ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

“IMPLEMENTACIÓN DE UN SISTEMA DE VISIÓN Y CONTROL ASISTIDO DE UN BRAZO ROBÓTICO INDUSTRIAL”

INFORME DE MATERIA INTEGRADORA

Previo a la obtención del Título de:

INGENIERO EN ELECTRICIDAD, ESPECIALIZACIÓN EN ELECTRÓNICA Y AUTOMATIZACIÓN INDUSTRIAL

LUIS EDUARDO MALUSIN LEÓN

DAVID HUGO FREIRE SALGADO

GUAYAQUIL – ECUADOR

AÑO: 2018

DEDICATORIA

El presente proyecto lo dedico a mi familia, mis amigos y todas aquellas personas que me ayudaron a enriquecerme como profesional y sobre todo como persona, en especial a mis padres que estuvieron junto a mí en cada momento, quiero decirles que, gracias a su formación y atención, el fruto de su unión es ahora un profesional.

Luis Malusin.

Este proyecto se lo dedico a Dios, a mis padres y a mis amigos que creyeron en mí. Muchas gracias por ayudarme a llegar hasta aquí.

David Freire.

TRIBUNAL DE EVALUACIÓN

Msc. Janeth Carolina Godoy	**PhD. Ángel Domingo Sappa**
PROFESOR DE MATERIA INTEGRADORA	TUTOR ACADÉMICO

DECLARACIÓN EXPRESA

"La responsabilidad y la autoría del contenido de este Trabajo de Titulación, nos corresponde exclusivamente; y damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

....

Sr. Luis Eduardo Malusin León Sr. David Hugo Freire Salgado

RESUMEN

Este documento corresponde a la implementación de un sistema de control usando visión sobre un brazo robótico industrial, para generar una forma más eficaz de aprendizaje de trayectorias e implementarlo dentro de la industria. Con este propósito se desarrolló una aplicación en la cual se adquieren puntos de un espacio de trabajo usando una cámara industrial, para transmitirlas a través de la nube como trayectorias hacia el robot.

El capítulo 1 consiste en el desarrollo de la problemática con respecto a métodos de programación de robots industriales que ganan espacio en la actualidad, como es la programación por demostración, además de los objetivos que este proyecto espera cumplir.

El capítulo 2 se presenta con detalle tanto de los equipos recibidos, así como un pequeño análisis de investigaciones previas similares a la aquí propuesta, después se detalla los componentes usados para la solución planteada, dando así al lector un fondo teórico sobre los temas a tratar en cada etapa a desarrollar en los siguientes capítulos.

En el capítulo 3 se describe los pormenores de cada etapa implementada incluyendo detalles sobre capturas de color por parte de la cámara y la programación de una aplicación para generación de trayectorias con su posterior conversión en el código AS, el lenguaje que el brazo robótico necesita para moverse. Finalmente se enseñan las dos formas de transmisión de datos: local o remota.

En el capítulo 4 se muestran los movimientos realizados por el robot. Además, se realiza un análisis comparativo entre la trayectoria deseada del operador y la ejecutada por el brazo robótico.

ÍNDICE GENERAL

CAPÍTULO 1

1. DELIMITACIÓN DEL PROBLEMA.

1.1. Planteamiento del problema.

En la industria existen muchos procesos que requieren de extremo cuidado y de precisión, algunos de ellos son realizados a través de la intervención humana, aunque en otros se emplean robots articulados.

Los robots facilitan el trabajo tedioso y permiten una confiabilidad superior en el proceso, por lo que se los prefiere usar con mayor frecuencia. Un robot es programado con los movimientos que deberá de realizar para que realice la tarea deseada. Esto implica que se deben considerar todas las acciones, tanto positivas como negativas, para que dicho proceso se realice de manera satisfactoria [1].

Normalmente, la programación se la realiza a través de un controlador (Teach Pendant) o por un computador. Por ejemplo: en un sistema de ensamblaje de autos que tenga un brazo robótico, se debe indicar al robot de alguna manera para que elija, de una pila de productos, el adecuado para poder construir el vehículo. Si llegase a faltar alguna pieza, el robot debe esperar a que se encuentre disponible para evitar problemas en los siguientes pasos del proceso.

Debido a que pueden existir un sin número de condiciones que se deban validar, la programación de los robots se puede volver complicada y extensa si se emplean los métodos mencionados anteriormente. Además, puede llegar el caso en que se necesite añadir más pasos al programa. Al ser un proceso complicado, existe el riesgo de que se borren líneas de comandos si no lo maneja un experto. El tener que llamar constantemente a esa persona puede ser molesto tanto por el tiempo como cuestiones monetarias que involucran este tipo de servicio.

Por lo tanto, debería existir alguna forma donde se pueda programar al robot de manera más simple. De esta forma, cualquier operador de fábrica podría realizar cambios al robot sin necesidad de contar con otra persona.

1.2. Objetivos.

1.2.1. Objetivo general

Desarrollar la programación por demostración de un brazo robótico industrial para propósitos educativos de la materia Introducción a la Robótica Industrial.

1.2.2. Objetivos específicos

Definir la trayectoria del robot KAWASAKI identificando las características visuales, como el color, de una imagen capturada por una cámara industrial.

Programar al robot KAWASAKI, a través del lenguaje AS para que se pueda mover de una manera coordinada.

Emplear la arquitectura cliente-servidor para comunicar al robot KAWASAKI con un terminal de control.

1.3. Justificación.

Como punto de partida de toda aplicación que intervengan los brazos robóticos industriales, se requiere la enseñanza del robot acerca de su trabajo a realizar, incluso periódicamente por propósitos de calibración.

A esto se le llama programación del robot (o en inglés teaching). La clase de programación que se usará es por demostración, debido a que es más sencillo y adaptable que la actualmente usada; incluso es la mejor opción en algunas aplicaciones. Por ejemplo, para realizar el transporte de materiales en situaciones de riesgo por contaminantes o difícil acceso sería mejor usar este tipo de programación, ya que se puede realizar telecontrol, para evitar el contacto directo y prevenir la exposición de riesgo de los operadores.

La programación asistida por telecontrol propuesta reducirá la carga intelectual sobre el operador al ser realizada de manera dinámica mediante sus movimientos, es más sencilla al no requerir líneas de código del operador y no necesitará estar en el mismo ambiente de trabajo. Además, esta clase de control servirá como una base educativa, que permitirá al estudiante familiarizarse con el control por visión sobre un brazo robótico.

Para el telecontrol se construirá una aplicación, donde se emplearán técnicas de visión utilizando una sola cámara y el software Visual Studio para la generación de códigos.

1.4. Alcance.

Se diseñará una interfaz gráfica donde se visualice el área de trabajo de un robot industrial de manera virtual. Dicha interfaz será programada usando Microsoft Visual Studio 2013, en el lenguaje de programación C++.

Además, empleando dicha interfaz manipular el robot de manera remota a través de un computador. Para ello es necesario contar con dos cámaras: una que enfoque el área real de trabajo del robot y otra que muestre solamente el brazo robótico.

Con la imagen del área de trabajo, se realizará una discriminación visual para reconocer el objeto que se requiere manipular con el robot. Esta tarea será ejecutada con la librería OpenCV, la cual permite trabajar sobre imágenes. Entonces, se usará OpenCV para generar la trayectoria a partir del objeto reconocido mediante una técnica llamada Frame-by-Frame.

Finalmente, la trayectoria es transmitida al robot a través de la arquitectura de comunicación cliente-servidor.

CAPÍTULO 2

2. ESTADO DEL ARTE.

2.1. Antecedentes.

El brazo robótico Kawasaki del laboratorio de Control de Procesos Industriales de la Escuela Superior Politécnica del Litoral fue adquirido en el año 2015. Este robot trabaja dentro de un sistema de envase y vaciado de cilindros contenedores. Esta pequeña planta se llama Lucas Nülle IPA 26 y fue recibida junto con el brazo robótico.

La función del robot es la de transportar los contenedores individualmente desde un pallet hacia el sector de destapado, para después vaciar el cilindro en un depósito. Estas operaciones vinieron pre-programadas por el fabricante de la planta. Sin embargo, no se emplea el programa original del brazo robótico para fines académicos debido a que esa parte del sistema no se encuentra disponible.

Este robot se lo usa actualmente en las prácticas de la materia Introducción a la Robótica Industrial que imparte la ESPOL, como complemento del componente teórico. Las prácticas incluyen realizar operaciones básicas como mover objetos pequeños usando unas ventosas en el terminal del brazo. Para realizar la programación de estas tareas, se emplea un instrumento de mano (Teach Pendant) que vino con el robot. Esta manera de introducir órdenes se denomina programación activa y requiere estar cerca del brazo para que funcione de manera adecuada. El manipular el Teach Pendant por un tiempo moderado resulta cansado para el estudiante, debido al peso del mismo y porque requiere presionar un interruptor en la parte posterior de manera constante para mover el brazo robótico.

En cuanto a métodos de control mediante el uso de visión, en Ottawa, Canadá se desarrolló una interfaz empleando gestos con la mano en un entorno que comprendía un guante con 3 puntos guías y 2 cámaras, además del uso obligatorio de ropa negra por parte del operador para facilitar el reconocimiento de las guías [2].

El sistema adquiere datos de posición inicial que serán usados para minimizar la búsqueda de las guías en cada cuadro (frame*),* este proyecto se encuentra dentro de una red LAN para un procesamiento y ejecución en tiempo real.

