UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE GUAYAQUIL

CARRERA DE INGENIERIA ELECTRÓNICA

"DISEÑO E IMPLEMENTACIÓN DE ROBOT MÓVIL MULTITAREA CON VISIÓN ARTIFICIAL Y PROGRAMACIÓN EN LENGUAJE PYTHON PARA LA UNIVERSIDAD POLITÉCNICA SALESIANA"

Trabajo de titulación previo a la obtención del

Título de **Ingeniero Electrónico**

AUTORES:

- **RICHARD KEVIN AGUDO UBE**
- **FRANKLIN PAÚL GÓMEZ ZAMBRANO**

TUTOR:

MSC. MONICA MARIA MIRANDA RAMOS

GUAYAQUIL – ECUADOR

2022

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros **Richard Kevin Agudo Ube** con cédula de identificación N° **0931080352** y **Franklin Paúl Gómez Zambrano** con cédula de identificación N° **0951088194;** manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Guayaquil, 27 de abril del año 2022.

(f)Richard Kevin Agudo Ube
C.I: 0931080352

(f)Franklin Paúl Gómez Zambrano
C.I: 0951088194

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros **Richard Kevin Agudo Ube** con cédula de identificación N° **0931080352** y **Franklin Paúl Gómez Zambrano** con cédula de identificación N° **0951088194**, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del **Proyecto técnico: "Diseño e implementación de robot móvil multitarea con visión artificial y programación en lenguaje Python para la Universidad Politécnica Salesiana"**, el cual ha sido desarrollado para optar por el título de: **INGENIERO ELECTRÓNICO**, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Guayaquil, 27 de abril del año 2022.

(f)Richard Kevin Agudo Ube
C.I: 0931080352

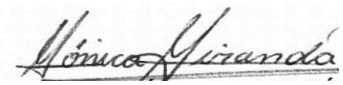
(f)Franklin Paúl Gómez Zambrano
C.I: 0951088194

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo Mónica María Miranda Ramos con documento de identificación N° 0917271785, docente de la **Universidad Politécnica Salesiana**, declaro que bajo mi autoría fue desarrollado el trabajo de titulación: **"DISEÑO E IMPLEMENTACION DE ROBÓT MÓVIL MULTITAREA CON VISIÓN ARTIFICIAL Y PROGRAMACIÓN EN LENGUAJE PYTHON PARA LA UNIVERSIDAD POLITÉCNICA SALESIANA"**, realizado por **Richard Kevin Agudo Ube** con cédula de identificación N° **0931080352** y por **Franklin Paúl Gómez Zambrano** con cédula de identificación N° **0951088194** obteniendo como resultado final el trabajo detitulación bajo la opción **Proyecto técnico**, que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Guayaquil, 27 de abril del año 2022.

Atentamente,



Ing. Mónica María Miranda Ramos, MSC
C.I: 0917271785

DEDICATORIA

Le dedico este proyecto a Dios por darme la fuerza para culminar con mis estudios. A mis padres por apoyarme en cada etapa de este camino. A mis hermanas por apoyarme incondicionalmente. A mis abuelos por sus consejos para lograr ser un excelente profesional y buen ser humano. Por último, agradezco a todas las amistades que me acompañaron en estos años de universidad.

Richard Agudo

DEDICATORIA

Dedico este proyecto a mis padres que me han dado su apoyo incondicional desde siempre. A mi hermana, que cuando no tenía suficientes recursos ella me ayudaba con aquello. A mi abuelo que en paz descanse, por haberme enseñado lo que es la vida. A mi novia, mi compañera infaltable, siempre dándome apoyo moral en cada circunstancia que me encontraba. Y sobre todo a aquellos buenos compañeros que hice en estos años universitarios.

