/joint_command', JointCommand, queue_size=10)
58      limb = Limb()
59      # Desplazamos a la posicion inicial
60      limb.move_to_neutral()
61      limb.move_to_joint_positions({"right_j6":2})
62      group.clear_pose_targets()
63      group.set_start_state_to_current_state()
64      pose_goal = geometry_msgs.msg.Pose()
65      # Orientation durante la tarea
66      pose_goal.orientation.x = -1
67      pose_goal.orientation.y = 0.0
68      pose_goal.orientation.z = 0.0
69      pose_goal.orientation.w = 0.0
70      q0=np.array([])
71      q1=np.array([])
72      q2=np.array([])
73      q3=np.array([])
74      q4=np.array([])
75      q5=np.array([])
76      q6=np.array([])
77      # Cartesian position - Carga '01'
78      pose_goal.position.x = # Denifinir puntos de acuerdo a la logica
79      pose_goal.position.y = # Denifinir puntos de acuerdo a la logica
80      pose_goal.position.z = # Denifinir puntos de acuerdo a la logica
81      group.set_pose_target(pose_goal)
82      carga1=group.plan().joint_trajectory.points
83      n=len(carga1)
84      joint_state.position = [carga1[-1].positions[0], carga1[-1].positions[1], carga1[-1].positions
        [2], carga1[-1].positions[3],carga1[-1].positions[4], carga1[-1].positions[5], carga1[-1].
        positions[6]]
85      moveit_robot_state.joint_state = joint_state
86      group.set_start_state(moveit_robot_state)
87
88
89      tmp=np.array([])
90
91      if(n>8):
92          tmp=np.append(tmp,0)
93          k=int(math.sqrt(n)+2)
94          r=int((n-1)/float(k))
95          for i in range(k):
96              print i
97              tmp=np.append(tmp,int(r*(i+1)-1))
98          tmp=np.append(tmp,n-1)
99      else:
100         for i in range(n):
101             print i
102             tmp=np.append(tmp,i)
103
104     print "tmp:", tmp
105     for i in range(len(tmp)):
106         q0=np.append(q0, carga1[int(tmp[i])].positions[0])
107         q1=np.append(q1, carga1[int(tmp[i])].positions[1])
108         q2=np.append(q2, carga1[int(tmp[i])].positions[2])
109         q3=np.append(q3, carga1[int(tmp[i])].positions[3])
110         q4=np.append(q4, carga1[int(tmp[i])].positions[4])
```

```
111             q5=np.append(q5, carga1[int(tmp[i])].positions[5])
112             q6=np.append(q6, carga1[int(tmp[i])].positions[6])
113 
114         q=np.array([q0,q1,q2,q3,q4,q5,q6])
115         print "q001",q0
116         m_time, t_min_tiempo=min_time(q)
117         start = time.time()
118         opt=Opt_avalos(q,f,0.9)
119         v_time=opt.full_time()
120         j_1,v_1,a_1,jk_1=generate_path_cub(q,v_time,f)
121         ext_1=len(j_1[0,:])
122         end = time.time()
123         print('Process Time:', end-start)
124         v_jk=sqrt(np.mean(np.square(jk_1)))
125         print("Opt Time:",v_time)
126         print("Min Time:",m_time)
127         print('Optimizacion:',opt.result())
128         max_v=np.amax(np.absolute(v_1))
129         max_ac=np.amax(np.absolute(a_1))
130         max_jk=np.amax(np.absolute(jk_1))
131         print "Max Velo:",max_v
132         print "Max Acel:",max_ac
133         print "Max Jerk:",max_jk
134         #raw_input('Iniciar_CT_execute?')
135         my_msg=JointCommand()       #Se construye el mensaje para el Sawyer
136         # POSITION_MODE
137         my_msg.mode=4
138         my_msg.names=["right_j0","right_j1","right_j2","right_j3","right_j4","right_j5"]#,"right_j6"]
139         print("Control por trayectoria iniciado.")
140         joint_state.position = [carga1[-1].positions[0], carga1[-1].positions[1], carga1[-1].positions
            [2], carga1[-1].positions[3],carga1[-1].positions[4], carga1[-1].positions[5], carga1[-1].
            positions[6]]
141         moveit_robot_state.joint_state = joint_state
142         group.set_start_state(moveit_robot_state)
143         raw_input('Iniciar_ejecucion?')
144         for n in xrange(ext_1): # Se adiciona los valores para este caso la ultima aritulacion de giro del
            gripper no se usa
145             my_msg.position=[j_1[0][n],j_1[1][n],j_1[2][n],j_1[3][n],j_1[4][n],j_1[5][n]]#,j_1[6][n]]
146             my_msg.velocity=[v_1[0][n],v_1[1][n],v_1[2][n],v_1[3][n],v_1[4][n],v_1[5][n]]
147             my_msg.acceleration=[a_1[0][n],a_1[1][n],a_1[2][n],a_1[3][n],a_1[4][n],a_1[5][n]]
148             pub.publish(my_msg)
149             rate.sleep()
150         print("Control por trayectoria terminado.")
151         gripper.open()
152         data.writeoff()
153         print("Programa terminado.")
154     except rospy.ROSInterruptException:
155         rospy.logerr('Keyboard interrupt detected from the user.')
156 
157 if __name__ == '__main__':
158     main()
```