2.2 Marco Teórico.

La robótica, esta fue concebida hace muchas décadas desde el área de la ciencia ficción intrigando al ser humano sobre estas máquinas inteligentes capaces de realizar las tareas de una persona regular.

Pues bien, el avance de la tecnología y la búsqueda de una producción más rentable han permitido la introducción de los robots en la industria, los cuales se desenvuelven en principalmente dos escenarios:

- Actividades repetitivas las cuales desarrollan con gran exactitud, bajo costo y alta productividad que un trabajador regular.

- Actividades específicas en ambientes peligrosos, poco agradables en los cuales puedan necesitar asistencia para realizar la actividad solicitada con mayor precisión.

La robótica aún no ha sido completada ya que muchas tareas que son fáciles y rutinarias para un humano son complejas y difíciles para un robot.

La mayoría de las tareas de un robot consiste en operaciones tipo pick and place pre programadas a mano por humanos, por lo cual una búsqueda para realizar esta tarea de importancia básica en toda programación es el objeto de esta investigación, entregando al operador una forma diferente de realizarlo a través de un método menos complejo.

Este proyecto está constituido en 3 etapas básicas las cuales son visión, comunicación y ejecución, las cuales definiremos a continuación:

2.2.1 Visión.

Introducción

La robótica implica equipos cada vez más robustos que se acerquen a los sistemas de un ser humano capaz de trabajar en diferentes escenarios sin que estos tengan que ser adecuados para su operación.

Una aproximación a esto es el reconocimiento de objetos ya sea para manipulación o para evitar una colisión, el uso de una cámara para que el robot sea consciente de su entorno de trabajo, mejorar su rendimiento especialmente en reconocimientos de posibles errores.

El uso de visión permite una manera más intuitiva y natural forma de mando a ser empleada por un operador, sin la limitante de aparatos físicos [2].

A continuación, se detalla tanto el software que incluye el IDE Visual Studio y la librería OpenCV como el hardware que implica a los medios por el cual se lleva a cabo el reconocimiento visual.

Visión por computadora

Es la transformación de datos obtenidos por una cámara de video en una nueva representación para cumplir un objetivo en particular. Se podría pensar que es una tarea sencilla por ser algo cotidiano en las vidas de las personas; sin embargo, en un sistema de visión computarizada, la computadora recibe un arreglo de números de la cámara para trabajar y nada más. Con frecuencia, estos números no representan fielmente a la imagen obtenida puesto que dentro de esa matriz existe un componente de ruido que aporta muy poca información, además de otros factores. El trabajo por realizar es entonces, convertir esta matriz de números en algo que se pueda manipular mejor [3].

Otro de los inconvenientes al capturar una imagen de una cámara es la pérdida de información. Una de las formas en que esta ocurre es debido a que la formación de la imagen involucra proyectar un objeto del mundo tridimensional a una superficie bidimensional. En ese proceso se perdieron los datos de profundidad de la imagen y una persona no puede notar si se trata de un gran objeto en la distancia o de uno pequeño que se encuentra cerca [4].

Realizar la corrección de estos dos problemas y de otros más es el objetivo por completar para poder manipular de mejor manera las imágenes obtenidas por una cámara y realizar tareas específicas. En este caso, se realizará un seguimiento de color para poder representar la trayectoria que el brazo robótico debe seguir.

Antes de cualquier tarea existe el reconocimiento y por tal se quiere decir la identificación de la posición y orientación de un objeto en el espacio, se usarán para esta investigación códigos en Visual Studio C++ para la aplicación de la librería OpenCV.

Software de desarrollo

Visual Studio

Entorno en el cual se desarrollará el código en el lenguaje de programación C++ para reconocimiento de la imagen usando el software de uso libre OpenCV. Este IDE amigable con el usuario y gran comunidad es lo que ayudará a realizar la conversión de las trayectorias captadas por la cámara en datos que le van a servir al robot para su movimiento [5].

Cuenta con soporte para otros lenguajes como C, C#, Basic, Java entre otros además del lenguaje C++.

OpenCV

OpenCV (Open Source Computer Vision Library) es una librería de uso libre, la cual provee la infraestructura para el desarrollo de aplicaciones de visión por computadora en productos comerciales de fácil uso y con código modificable [6].

La librería incluye algoritmos que comprenden ya sea visión clásica como visión por estado del arte, estos algoritmos pueden ser usados para detectar y reconocer objetos, caras, objetos en movimiento, entre otros.

Esta librería empezó como un proyecto en Intel en el año 1998. Desde el 2000 se encuentra disponible con licencia BSD, es decir, de código abierto. Se encuentra enfocada en proveer las herramientas necesarias para resolver los problemas de visión tal como se expusieron anteriormente [7].

Esta herramienta presenta los recursos necesarios para poder reducir el ruido presente en la imagen, así como poder segmentar la información necesaria para la aplicación deseada. Entre las funciones que posee OpenCV, se debe explicar el funcionamiento de algunas de ellas para este proyecto:

Espacio de color

El color tiene mucha importancia en sistemas de extracción de contenido de imágenes, el cual es almacenado en los vectores de intensidad de pixeles de la imagen para posteriormente obtener la información con facilidad. Sin embargo, el color puede ser representado en diferentes

espacios. Esta elección es muy relevante, no basta con elegir cualquiera de ellos, por lo que se debe tener en cuenta la aplicación para la cual se lo va a utilizar para elegir correctamente [8].

Se revisará algunos espacios de colores de interés para el uso en este proyecto:

RGB

Es usado con frecuencia para mostrar imágenes. Se compone de 3 colores: azul, verde y rojo. Como las cámaras, escáner emplean directamente la señal RGB tanto de entrada como de salida, este espacio de color es el más básico [8].

HSV

Este espacio es ampliamente usado en gráficos de computadora y es una manera casi intuitiva para describir el color. Sus tres componentes de color son: Matiz, Saturación y Valor. El componente de Matiz es invariante a cambios en iluminación o donde se coloque la cámara [8].

Para el presente proyecto se usará el espacio de color HSV. La imagen obtenida por la cámara se encuentra en el espacio RGB por lo que se debe aplicar una transformación con una función de OpenCV.

Funciones aplicadas a las imágenes

A continuación, se presentan algunas operaciones que se realizarán en las imágenes capturadas para facilitar su análisis.

Erosión y dilatación

Son operaciones morfológicas básicas, las cuales producen resultados que se pueden contrastar cuando se aplican a imágenes binarias o en escala de grises. Erosión causa que el objeto se disminuya su tamaño y remueve las partes pequeñas sustrayendo los objetos cuyo radio sea menor que el elemento estructural.

En imágenes binarias, adicionalmente la erosión remueve completamente los pixeles del perímetro de grandes objetos. En imágenes en escala de grises, reduce la iluminación de objetos luminosos en un fondo oscuro [9].

La dilatación en cambio incrementa el tamaño de los objetos, llenando agujeros, áreas huecas y conectando áreas que están separadas por

espacios más pequeños que el elemento estructural. Con imágenes binarias, adicionalmente añade pixeles al perímetro de cada objeto de imagen. Con imágenes en escala de grises, incrementa la luminosidad de objetos [9].

Segmentación

Los algoritmos de segmentación están basados en una o dos propiedades de valores de intensidad básicas: discontinuidad y similitud. La primera es para partir la imagen de entrada basándose en cambios bruscos de intensidad como en sus bordes. La segunda propiedad mencionada se basa en partir la imagen en regiones similares de acuerdo con un criterio predefinido. Uno de los métodos empleados para la segmentación es a través de un umbral. Existen dos tipos de técnica a través del nivel de umbral: global y local. La técnica global es cuando a toda la imagen se le asigna un solo valor de umbral. En cambio, en la técnica local, el valor de umbral se le asigna a cada píxel para determinar si pertenece al pixel de enfrente o atrás de la imagen. El umbral es útil para discriminar el primer plano del trasfondo de una imagen. Al seleccionar un valor adecuado, una imagen en blanco y negro puede convertirse en una imagen binaria. Esta imagen binaria debería contener toda la información necesaria acerca de los objetos de interés. La ventaja de usar este método es la reducción de complejidad de los datos y simplifica el proceso de reconocimiento y de clasificación [10].

Esta segmentación da como resultado una imagen denominada binaria, debido a que se solo posee dos colores: blanco y negro. El color blanco indica que el objeto es el deseado o también se lo puede representar con el número 1, mientras que el negro es exactamente lo contrario y se lo puede representar con el número 0.

Movimiento

Analizar una imagen en movimiento implica unir datos de un grupo de imágenes ligeramente diferentes en el espacio. Para poder aplicar este método de análisis, debe existir movimiento relativo: en este caso la cámara se mantendrá estática mientras el objeto se encontrará cambiando de posición. Así se obtiene una cadena de imágenes con ligeros cambios entre ellas. Para estudiar esta cadena, se lo puede hacer de dos formas: de manera continua y de manera discreta [11].

El objetivo de hacer este análisis es captar la trayectoria descrita por el objeto de color y obtener sus coordenadas para después manipular esa información. La manera elegida en este proyecto es el estudio continuo de la cadena de imágenes con el objeto estático, debido que para esta aplicación es más sencillo hacerlo con la cámara fija.