Franklin Gómez

AGRADECIMIENTO

Quiero empezar agradeciéndole a Dios por haberme dado valentía y sabiduría para tomar buenas decisiones en cada obstáculo que se me presentaba en esta etapa. A mis padres por su apoyo emocional y económico, por brindarme sus mejores consejos para llegar a culminar mis estudios, me alegro mucho aún poder contar con el apoyo de ellos.

A mis hermanas que, gracias a ellas, cada mal momento que llegaba a tener en el camino, siempre estaban dispuestas a ayudarme. A mis abuelos por inculcarme buenos valores y siempre enseñarme que la familia es lo más importante. Finalmente, a todos mis amigos que estuvieron conmigo en todo este trayecto.

Richard Agudo

AGRADECIMIENTO

Las palabras de agradecimiento quedan cortas, parece que fue ayer cuando faltaba un día para mi primer día en la universidad, estaba nervioso pero mis padres estaban ahí aconsejándome, dándome buenos ánimos para que todo saliera bien. Siempre estaré agradecido con ellos por su inmenso apoyo incondicional desde el primer día hasta el último día, cuantas veces se sacrificaron sólo por brindarme una buena educación, y estoy muy orgulloso por tenerlos aun a mi lado.

Agradezco a mi hermana, mucho más que eso ha sido como una segunda madre ya que ella siempre ha estado ahí para mí, y a mi cuñado que cuando me faltaba sustento económico para pagar mis pensiones, ellos me apoyaban de vez en cuando y estaré siempre en deuda con ellos.

También agradezco a mi abuelo, que en paz descanse, por haberme enseñado tantas cosas durante la pandemia, son tantas anécdotas que tengo con él, me dio consejos, lecciones de vida. Estoy y estaré siempre orgulloso del buen hombre que fue.

A mi novia, ella ha sido mi compañera infaltable en cada momento, le agradezco por siempre estar ahí escuchándome y dándome apoyo moral y sobre todo buenos consejos para mi día a día.

No podía faltar los buenos compañeros que hice en mi vida universitaria, cuantas veces recibí el apoyo de ellos en distintas actividades y talleres, obviamente este apoyo fue recíproco y agradezco por su gentileza y sobre todo por su compañerismo y grata amistad.

Franklin Gómez

RESUMEN

AÑO	ALUMNOS	DIRECTOR DE PROYECTO	TEMA DE PROYECTO DE TITULACIÓN
2022	AGUDO UBE RICHARD KEVIN GÓMEZ ZAMBRANO FRANKLIN PAÚL	MSC. MONICA MARÍA MIRANDA RAMOS	"DISEÑO E IMPLEMENTACION DE ROBÓT MÓVIL MULTITAREA CON VISIÓN ARTIFICIAL Y PROGRAMACIÓN EN LENGUAJE PYTHON PARA LA UNIVERSIDAD POLITÉCNICA SALESIANA"

El presente proyecto consta de diseñar e implementar un prototipo robótico móvil multitarea con visión artificial y programación en lenguaje Python para la Universidad Politécnica Salesiana, basado en la problemática por la falta de prototipos robóticos para el desarrollo de prácticas estudiantiles que sean aplicadas para hacer pruebas sobre nuevas tecnologías a un equipo o sistema en especial dentro de los laboratorios de la universidad.

El proyecto consta de varias etapas que reúnen habilidades que se adquieren a lo largo de la carrera universitaria, en este caso de la carrera de ingeniería electrónica. Se plantea una solución práctica a través del diseño de un prototipo robótico móvil con un sistema de visión artificial multitarea. Consta de una cámara que se conecta a un computador que dispone de algoritmos en el lenguaje de programación Python para la detección y control por medio de visión artificial, como también una tarjeta microcontroladora programada en C++ para el manejo de los encoder, que da información a los algoritmos de posicionamiento y también que se comunique de manera telemétrica, donde la transmisión es bidireccional entre el prototipo y el computador, teniendo una visualización en su nivel de carga. Al momento de llegar a un nivel asignado en el voltaje, el robot autónomamente debe ir a una zona asignada para su recarga, en el GUI se observa el nivel de batería, la distancia recorrida, el ángulo respecto al plano y su velocidad. Se desarrolló una placa PCB con un microprocesador, específicamente el chip Atmega 328 y un driver de potencia para el control de los motores.