Momentos de imagen

En visión por computadora, el momento de una imagen es el promedio de las intensidades de sus píxeles. La selección de esos pixeles es basada en interpolación o alguna característica deseada. Se encuentran separados por orden: La orden del momento m (p, q) depende de los índices del momento p y q. Al sumar estos dos índices, se obtendrá el orden del momento de la imagen. Es decir, el momento m (0,1) representa el centro del objeto en una imagen. Se puede notar que al sumar los dos índices da el número 1, por lo que es de primer orden. Estos momentos brindan información del objeto de la imagen acerca de su posición y momentos centrales, además del origen del sistema de coordenadas. De los momentos m (1,0) y m (0,1) se puede calcular el centro del objeto observado. Esto se obtiene dividiendo los momentos con el momento central m (0,0) [12].

Una vez visto la parte del software del sistema de visión, ahora se tratará acerca de la cámara a emplear.

Cámara industrial Basler acA 1300-75 gc

Esta cámara posee un sensor de escaneo progresivo CMOS, que permite generar una imagen a la máxima velocidad de fotogramas teniendo así un mejor seguimiento de las trayectorias a realizar.

Su temperatura de trabajo se encuentra en 28 ± 1 °C muy por debajo de su límite que es de hasta 50 grados centígrados en operación o hasta 80 grados centígrados cuando no esté trabajando.

Además, puede tomar imágenes con una velocidad de 81 FPS en modo normal, lo cual cumple las expectativas para esta aplicación. Su resolución es de 1280 x 1024 píxeles y comunica sus datos a través de la interfaz Gigabit Ethernet.

Figura 2.1 Cámara IP Basler [13].

2.2.2 Comunicación.

Comunicación TCP/IP

El protocolo **TCP/IP** accede a la red física a través de los protocolos de más bajo nivel ofreciendo la posibilidad de interconectar redes de dos arquitecturas distintas proporcionando una comunicación fiable entre dos máquinas en cualquier punto de la red con una comunicación por paquetes [14].

Protocolo **TCP** para el transporte divide en paquetes la información a transmitir mientras los enumera para que el receptor la ordene y verifique lo transmitido por el servidor, si existe algún paquete perdido durante la transmisión, el receptor solicita su retransmisión al emisor.

El protocolo **IP** está a cargo de marcar cada paquete de información con la dirección del emisor y de receptor válidas. Cada computador posee una dirección única llamada IP que será usada para la respectiva comunicación con otros computadores en red.

Cloud

Para la comunicación el avance de la industria nos impulsa a una cuarta revolución que es el internet de las cosas, lo cual implica una comunicación activa entre sensores, actuadores y diferentes equipos integrándolos en procesos más complejos dejando atrás la idea de ser considerados una isla con accesos limitados mejorando su accesibilidad.

En la industria robótica y de automatización, la nube facilita el aprendizaje del robot al colectar datos de medios físicos y de su entorno para un mejor desarrollo además de poseer una mayor capacidad de almacenamiento sin comprometer su procesamiento en comparación a los sistemas de almacenamiento convencionales [15].

El uso de la nube en diferentes equipos nos lleva a la creación de arquitecturas a diferentes niveles Maquina-Maquina (M2M) o Maquina-Nube (M2C). Entre los beneficios de un cómputo colaborativo es el repartimiento equitativo de carga de procesamiento disminuyendo el tiempo de entrega de cualquier resultado [16].

2.2.3 Ejecución.

Robot Kawasaki RS03N

Especificaciones generales

A nuestra disposición en el laboratorio de Control de Procesos Industriales de la FIEC se encuentra el robot industrial Kawasaki RS03N, perteneciente a los robots para propósitos generales con diferentes aplicaciones en la industria como: palatización, pick and place, soldadura, etc.

Figura 2.2 Robot Kawasaki RS03N [17].

Tipo	Articulado
Grados de libertad	6 ejes
Carga útil	3 kg
Alcance horizontal	620 mm
Alcance vertical	967 mm
Repetitividad	±0.05mm
Velocidad máxima	6000 mm/s

Tabla 2.1 Especificaciones generales [17].

Especificaciones técnicas

Otras características más específicas del robot indican que tiene una protección IP 54.

En cuanto a los servomotores del robot estos son AC sin escobillas, con frenos en todos los ejes, con las siguientes características:

Eje	Máxima Velocidad	Máximo Torque	Momento de Inercia
JT1	360°/s	-	-
JT2	250°/s	-	-
JT3	2250°/s	-	-
JT4	540°/s	5.8 N*m	0.12 kg*m
JT5	225°/s	5.8 N*m	0.12 kg*m
JT6	540°/s	2.9 N*m	0.03 kg*m

Tabla 2.2 Especificaciones técnicas [17].

Espacio de trabajo

Es el área máxima en la cual se desenvuelve el robot, considerando las limitaciones de cada articulación.

Eje	Rango de movimiento
JT1	±160°
JT2	+150°~ -60°
JT3	+120°~ -150°
JT4	±360°
JT5	±135°
JT6	±360°

Tabla 2.3 Rango de movimiento [17].

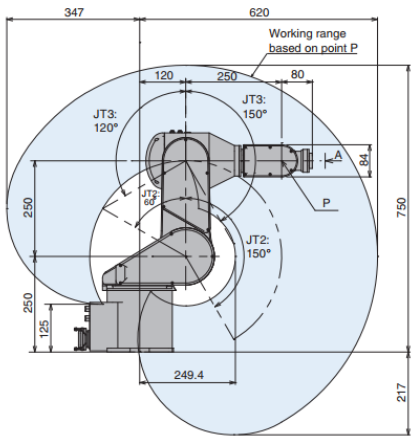

Figura 2.3 Espacio de trabajo [17].

Programación por demostración

Esta es una técnica en la cual, a diferencia de la programación convencional, este aprende al ver los movimientos a realizar ya sea por una persona u otro robot segmentos o la tarea completa a desempeñar.

Aprendiendo una tarea

Para realizar esto uno busca mediante código que el robot extraiga las características más importantes de la demostración o seguimiento que se realiza, además de asegurar la reproducibilidad del mismo, teniendo distintos acercamientos para su aprendizaje.

Aprendizaje simbólico

La forma más común para la codificación simbólica es segmentar y codificar secuencias de acciones predefinidas esto incluye posturas y posiciones [18].

La ventaja de esto es la relativa facilidad en que se puede aprender acciones de gran dificultad, pero esto implica un gran conocimiento a priori para la generación de segmentos de manera eficiente.

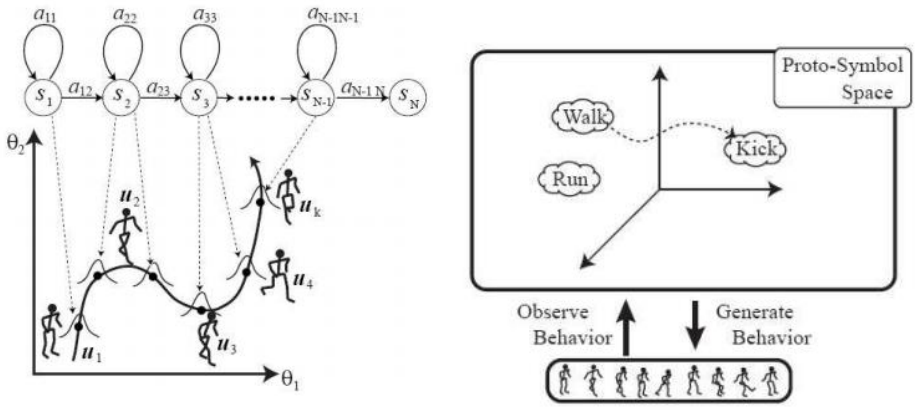

Figura 2.4 Ejemplo de un aprendizaje simbólico [18].

Aprendizaje a nivel de trayectorias

La correcta codificación de movimientos es consecuencia de una buena selección de variables ya sea para un movimiento cíclico o discreto, la recolección de señales del movimiento de articulaciones y sus torques se

transmite ya sea a una transformación lineal o al uso de métodos no lineales [18].

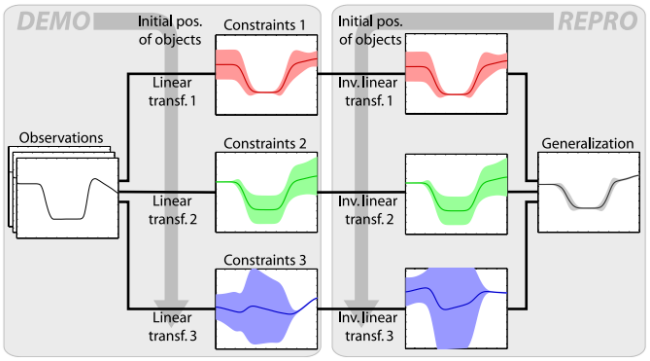

Figura 2.5 Ejemplo con trayectoria lineales [18].

CAPÍTULO 3

3. METODOLOGÍA DE TRABAJO.

En este capítulo se contempla el desarrollo completo del sistema de control propuesto.

Se presentará a continuación el diagrama primitivo del proyecto, el cual posee 3 etapas básicas: Visión, comunicación y ejecución.

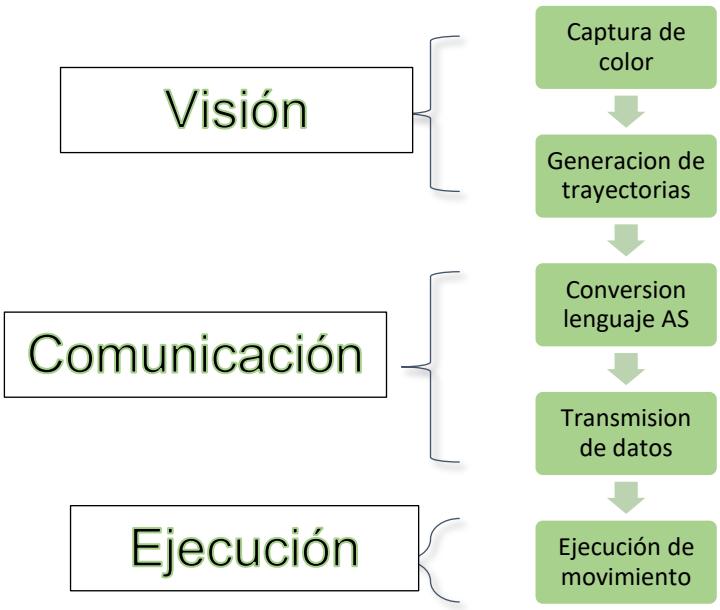

Figura 3.1 Diagrama primitivo.

De forma general, la solución propuesta como alternativa a la programación del brazo robótico incluye el uso de un guante que tendrá 2 colores en dedos distintos para realizar la trayectoria deseada, la cual será capturada por una cámara IP Basler. Después una aplicación realizada en lenguaje C++ genera un documento con coordenadas (X, Y) para, a través de lenguaje Python, su posterior conversión a lenguaje AS y consecuente realización por parte del brazo robótico Kawasaki RS003N.

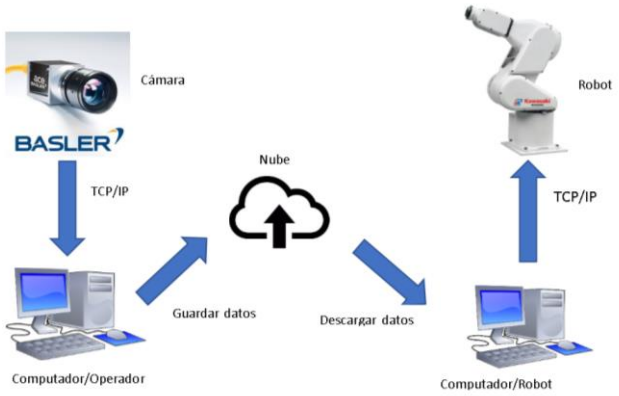

Figura 3.2 Diagrama esquemático general.

El sistema de visión se encuentra comprendido por: una cámara Basler y dos computadoras.

La cámara captura la información de la trayectoria necesaria para después transmitirla a la primera computadora.

3.1. Captura de color.

Mediante el uso de la cámara industrial Basler se procede a realizar la captura de trayectorias. Esto se realiza a través de código, con la ayuda de la librería OpenCV. Esta librería permite extraer de la imagen información como su textura, iluminación o en este caso, eliminar el espectro de colores hasta obtener el color deseado. La realización de las trayectorias por parte del usuario será empleando un guante con colores en sus dedos.

Los colores por filtrar deben ser fáciles de identificar para optimizar más el proceso. Por ello se elige el color rojo y el color verde, aunque también es viable emplear otros colores. El color rojo y verde estarán presentes en el guante y realizan distintas acciones en la ejecución de la trayectoria para la aplicación que se ha escogido, es importante que se realice este reconocimiento ya que de esto dependerá que en la trayectoria final se obtenga lo deseado.

Para que la cámara capture un determinado color, es necesario ajustar diversos parámetros del espacio de color HSV que ella observa en el espacio de trabajo. Estos parámetros son: matiz, saturación e intensidad, los cuales se ajustan mediante un trackbar en la aplicación, como se aprecia en la siguiente figura 3.3:

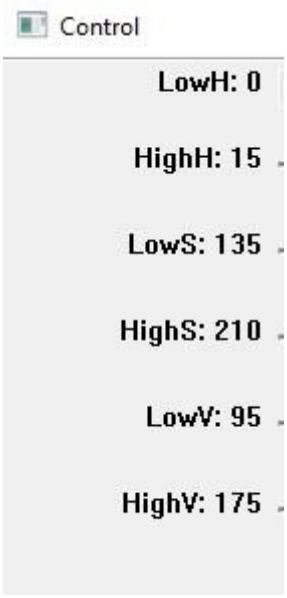

Figura 3.3 Trackbar de ajuste de espacio de color

La figura 3.4 muestra un objeto de color rojo en el espacio de trabajo, donde se modifican los parámetros mencionados para poder discriminar este color de los demás. En la figura 3.5 se aprecia el resultado de esta operación.

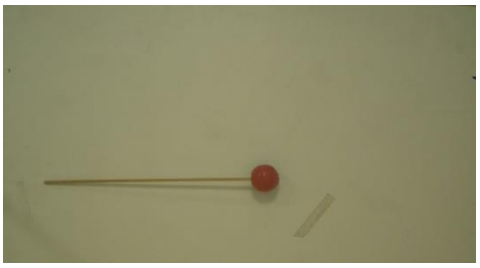

Figura 3.4 Vista normal del objeto en el espacio de trabajo

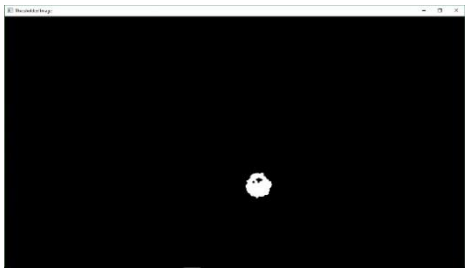

Figura 3.5 Vista del objeto con filtro de color

3.2. Generación de trayectorias.

Una vez realizado la captura de color por parte de la cámara se obtendrá una serie de puntos que corresponden al centro de masa del área de color rojo que se muestra ante la cámara; en este caso se usará el guante, pero otros objetos adquieren el mismo resultado: como pelotas o dibujos de color rojo.

Estos puntos son empleados para determinar la trayectoria del brazo robótico. Sin embargo, es necesario notar que pertenecen al plano de la cámara, por lo que se debe encontrar alguna forma de llevarlos al plano del robot.

Para seleccionar el color deseado, se debe usar la librería OpenCV como se mencionó anteriormente. La cámara Basler requiere a su vez de librerías adicionales para poder activarla, de todos ellos se eligió C++. De todas las funciones que existen en OpenCV, se eligieron unas específicamente para filtrar la imagen y obtener el color rojo.

Esta función toma la imagen de la cámara junto con unos niveles de color: H, S, V y devuelve la imagen que será observada durante toda la grabación. Estos niveles de color permiten la selectividad de la imagen; es decir, que variando estos niveles es posible hacer que la cámara detecte al objeto o no.

Por otro lado, esta función permite obtener los momentos de la imagen filtrada con el paso anterior. Estos momentos son necesarios para calcular el área del objeto y discriminar: si el objeto es muy pequeño, el programa no lo detectará, por lo que debe ser más o menos del tamaño necesario.

Inicialmente, se crean dos puntos finales con valores predeterminados para evitar problemas al iniciar el programa. Esta pequeña fórmula permite obtener los puntos necesarios para la trayectoria.

Para que se aprecie la trayectoria en pantalla es necesario dibujar una línea entre todos los puntos obtenidos anteriormente. La función line permite este efecto, la cual recibe los puntos inicial y final de cada movimiento para trazar la línea entre ellos. Finalmente, guarda esta línea en una imagen del entorno; para observarla es necesario añadirla a la grabación principal:

Esto es posible debido a que esencialmente una imagen es una matriz de puntos, por lo que una operación como la suma de dos matrices es sencilla de realizar.

3.2.1. Calibración de coordenadas del espacio de trabajo.

Para realizar el proceso de calibración se necesita saber el equivalente de 1 cm en número de pixeles o su equivalente en el plano de la cámara, para lo cual se procede a obtener las coordenadas de cada esquina del área de trabajo de la cámara mientas las señalamos en un plano, se toma las medidas de distancia tanto en el plano real como el de la cámara y se procede con el uso de una regla de tres debido a su relación lineal.

El área de trabajo tiene las siguientes medidas:

Cámara	Mundo real
Largo: 1238 pixeles	Largo: 65.16 cm
Ancho: 646 pixeles	Ancho: 34 cm

Tabla 3.1 Medidas del espacio de trabajo del proyecto

Por lo tanto:

$$1cm = 19\,pixeles \qquad (3.1)$$

Una vez obtenido el dato de la cantidad de pixeles por cm, queda a disposición del programador la escala en la cual el robot replicara el movimiento, para esto se debe tener en consideración no excederse del área de trabajo del robot y las limitaciones de los ejes, para que toda trayectoria se pueda realizar.

Para la generación de las coordenadas en el espacio de trabajo del robot usamos:

$$\text{tempx} = 0.37 * (\text{posX} - 636) \qquad (3.2)$$

$$\text{tempy} = 0.47 * (\text{posY} - 445) \qquad (3.3)$$

Cuando es el color rojo, se guardan los puntos en un archivo de texto plano. Los puntos iniciales se vuelven iniciales y se repite de nuevo el ciclo

Si se detectara el color verde, en cambio, sucede lo siguiente:

Después de ser ajustados los puntos a coordenadas en el espacio de trabajo del robot se procede a llevarlas a código AS.