Este proyecto se logró desarrollar a lo largo de 5 prácticas que complementa la enseñanza en la materia de teoría de control a los estudiantes y puedan tener un entendimiento cualitativo y recíproco de cómo realizarlo por medio de Python.

Palabras Claves: Visión Artificial, Python, Encoder, Telemétrica, Robótica, Multitarea, Bidireccional, Microprocesador, Driver.

ABSTRACT

YEAR	STUDENTS	PRJ. DIRECTOR	SUBJECT
2022	AGUDO UBE RICHARD KEVIN GOMEZ ZAMBRANO FRANKLIN PAUL	MSC. MONICA MARÍA MIRANDA RAMOS	"DESIGN AND IMPLEMENTATION OF A MULTITASKING MOBILE ROBOT WITH ARTIFICIAL VISION AND PROGRAMMING IN PYTHON LANGUAGE FOR THE SALESIAN POLYTECHNIC UNIVERSITY"

This project consists of designing and implementing a multitasking mobile robotic prototype with artificial vision and programming in Python language for the Salesian Polytechnic University, based on the problem of the lack of robotic prototypes for the development of student practices that are applied to test new technologies or algorithms specific to a special team or system within the laboratories of the university.

The project consists of several stages that bring together skills that are acquired throughout the university career, in this case the electronic engineering career. A practical solution is proposed through the design of a mobile robotic prototype with a multitasking artificial vision system. It consists of a camera that is connected to a computer that has algorithms in the Python programming language for detection and control through artificial vision, as well as a microcontroller card programmed in C++ for managing the encoders, which gives information to the positioning algorithms and also that it communicates telemetrically, where the transmission is bidirectional between the prototype and the computer, having a visualization of its load level. At the time of reaching an assigned voltage level, the robot must autonomously go to an assigned area for recharging, in the GUI the battery level, the distance traveled, the angle with respect to the plane and its speed are observed. A PCB board was developed with a microprocessor, specifically the Atmega 328 chip, and a power driver to control the motors.

This project was developed through 5 practices that complements the teaching of control theory to students and they can have a qualitative and reciprocal understanding of how to do it through Python.

Keywords: Artificial Vision, Python, Encoder, Telemetry, Robotics, Multitask, Bidirectional, Microprocessor, Driver.

ABREVIATURAS

GUI: Graphical User Interface.

PCB: Printed Circuit Board.

INDICE GENERAL

INDICE DE FIGURAS

INTRODUCCION

La Universidad Politécnica Salesiana como centro de estudios universitarios posee laboratorios de punta en todas sus áreas, que tiene herramientas para la formación continua en cada materia, entre la teoría y la práctica dentro de sus instalaciones hasta poder obtener el título de ingeniería. En este caso se plantea el tema de VISIÓN ARTIFICIAL Y PROGRAMACIÓN EN LENGUAJE PYTHON PARA LA UNIVERSIDAD POLITÉCNICA SALESIANA que consiste en el diseño de un módulo de 5 prácticas, el cual funciona usando varios conocimientos que se aprenden a lo largo de la carrera, haciéndolo así ideal para el enfoque del estudiante.

Muchas veces al tomar una materia, se enfoca en el tema en específico que se trata en esta, y a veces se ve en la dificultad de unir conceptos mientras se avanza en el estudio. Este prototipo aplica el tema de la visión artificial a través de una cámara como método de visualización del sistema, el manejo de un microcontrolador para la adquisición de datos y mediante el uso del lenguaje en programación Python, se logra la integración de todos los componentes previamente dichos. Luego se diseña una tarjeta electrónica PCB para la conexión del prototipo con la computadora por telemetría, en este caso pues con un módulo bluetooth, explicando cómo se encuentran las conexiones hechas para su funcionamiento.