3.3. Conversión a lenguaje AS

Una vez almacenado el archivo con los puntos obtenidos por la calibración, se procede a un proceso de conversión en lenguaje AS para la posterior interpretación por parte del robot que ejecutaras las ordenes enviadas.

Dado el formato del código que se encuentra dentro de la memoria del controlador del robot, el código de conversión incluirá en la primera línea el comando SPEED el cual establecerá en el robot una velocidad inicial, posteriormente se incluirán los comandos de movimientos con los respectivos puntos que se encuentran dentro del espacio de trabajo del actuador, para finalizar el código incluye el comando END para que la respectiva ejecución termine de forma ordenada y sin fallos para poder recibir una próxima operación.

Esto se realiza con un código elaborado en Python, como se muestra en el anexo 1.

Se puede observar que se crea otro archivo dentro del servidor con las instrucciones necesarias, de la siguiente manera: Si se usó el color rojo para el punto, la instrucción es “JMOVE”. Si se usó el color verde, la instrucción es “JAPPRO”. Después van los puntos (X, Y) del documento original, añadiendo el punto Z, el cual es constante debido a que el brazo solo se moverá en dos dimensiones. De esta forma se obtiene el código AS necesario para el movimiento del terminal del robot.

3.4. Transporte de información.

Existen dos formas para transportar los datos obtenidos anteriormente: local y remota. Local se refiere al uso de la red del Laboratorio de Control de Procesos, mientras que de manera remota tiene que ver con el empleo de la red de la Espol entre la biblioteca y el Laboratorio. Ambas opciones usan el lenguaje de programación Python para trasmitir datos, la diferencia radica en que el código se complica cuando se requiere la transmisión remota.

3.4.1. Local.

En la transmisión local, el archivo AS de la trayectoria se almacena en un servidor web HTTP Apache. Además, se emplea un servidor y cliente FTP, llamado FILEZILLA. Ambos servidores se obtienen al trabajar en un paquete de software libre llamado XAMPP, por lo que se necesita un cliente FTP para recolectar el archivo deseado.

Se configura el servidor Apache y el servidor FILEZILLA de acuerdo con los requerimientos del operador.

Debido a que FILEZILLA es un servicio de manejo de archivos, el cual usaremos por lo cual se procede con la creación de un cliente FTP.

Existen varias maneras de crear un cliente FTP, donde una de ellas es usando código Python, como se muestra en el anexo 1. Ejecutando el cliente FTP, se puede descargar el código AS de la trayectoria en la computadora que da la orden al robot.

El robot abre un puerto de comunicación TCP\IP con el comando TCP_LISTEN e inmediatamente confirma la comunicación mediante el uso del comando TCP_LISTEN, al recibir cualquier tipo de dato por parte del servidor. Esto se encuentra detallado en el siguiente código:

```
.PROGRAM open_socket() #61
  er_count = 0
listen:
  TCP_LISTEN ret1,port
  IF ret1<0 THEN
    IF er_count>=5 THEN
      PRINT "Connection is failed (LISTEN). Program is stopped."
      sock_id = -1
      GOTO exit
    ELSE
      er_count = er_count+1
      PRINT "TCP_LISTEN error = ",ret1," error count = ",er_count
      GOTO listen
    END
  ELSE
    PRINT "TCP_LISTEN OK ",ret1
  END
  er_count = 0
```

Figura 3.6 Apertura de puerto por robot

Interfaz local del emisor

Una vez que se cuenta con estos programas para guardar puntos y transformar, se vuelve necesario tener una interfaz para comunicarse con el servidor local Apache. Una de las opciones consideradas es la elaboración de una página web escrita en HTML y PHP.

Figura 3.7 Interfaz local del emisor

El botón de grabar activa la aplicación de la cámara para poder capturar la trayectoria deseada. Una vez generado el archivo de puntos, el segundo botón los convierte a lenguaje AS dentro del servidor.

El primer botón ejecuta el programa escrito en C++ de la cámara y permite guardar el archivo de puntos en el servidor local. El segundo botón por el otro lado ejecuta el programa escrito en Python que transforma los puntos en código AS. En anexos se presenta el resto de código de la página web.

3.4.2. Remoto.

Para el control de forma remota se usa la nube, debido a que el servidor Apache configurado solo funciona en una red local. En este proyecto se emplea Google Cloud debido a su facilidad de manipulación de datos.

Nos permite por lo tanto la transmisión de archivos en cualquier espacio que cuente con acceso a internet.

El cual descarga solo la información que se encuentra dentro del archivo almacenado en la nube por lo cual, el paso siguiente es darle formato primero a como estaba inicialmente en la nube y después pasarlo a código AS para poder ser enviado al robot.

Para efectos de tener una guía de lo que debe de ilustrar el brazo robótico se creó un código que permite graficar de las coordenadas que fueron usadas con anterioridad en el diseño del dibujo.

Interfaz del emisor

Debido a que se emplea la nube y no el servidor Apache, la interfaz escrita en lenguaje PHP no se puede utilizar para el envío de datos. Por ello se creó una segunda interfaz, pero en lenguaje Python.

Figura 3.8 Interfaz del receptor

En esta interfaz, el primer botón hace la misma función de activar la aplicación que permite a la cámara capturar la trayectoria deseada. Después de obtenido el archivo, se lo puede transformar a lenguaje AS antes de subirlo a la nube. También cuenta con un botón de salida para terminar el proceso si se lo requiere.

Interfaz del receptor

También es necesaria una interfaz que permita ejecutar estos archivos escritos en Python para descargar los datos del servidor y trasmitir la información al robot. De otra manera, se volvería engorroso buscar cada programa por separado para ejecutarse.

Figura 3.9 Interfaz del receptor

El primer botón realiza la descarga del archivo AS de la nube. Una vez obtenido el archivo, se espera a que el robot se encuentre en estado de recepción de comandos para poder presionar el botón de envío. Se colocó un botón de salida por si se necesite interrumpir la operación, aunque también se puede cerrar la ventana de la interfaz para el mismo efecto.

3.5. Ejecución de movimiento:

Antes de realizar la ejecución se debe aclarar que existirán 2 formas para preparar el robot para iniciar la tarea.

3.5.1. Local.

De manera local se debe encender los motores usando el Teach pendant que viene junto al controlador y si este se encuentra en modo repeat, hará que el programa se inicie automáticamente iniciando así la tarea.

3.5.2. Remota.

Mediante el uso de una VPN y un escritorio remoto se procederá a la activación de una señal programada en un PLC lo cual encenderá una bobina que permitirá la activación de los motores.

Establecido el encendido de los motores se podrá pasar a la adquisición de datos.

El robot recibe la información y las ejecuta línea por línea. El programa dentro de la memoria del robot se desglosa en varias etapas:

Inicialización:

El robot toma una posición inicial con la ejecución del programa mv2hm, el cual es un programa netamente de movimientos del robot. Luego se inicializan variables de comunicación y que se usaran durante la programación.

Posición inicial:

Cuando el robot se encuentra en posición esta se adquiere y se guarda para usarla como origen al plano de trabajo del robot. Este paso se realiza con la intención de que todos los puntos que se ejecutaran sean con respecto a la nueva base establecida y no a la referencia original del terminal.

SPEED:

El programa espera como primera orden un comando de velocidad la cual se mantendrá durante toda la ejecución, una vez adquirida retorna una confirmación por consola para poder proseguir.

Comandos:

Una vez establecidos la posición inicial y la velocidad se espera las siguientes órdenes de movimiento, estas pueden ser JMOVE o JAPPRO.

JAPPRO:

Para aquellos casos en los que se necesite hacer un trazo levantado se implementó el uso del color verde, el cual se verá reflejado como un comando JAPPRO para el robot. Lo cual significa que el robot se acerca al punto enviado, pero será una aproximación al alejarse de este una medida indicada en el eje Z.

JMOVE:

El reconocimiento del color rojo implicará un trazo a dibujar lo cual para el robot será un comando JMOVE. Este trazo involucrara un movimiento tipo JOINT por

parte del robot es decir un movimiento proporcional por cada eje usado durante la acción.

END:

Realizados los movimientos el robot procede a cerrar los puertos de comunicación.

CAPÍTULO 4

4. ANÁLISIS DE RESULTADOS.

Este capítulo reúne la información tratada en capítulos anteriores y muestra los resultados adquiridos durante la implementación del control asistido por visión.

Por lo tanto, explicará las diferentes etapas tanto de visión, adquisición de datos, conversión lenguaje AS y ejecución por parte del robot.

4.1. Reconocimiento de color.

El sistema de visión compuesto por la cámara y aplicación programada en C++ realiza la captura del centro de masa de la sumatoria de todos los pixeles que son identificados por el software con la configuración de color preestablecida, razón por la cual el uso del guante con la intención de darle más control por parte del usuario y al mismo tiempo reducir la posibilidad de que exista interferencia externa ya sea por algún rastro de color en la piel del operario o es sus ropas resultó ser una buena idea. Por esta misma razón el espacio de trabajo de la cámara se lo realizó sobre un fondo blanco.

Figura 4.1. Uso de guante en espacio de trabajo.

Además, se implementó un método de entrada y de salida para la detección de colores que disminuye aún más la interferencia de colores externos. Se trata de presionar dos teclas: la tecla “e” para empezar la captura de trayectorias y la tecla “t” para finalizar.

Aquí se muestra el inicio de la captura de la trayectoria a seguir del robot. Se emplea el guante con colores propuesto anteriormente para realizar el movimiento.

Figura 4.2. Gráfico realizado a pulso.

Como se puede observar, el trazo depende únicamente de la habilidad del operador para dibujarlo, por lo que es posible que la trayectoria grabada por la cámara no sea la deseada, debido a su mala coordinación. Por ello, se imprimieron unos modelos para facilitar la construcción del movimiento a realizar.

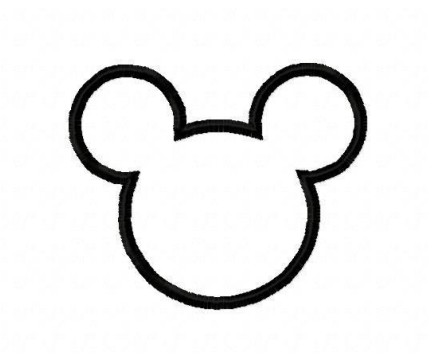

Figura 4.3. Modelo de trayectoria

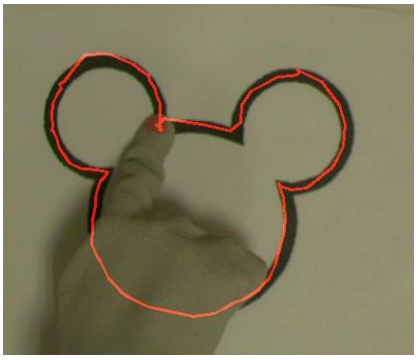

Figura 4.4. Trayectoria realizada usando el modelo

4.2. Adquisición de datos.

Una vez obtenida la trayectoria, el archivo con las coordenadas es generado inmediatamente:

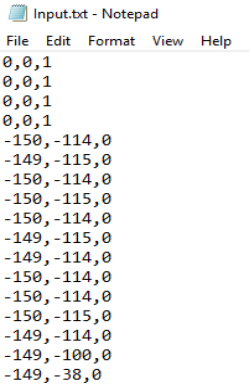

Figura 4.5. Archivo de coordenadas de un movimiento continúo

Los dos primeros puntos indican las coordenadas en (X, Y) respectivamente, mientras que la tercera coordenada indica el tipo de movimiento en cada paso. Si es 0, el movimiento es JMOVE que significa realizar un trazo con todas las junturas del robot. Si es 1, el movimiento es JAPPRO, con la cual el robot realiza una aproximación. Al comienzo, la trayectoria tiene un JAPPRO para que el terminal se dirija al comienzo del movimiento rápidamente.

Como se puede observar en la figura 4.5, la trayectoria no tiene otros movimientos JAPPRO después de comenzar. Por lo tanto, el movimiento es continuo y sin espacios.

En la siguiente figura se aprecian los puntos de una trayectoria que si tiene espacios:

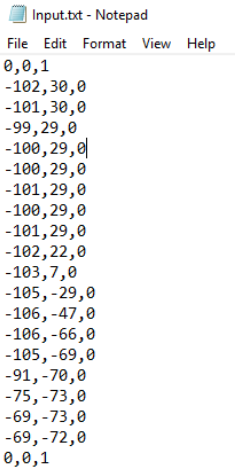

Figura 4.6 Archivo de coordenadas de un movimiento no continuo

La trayectoria se trata del nombre “LUIS”, por lo que necesita de movimientos JAPPRO para que el robot levante el marcador al finalizar cada letra. Se puede ver que mientras más complicado sea el movimiento, mayor cantidad de puntos necesitara para su realización.

Con los puntos obtenidos se realizó un dibujo para mostrar al operador como quedaría la trayectoria en el segundo computador. Se puede notar que el dibujo es continuo, y es como se observaría la trayectoria del robot sin la ayuda de JAPPRO.

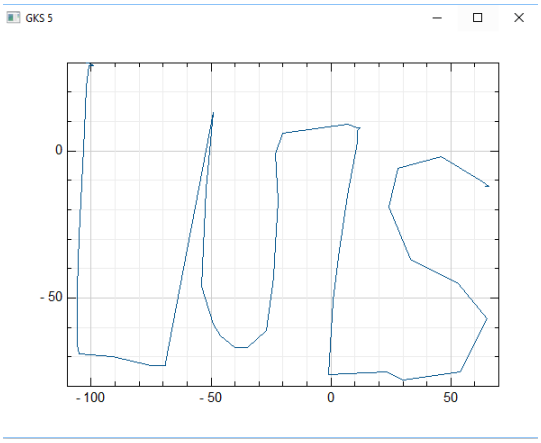

Figura 4.7. Dibujo realizado por el operador

En este dibujo no es posible observar la acción del JAPPRO, por lo que lo convierte en una guía aproximada acerca de la trayectoria deseada por el operador.

Sin embargo, surgió un inconveniente al momento de tomar los puntos de las coordenadas para dibujar o enviarlos al robot: ambas graficas se encuentran reflejadas para el operador. La siguiente figura muestra un ejemplo de dibujo revertido:

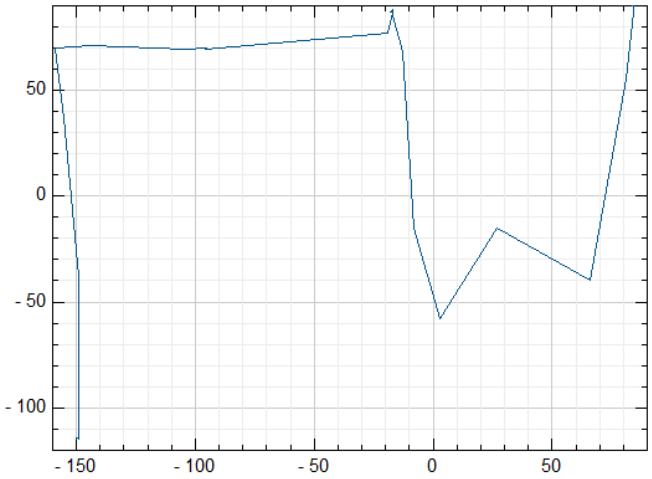

Figura 4.8. Dibujo invertido realizado por el operador

Se puede ver que, al ser comparadas la figura 4.2 con la figura 4.8, efectivamente se produjo una reflexión y, por lo tanto, el operador no observó el trazo original realizado.

Para solucionar este problema, se puede observar que los puntos en el dibujo necesitan reflejarse de igual manera, cambiando la perspectiva sin modificar valores. Basta con multiplicar por -1 los puntos del eje Y para tener el resultado deseado.

4.3. Conversión a lenguaje AS.

Luego se procede a transformar a lenguaje AS, quedando el archivo de la siguiente manera:

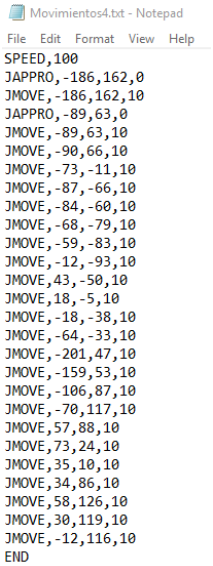

Figura 4.9. Coordenadas en comando AS.

En este caso, las dos coordenadas (X, Y) quedan idénticas como el archivo anterior. El cambio radica en que la tercera coordenada es Z y se mantiene constante debido al movimiento en dos dimensiones que realiza el robot. El valor de esta coordenada depende si el movimiento de JMOVE o JAPPRO para lo cual será 10 o 0 respectivamente, lo cual será clave para realizar los trazos alzados que requiera la gráfica.

4.4. Ejecución.

A continuación, se presenta el movimiento realizado por el brazo robótico, donde en el terminal se coloca un marcador para apreciar mejor el seguimiento de la trayectoria programada.

Al principio se fijó el marcador usando ligas elásticas, sin embargo, no era un buen método puesto que no quedaba recto al momento de dibujar y causaba imperfecciones en el trazo. Por ello se optó por colocar una base metálica, como se puede apreciar en la siguiente figura:

Figura 4.10 Robot con un marcador en el terminal

El robot realizó el trazo en una pizarra acrílica, colocada cerca de su base, de esta manera se puede borrar el trazo anterior y comenzar otro fácilmente. Otro aspecto por tomar en cuenta es el terminal del robot, el cual necesitó encontrarse en un punto de partida específico antes de comenzar a realizar la trayectoria. Este punto fue elegido tomando en cuenta que se debe aprovechar el máximo espacio posible del pizarrón.

Figura 4.11. Robot en posición de dibujo

Para trazos continuos, como aquellos donde se usaron los modelos impresos, la trayectoria queda de la siguiente manera en la pizarra:

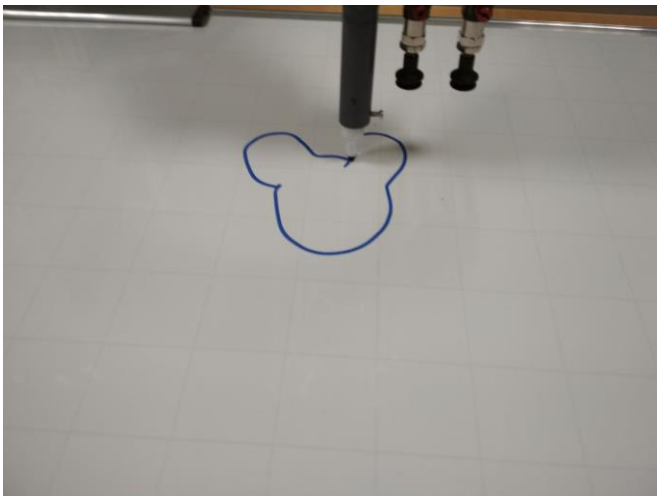

Figura 4.12. Dibujo realizado con patrón de trayectoria

Comparando con la figura 4.4 se observa que el trazo dibujado con el marcador es bastante similar al realizado con el guante guía.