El presente trabajo de investigación se divide en cinco capítulos. El capítulo I describe el problema de investigación, así como la importancia, alcance y delimitación, planteando los objetivos a cumplirlos. El capítulo II profundiza los fundamentos teóricos, sentando las bases para la experimentación del prototipo. El capítulo III contiene todos los procedimientos utilizados para llevar a cabo el proyecto, es decir la metodología que se utilizó para resolver la problemática del proyecto. El capítulo IV se presentan los resultados del proyecto, se visualiza cada una de las prácticas que tenemos establecida para el aprendizaje del usuario. Finalmente, en el capítulo V se realizó un análisis profundo de cada una de las prácticas que gobierna el proyecto y a su vez se pudieron deducir los resultados obtenidos del mismo.

1. EL PROBLEMA

1.1. Descripción del problema.

Se evidencia que muchas veces en laboratorios, existe una grave problemática por falta de prototipos robóticos que se apliquen para hacer pruebas sobre nuevas tecnologías, programas o algoritmos que muchas veces son muy específicos a una marca o equipo en especial. Para poder solucionar esta problemática, se plantea una solución práctica a través del desarrollo e implementación de un prototipo robótico como además un sistema de visión artificial multitarea, donde se pueda aprender varias etapas del desarrollo para lograr un robot comercial o a su vez aplicación a nivel industrial.

1.2. Antecedentes.

El presente proyecto técnico complementa los conocimientos adquiridos en el aula para el desarrollo del aprendizaje de los estudiantes. Provee de distintas herramientas y aplicaciones para implementar diversos sistemas donde se requiera el uso de la visión artificial. Es portátil y de replicación versátil, que sienta un parecido a los robots móviles manejados mediante los distintos teléfonos celulares por medio de aplicaciones o kits de otras marcas.

La creación de herramientas didácticas para desarrollar prácticas académicas busca mejorar los enfoques y conocimientos teóricos como prácticos de los estudiantes en diferentes campos, con destreza profesional y experiencia relevante a aras de aplicarlo e implementarlo en las industrias.

1.3. Importancia y alcances.

El tema que se desarrolló, muestra lo fundamental que es contar con diferentes prototipos para que en los laboratorios se puedan realizar prácticas y poder visualizar los resultados. También favorece de manera significante, ya que, cada estudiante puede realizar prueba y error de cada una de sus simulaciones y, de esta manera ir adquiriendo conocimientos sobre nuevas tecnologías como lo es la visión artificial, mejorando así, su manera de ejecutar programas. Con estos precedentes, la comunidad estudiantil puede destacarse en el campo de visión artificial junto con proyectos industriales. Por último, pueden ir puliendo su técnica, para que, en un futuro puedan explanar proyectos de titulación, relacionada con estas nuevas tecnologías.

1.4. Delimitación del problema.

1.4.1. Temporal.

La implementación de este proyecto se realizó en el periodo lectivo número 58, entre los años 2021 y 2022.

1.4.2. Espacial.

El proyecto se desarrolló para ser usado en las instalaciones de la Universidad Politécnica Salesiana. sede Guayaquil, campus Barrio Cuba, en Chambers 227 y 5 de junio. En la Figura 1 se puede apreciar la vista superior de la institución.

Figura 1. Vista superior del Campus Guayaquil.

1.4.3. Académica.

Para la implementación del presente proyecto se aplicaron conocimientos teóricos y prácticos adquiridos en las materias CAD, Microcontroladores I, Teoría de Control y además Electiva (Robótica). También se utilizó el lenguaje de programación Python por su creciente popularidad en desarrollo de proyectos, visión artificial que es una tecnología que utiliza y procesa las imágenes adquiridas mediante una cámara para el reconocimiento de diversos colores, donde se puede ubicar el robot en el plano y darle distintas tareas, entre ellas ir a un punto de recarga.

1.5. Objetivos.

1.5.1 Objetivo General.

Diseñar e implementar un robot móvil multitarea con visión artificial y programación en lenguaje Python para que sea ubicado en el Laboratorio de Sistemas de Control Automático de la Universidad Politécnica Salesiana sede Guayaquil.

1.5.2 Objetivo Específicos.

- Diseñar e implementar una tarjeta PCB para un robot multitarea con telemetría.
- Desarrollar un prototipo de robot móvil en CAD fabricando sus piezas en corte CNC e impresión 3D.
- Implementar un sistema de carga automática para el robot móvil y este pueda recargarse autónomamente.
- Elaborar 5 prácticas para aprendizaje e implementación de visión artificial y control del prototipo.

2. MARCO TEORICO REFERENCIAL

2.1. Python.

Es un lenguaje de programación moderno creado por Guido van Rossum a principios de los 90. La implementación de la especificación, conocida como Python, está cubierta por una licencia de software gratuita y descargable desde el sitio web oficial. El hecho es que la tecnología libre y abierta tiene una ventaja significativa sobre la tecnología propietaria. La conclusión es que puede usarlo sin pagar regalías. Esto significa que los estudiantes aún pueden usar Python de forma gratuita fuera del Tecnológico de Monterrey, por ejemplo, para escribir software en un entorno comercial o para realizar estudios de posgrado en otras universidades. En la Figura 2 se puede observar el logo de Python actualmente (Ortiz Ramirez, 2010).

Figura 2. Logo del lenguaje de programación Python (Python Software Foundation, s.f.).

2.1.1. Tkinter.

El paquete tkinter es la parte de interfaz en Python, con esto se puede lograr diseñar el panel de control y añadir lo que sea necesario para un buen funcionamiento de lo que se requiera. Tk y tkinterest son accesibles en las plataformas Unix, así como en los sistemas Windows.

Al iniciar el proceso desde la línea de comandos, aparece una ventana, en la cual, se muestra una interfaz en Tk simple, esto permite saber que se instaló de una manera correcta en nuestro sistema y muestra la versión de Tcl/Tk que está instalada para que pueda leer documentos específicos de Tcl/Tk. En la Figura 3 se visualiza un ejemplo de aplicación de Tkinter (Python Software Foundation, 2021).

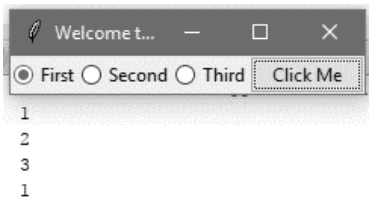


Figura 3. Ejemplo de interfaz gráfica con librería Tkinter (Ebrahim, 2018).

2.1.2. Pillow.

La biblioteca de imágenes de Python, llamada PIL, se utiliza para cargar y mantener imágenes de bibliotecas de código abierto.

Previo a desarrollar modelos predictivos para datos de imágenes, debe aprender a cargar imágenes y manipularlas. La biblioteca Python estándar más popular para la carga de imágenes y el procesamiento de datos de almohadas (Brownlee, 2020).

2.1.3. Imutils.

Es un conjunto conveniente de funciones que admiten funciones básicas de procesamiento de imágenes, como traducción, rotación, cambio de tamaño, encuadre, representación de imágenes de Matplotlib, alineación de contornos, detección de bordes y más. utilizando OpenCV y Python (Python Software Foundation, 2021).

2.1.4. Pyserial.

Este módulo proporciona acceso al puerto serie. Proporciona backends para Python que se ejecutan en Windows, OSX, Linux, BSD (puede ser cualquier sistema compatible con POSIX) e IronPython. El módulo llamado "serie" selecciona automáticamente el backend apropiado y tiene una licencia gratuita. En la Figura 4 se detalla el logo de la librería a usar (Liechti, 2018).