Para trazos no continuos, el robot levantó el marcador cuando fue necesario al recibir la orden JAPPRO, además de que graficó en la perspectiva correcta, tal como se muestra a continuación:

Figura 4.13. Dibujo de un nombre realizado a pulso

Se observa que el dibujo realizado es, además, una réplica de la figura 4.7, solo que con los espacios necesarios entre las letras.

CONCLUSIONES Y RECOMENDACIONES

El sistema de control implementado propone al operador un método para realizar de forma más cómoda la generación de trayectorias o movimientos punto a punto, claro está dentro de un ambiente controlado, ya que el instrumento para la elaboración y adquisición de puntos necesita una cantidad de lúmenes constante. Por lo tanto, la luz tiene un impacto importante en el sistema propuesto debido a que, si cambia, el color no es capturado completamente, provocando la caída del sistema surgiendo la necesidad de re calibración, esta situación ocurrirá también si el usuario deseara transportarlo a otro ambiente de trabajo. Esto se solucionaría con la integración de un sistema de luces acoplado a la estructura de la cámara o al guante del operador, minimizando horas de paro del sistema y costos de mantenimiento por calibración.

Como se habló con anterioridad en el marco teórico, la industria se moviliza hacia tecnologías inclusivas dentro de redes más accesibles y menos complejas, por lo cual se implementó en este proyecto el uso de la nube como medio de transmisión de datos. Esta herramienta permite el control del robot a distancia, es decir una tele operación, limitada únicamente por la accesibilidad al internet por parte del computador en red con el brazo robótico, lo cual implicaría la necesidad de una conexión estable y segura para que los datos no se pierden en el proceso, aumentando la confiabilidad del sistema. El uso de la nube deriva en la creación de archivos para la transmisión de los comandos a realizar. Dado el tiempo que se necesita para la carga y descarga del archivo, además del reordenamiento del documento, ya que en el proceso se adhieren caracteres basura que necesitan ser omitidos para una correcta ejecución, esto se refleja en un mayor tiempo de procesamiento del sistema. Por este motivo el control implementado no se podrá realizar en tiempo real.

Dada la aplicación que se desarrolló para la implementación del control de visión, se le concedió a la cámara una amplia área de trabajo con el propósito que el usuario tenga una mayor movilidad y espacio para detalles si es necesario. Esto implica un ajuste a realizar para que se pueda ejecutar el movimiento dentro del área de trabajo del robot, para lo cual se realizaron diferentes mediciones de ambas áreas empleadas en el proceso de calibración. Cabe recalcar que para el trazado de las trayectorias se usaron movimientos de tipo lineal para minimizar el estrés sobre los servomotores que podrían ocurrir al realizar una tarea prolongada.

Sin embargo, esta acción implica una limitación física sobre los trazos a realizar incluso dentro del área de trabajo del robot, dado que trazos complejos en áreas próximas darían lugar a la necesidad del uso súbito de una mayor cantidad de articulaciones, causando un estrés mecánico sobre los servomotores que forman parte de las articulaciones rotatorias.

Cabe recalcar que, dadas algunas limitaciones en cuanto a hardware el proyecto se lo realizo en 2D. Pero con la adquisición de más cámaras colocándolas de tal forma que permitan obtener datos de profundidad, o una cámara 3D, estas se complementarían junto a la programación realizada para obtener un control en 3D y así aprovechar el espacio de trabajo total del terminal a controlar.

Para la calibración que corresponde al tono de color a reconocer por la cámara, esta se debe realizar in situ con la fuente de luz específica que se va a usar en la aplicación, sobre el espacio de trabajo donde la cámara tenga rango de visión y con un color de fondo que ayude al reconocimiento. Por lo general se usan los extremos del espectro de colores, ya sea negro o blanco, en nuestro caso usamos un fondo de color blanco para una mejor reflexión de la luz.

Para el uso de la nube se recomienda adquirir un espacio dentro de Google Cloud, ya que permite la integración de esta herramienta dentro de la programación a realizar, además de que actualmente cuenta con gran soporte por parte de la comunidad, aunque sea una herramienta relativamente nueva.

BIBLIOGRAFÍA

[1] A. Denasi, B. T. Verhaar, D. Kostic, D.J.H Brujinen, H. Nimeijer, T.P.H Warmerdam, “Robot Programming by Demonstration”, Proceedings of the 10th Philips Conference on Applications of Control Technology, Hilvarenbeek, Netherlands, 2009, pp. 149-153

[2] J. Kofman, X. Wu, T.J Luu,S. Verma, “Teleoperation of a Robot Manipulator Using a Vision-Based Human-Robot Interface”, IEEE Trans. Ind. Electronics, Vol 52, no. 5, October, 2005, pp. 1206-1219

[3] G. Bradsky, A. Kaehler, “Overview”, in LearningOpenCV, 1th edition. Sevastopol: O’Really, 2008, ch. 1, pp. 18-20.

[4] P. Corke, “Image Formation”, in Robotics, Vision and Control, Berlin: Springer, 2011, ch. 11, pp. 251.

[5] Visual Studio [Online] disponible en https://www.visualstudio.com/es/vs/features/, visitado el 4 de febrero del 2018.

[6] OpenCV [Online] disponible en https://opencv.org/about.html, visitado el 4 de febrero del 2018

[7] K. Pully, A. Baksheev, K. Kornyakov, V. Eruhimov, “Real-Time Computer Vision with OpenCV”, acmqueue, vol. 10, no. 4, pp. 2-3, Apr. 2012.

[8] S. Sergyan “Color Content-based Image Classification”, 5th Joint Symposium on AMIaI, Slovakia,Poprad, 2007, pp. 2-4.

[9] L. Gracia, C. Perez-Vidal, C. Gracia, “Computer Vision Applied to Flower, Fruit and Vegetable Processing”, World Academy of Science, vol. 5, no. 6, pp. 5, 2011.

[10] S. Al-amri, N.V. Kalyankar, S.D. Khatmitkar, “Image Segmentation by Using Threshold Techniques”, Journal of computing, vol. 2, no. 5, pp. 1-2, May. 2010.

[11] G. Gomez, “Movimiento” de Visión Computacional, México, 2008, ch. 9, pp. 123.

[12] J. Lamer, D. Cymbalak, F. Jakab, “Computer vision based object recognition principles in education”, ICETA, 2013, pp. 4

[13] Basler acA1300-75gc disponible en https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/aca1300-75gc/ visitado el 7 de febrero del 2018

[14] K.R.Fall, W.R.Stevens, “TCP: The Transmission Control Protocol” in *TCP/IP Illustrated Volume1*, Addison-Wesley, Michigan, 2011, pp. 579-591.

[15] B. Kehoe, S. Patil, P. Abbeel, K. Goldberg, “A Survey of Research on Cloud Robotics and Automation”, IEEE Trans. Auto. Science and Engineering, Vol 12, no. 2, April, 2015, pp. 398-409

[16] G. Hu, W. P. Tay and Y. Wen, "Cloud robotics: architecture, challenges and applications," in *IEEE Network*, vol. 26, no. 3, pp. 21-28, May-June 2012.

[17] Kawasaki [Online] disponible en: https://robotics.kawasaki.com/en1/products/robots/small-medium-payloads/RS003N/ visitado el 4 de febrero del 2018

[18] Billard A., Calinon S., Dillmann R., Schaal S. (2008) Robot Programming by Demonstration. In: Siciliano B., Khatib O. (eds) Springer Handbook of Robotics. Springer, Berlin, Heidelber