Figura 4. Logo de la librería Pyserial (Liechti, 2018).

2.1.5. OpenCV.

La biblioteca OpenCV proporciona un marco de alto nivel para crear aplicaciones de visión artificial en tiempo real: estructuras de datos, procesamiento y análisis de imágenes, análisis estructural y más. Esta estructura facilita significativamente el aprendizaje y la implementación de diversas técnicas de visión artificial a nivel didáctico y de investigación, separando a los programadores de las propiedades de los sistemas de visión diferentes. En la Figura 5 se puede visualizar el logo de la librería a utilizar (V. M. Arévalo, 2004).

Figura 5. Logo de la librería OpenCV (OpenCV team, s.f.).

2.1.6. NumPy.

En Python, las matrices NumPy son la representación estándar de datos numéricos y permiten la implementación eficiente de operaciones numéricas en lenguajes de alto nivel. Como muestra este trabajo, el rendimiento de NumPy se puede mejorar mediante tres técnicas: vectorizar el cálculo, evitar copiar datos en la memoria y minimizar el número de operaciones. (van der Walt, Colbert, & Varoquaux, 2011).

2.1.7. Os-System

Esta librería permite ejecutar comandos en forma de cadena de texto, además de poder indicar la ruta del archivo como una variable. (Mosquera De la Cruz, y otros, 2020)

2.2. Visión Artificial.

La visión artificial ayuda a realizar funciones como, por ejemplo, inspección visual, detección de objetos, también sirve para medición de objetos y a su vez la colocación de estos mismos. Además, de aplicaciones más conocidas como identificar, rastrear diferentes productos y clasificar según colores o tamaños. En la Figura 6 se puede observar el proceso que realiza (COGNEX, 2018).

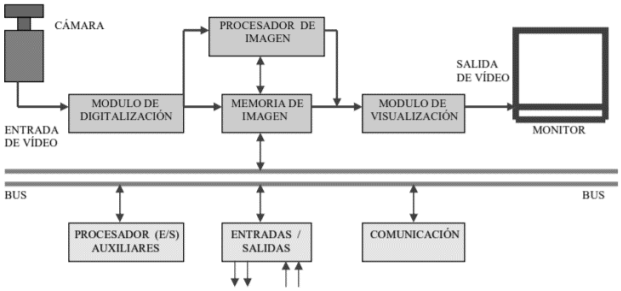


Figura 6. Diagrama de bloques de un sistema de visión artificial (ARTIFICIAL).

2.2.1. Sensor óptico.

El sensor puede ser una cámara a color o monocromática que produce una imagen completa del dominio del problema cada 0.0333 segundos. El sensor también puede ser una cámara de escaneo que genera una línea en cualquier momento. En este caso, el movimiento del sujeto a lo largo de la línea de exploración crea una imagen bidimensional. En la Figura 7 se aprecia el tipo de cámara que se empleó para el proyecto, en este caso se trata de una cámara USB de la marca Logitech de 720p (González Marcos, y otros, 2006).

Figura 7. Cámara USB Logtitech HD C720 (González Marcos, y otros, 2006).

2.2.2. Tarjeta de adquisición de imagen.

Permite digitalizar la señal de video entregada por el subsistema anterior (González Marcos, y otros, 2006).

2.2.3. Computador.

Ya obteniendo varias imágenes se las ubica en la memoria del computador, para que de esta manera continúen con su procesamiento y, a su vez, para que estas puedan ser manipuladas con el software (González Marcos, y otros, 2006).

2.2.4. Monitor de vídeo.

Permite ver las imágenes o escenas capturadas y los resultados de su procesamiento (González Marcos, y otros, 2006).