ANEXOS

Código de las funciones escritas en Python

```
import os
import google.cloud.storage
def download_file(archivo,archivol,bucket_name):

    num=0
    num2=0

    salida=open('Movimientos2.txt','w')
    storage_client = google.cloud.storage.Client()


    bucket= storage_client.get_bucket(bucket_name)


    blob=bucket.blob(os.path.basename(archivo))

    a=blob.download_as_string()
    salida.write(str (a))
    salida.close()
    salida2=open('Movimientos2.txt','r')
    salida3=open('Movimientos3.txt','w')

    line=salida2.read().splitlines()

    for caracter in line:
        estructura=caracter.split(r'\n')
        ancho=len(estructura)
        #print(estructura)

        for elemento in estructura:
            salida3.write(str(elemento)+"\n")
    #for carac in ancho:
     #    salida3.write(estructura[carac]+"/n")
    salida2.close()


  salida4=open(archivol,'w')
  salida5=open('Movimientos3.txt','r')
  line2=salida5.read().splitlines()

  for carac in line2:

      if num<1:
          estruc=carac.split(',')
          #s0=str(estruc[0])
          #sl=str(estruc[1])
          #salida4.write('SPEED,100'+"\n")
          num=num+1
      else:

          estruc=carac.split(',')
          s0=str(estruc[0])
          if s0=="END'":
              #salida4.write("END")
              break
          if s0=='JAPPRO':
              sl=str(estruc[1])
              s2=str(estruc[2])
              salida4.write(s0+","+sl+","+s2+","+'0'+'\n')
          else:
              if num2==0:
                  sl=str(estruc[1])
                  s2=str(estruc[2])
                  salida4.write("JAPPRO"+","+sl+","+s2+","+'10'+'\n')
                  num2=num2+1
              else:
                  sl=str(estruc[1])
                  s2=str(estruc[2])
                  salida4.write(s0+","+sl+","+s2+","+'10'+'\n')

  salida4.close()
  salida5.close()
  #gsutil cp gs://bucket_name/Movimientos.txt
  #print('Archivo {} descargado a {}.'.format(source_file_name,bucket))
```

```
import socket

def Main():
    host = '192.168.47.5'
    port = 49197

    entrada = open('Movimientos.txt', 'r')
    linea=entrada.read().splitlines()

    mySocket = socket.socket()
    mySocket.connect((host, port))

    for caracter in linea:
        mySocket.send(caracter.encode())
        data = mySocket.recv(1024).decode()

        print('Received from server: ' + data)

    mySocket.close()
```

```
from gr.pygr import *
import os,sys
import numpy as np
import cv2

def obtener_archivo(archivo1,archivo2):

    entrada=open(archivo1,'r')
    vxy=open(archivo2,'w')

    line=entrada.read().splitlines()
    listax=[]
    for caracter in line:
        estructura=caracter.split(',')
        s0=str(estructura[0])
        if s0 == "END":
            break
        else:

            s1=str(estructura[1])
            s2=str(estructura[2])
            if s1!="0" and s2!="0":
                vxy.write(s1+','+s2+"\n")

    vxy.close()
    entrada.close()
```

```
def dibujar(archivo):

    titulo="Interfaz"
    #os.system("graficaxy.py")
    entrada1=open(archivo,'r')
    linea1=entrada1.read().splitlines()
    x=[]
    y=[]

    for caracter1 in linea1:
        estructura1=caracter1.split(',')
        s0=(estructura1[0])

        s1=(estructura1[0])
        s2=(estructura1[1])
        #print(s1)
        x.append(s1)
        y.append(s2)

    entrada1.close()
    plot(x,y)
```

```
from ftplib import FTP

def grabFile(filename,exitfile):
    salida="RETR"+" "+ exitfile
    ftp=FTP('172.18.108.120')
    ftp.login(user='usuario1')
    ftp.cwd('mis_apps')
    #filename='Output.txt'
    localfile=open(filename,'wb')
    ftp.retrbinary(salida,localfile.write,1024)
    ftp.quit()
    localfile.close()
```

Código escrito en C++ para la cámara industrial

```
#include "stdafx.h"
#include <iostream>
#include <fstream>

#include <pylon/PylonIncludes.h>
#ifdef PYLON_WIN_BUILD
#include <pylon/PylonGUI.h>
#endif

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"


using namespace std;
using namespace cv; //Namespace for opencv objects
using namespace Pylon; // Namespace for using pylon objects
static const uint32_t c_countOfImagesToGrab = 200; // Number of images to be grabbed

int H_min = 22, S_min = 115, V_min = 107; //Declare min and max HSV values
int H_max = 180, S_max = 256, V_max = 256;
const double offset = 50.0; //tcp measure to center the robot effector
const double z_ori = -32.0; //Z value for base pose robot
bool bandera = 0;


string message;
int main(int argc, char** argv)
{
    int exitCode = 0;
    Pylon::PylonAutoInitTerm autoInitTerm;
    CGrabResultPtr ptrGrabResult;

    ofstream archivo,archivo1;
    archivo.open("Input.txt", ios::out);
    archivo1.open("Puntos.txt", ios::out);
    try
    {
        CInstantCamera camera(CTlFactory::GetInstance().CreateFirstDevice());
        cout << "Using device " << camera.GetDeviceInfo().GetModelName() << endl;
        camera.Open();

        GenApi::CIntegerPtr width(camera.GetNodeMap().GetNode("Width"));
        GenApi::CIntegerPtr height(camera.GetNodeMap().GetNode("Height"));
        camera.MaxNumBuffer = 5; //Count of buffers allocated for grabbing (max 10)
        camera.StartGrabbing();

        CImageFormatConverter fc, fc1; //Create a pylon ImageFormatConverter object

        fc.OutputPixelFormat = PixelType_BGR8packed; //Specify the output pixel format
        fc1.OutputPixelFormat = PixelType_BGR8packed; //Specify the output pixel format

        CPylonImage image, image1; //Create a pylonImage that will be used to create a opencv images later
```

```
int iLowH = 0;
int nlowH = 27;

int iHighH = 15;
int nhighH = 80;

int iLowS = 135;
int nlowS = 145;

int iHighS = 210;
int nhighS = 255;

int iLowV = 95;
int nlowV = 50;

int iHighV = 175;
int nhighV = 80;


//captura temporal de imagen de camara
camera.RetrieveResult(5000, ptrGrabResult, TimeoutHandling_ThrowException); //Wait for an image an then retrieve it.
fc1.Convert(image1, ptrGrabResult); //Convert the grabbed buffer to a pylon image
Mat origen = cv::Mat(ptrGrabResult->GetHeight(), ptrGrabResult->GetWidth(), CV_8UC3, (uint8_t*)image1.GetBuffer());

//imagen oscura del mismo tamano de la salida de la camara
Mat imgLines = Mat::zeros(origen.size(), CV_8UC3);


while (camera.IsGrabbing())
{
    camera.RetrieveResult(5000, ptrGrabResult, TimeoutHandling_ThrowException);

    if (ptrGrabResult->GrabSucceeded())
    {
        fc.Convert(image, ptrGrabResult); //Convert the grabbed buffer to a pylon image
        Mat ori_img = cv::Mat(ptrGrabResult->GetHeight(), ptrGrabResult->GetWidth(), CV_8UC3, (uint8_t*)image.GetBuffer());



        Mat imgHSV, imgN;

        cvtColor(ori_img, imgHSV, COLOR_BGR2HSV);
        cvtColor(ori_img, imgN, COLOR_BGR2HSV);

        Mat imgThresholded;
        Mat imgnegro;

        inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV), imgThresholded);
        //  threshold(imgHSV, imgThresholded, 0, 255, THRESH_BINARY);
        erode(imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));
        dilate(imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));

        dilate(imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));
        erode(imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));

        //PARTE DE COLOR NEGRO
        inRange(imgN, Scalar(nlowH, nlowS, nlowV), Scalar(nhighH, nhighS, nhighV), imgnegro);
        //  threshold(imgHSV, imgThresholded, 0, 255, THRESH_BINARY);
        erode(imgnegro, imgnegro, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));
        dilate(imgnegro, imgnegro, getStructuringElement(MORPH ELLIPSE, Size(5, 5)));
```

```
//Calcula los momentos de la imagen color rojo
Moments oMoments = moments(imgThresholded);

double dM01 = oMoments.m01;
double dM10 = oMoments.m10;
double dArea = oMoments.m00;


//Calcula los momentos de la imagen color negro
Moments oMomentsN = moments(imgnegro);

double dM01N = oMomentsN.m01;
double dM10N = oMomentsN.m10;
double dAreaN = oMomentsN.m00;

if (waitKey(30) == 101)
{
    bandera = 1;
}

if (waitKey(30) == 116)
{
    bandera = 0;
}

//si el area es menor a 10000, considerar que no hay objeto en imagen
if (dArea > 10000 && bandera==1)
{
    //calcula la posicion
    int posX = dM10 / dArea;
    int posY = dM01 / dArea;


      if (iLastX >= 0 && iLastY >= 0 && posX >= 0 && posY >= 0)
      {
          //dibuja una linea roja del punto anterior al actual
          line(imgLines, Point(posX, posY), Point(iLastX, iLastY), Scalar(0, 0, 255), 2);
      }
      int tempx, tempy;
      tempx = 0.37*(posX - 636);
      tempy = 0.47*(posY - 445);
      cout << "posicion en x es: " << tempx << endl;
      cout << "posicion en y es: " << tempy << endl;
      int cx, cy;
      cx = -1 * tempx;
      cy = -1 * tempy;
      archivo << tempx << ',' << cy << ',' << 0 << endl;
      archivo1 << cx << ',' << cy << ',' << 0 << endl;
      iLastX = posX;
      iLastY = posY;

      if (archivo.fail())
      {
          cout << "NO SE PUDO ABRIR EL ARCHIVO" << endl;
          exit(1);
      }

  }
  else if (dAreaN > 10000)
  {
      archivo << 0 << ',' << 0 << ',' << 1 << endl;
      archivo1 << 0 << ',' << 0 << ',' << 1 << endl;
  }
```

```
                imshow("Thresholded Image", imgThresholded);
                ori_img = ori_img + imgLines;
                imshow("Original", ori_img);

                if (waitKey(30) == 27)
                {
                    cout << "esc key pressed" << endl;
                    break;
                }

                if (waitKey(30) == 32)
                {
                    imgLines = 0;
                }
            }
        }
    }
    catch (GenICam::GenericException &e)
    {
        cerr << "An exception occurred." << endl << e.GetDescription() << endl;
        exitCode = 1;
    }
    archivo.close();
    //archivo1.close();
    return exitCode;
}
```