2.3. Atom.

Atom es un editor de texto hackeable creado por GitHub y construido en la plataforma de aplicaciones de escritorio Electron. Esto quiere decir que puede usarse desde un editor de texto para programación básica hasta un ID en su totalidad. También es altamente personalizable y ofrece miles de paquetes creados por la comunidad y temas para satisfacer todas sus necesidades. En la Figura 8 se visualiza el logo del software mencionado (Stack Overflow, 2018).

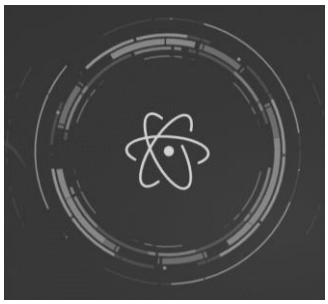


Figura 8. Logo de Atom (Atom, s.f.).

2.4. Telemetría.

Los sistemas de telemetría son los encargados de adquirir datos sobre las magnitudes físicas, procesarlas y, por último, trasladarlas en forma de datos de forma inalámbrica hacia el encargado de monitorear el sistema, donde se analiza la información para realizar optimizaciones en el sistema monitoreado en caso de ser necesarias (Cornejo Ortega & Tintin Suquilanda, 2010).

2.5. CMOS Image Sensor.

Es una tecnología utilizada hoy en día en la mayoría de los dispositivos móviles o cámaras digitales e implica el procesamiento de cada píxel, lo que hace que los voltajes individuales pasen a través de la matriz hasta el final a bajo costo y bajo consumo de energía. Las matrices ofrecen procesamiento de imágenes verticales u horizontales en píxeles, en lugar de cuadro por cuadro. En la Figura 9 se puede observar la compasión con ambos tipos de escaneado (Grassvalley, 2012).

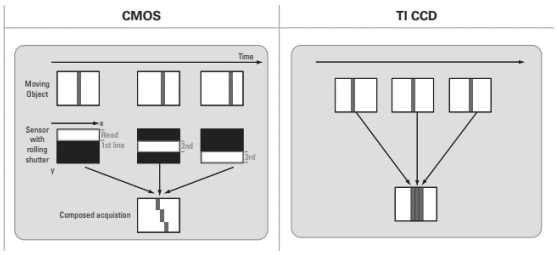


Figura 9. Escaneada línea a línea comparado a escaneado frame por frame (Grassvalley, 2012).

2.6. Interfaz USB.

Es un estándar de conectividad simple, topología en estrella, con protocolos que pueden descubrir y configurar automáticamente los dispositivos conectados. Consta de dos pares de cables, uno para transmisión de datos y otro para transmisión de energía. La velocidad de transferencia de datos depende del tipo de USB, el 2.0 se hizo popular en 2000 (S. Carballas, 2018).

2.7. Receptor de carga inalámbrico “Qi”.

Esta es una placa receptora de carga inalámbrica universal para teléfonos inteligentes Android que encaja perfectamente en el conector micro USB en la parte inferior, como se muestra en la Figura 10. Su diseño ultra delgado y duradero facilita la instalación de la placa receptora de carga. Este cargador permite que cargue de forma inalámbrica su teléfono inteligente Android.

Figura 10. Receptor de carga inalámbrico.

2.8. Módulo cargador de batería litio TP4056 micro-USB.

Hoy en día, la batería de litio, también conocida como batería de iones de litio, es una de las tecnologías más utilizadas. Consiste en una batería que contiene al menos dos celdas separadas de iones de litio. Cuando una batería opera en modo descarga, los iones de las dos se combinan químicamente para formar un elemento estable, y esta combinación es exotérmica, es decir, produce la energía a utilizar. Cuando la batería se agota es porque todos los iones están en estado fundamental. En el siguiente espacio se muestra la Figura 11, el cual es el módulo que se utilizó (Solís Cascante, 2018).

Figura 11. Módulo cargador de batería litio (Naylamp Mechatronics , 2021).

2.9. Regulador Boost Step-Up MT3608 con entrada micro-USB.

Es un módulo altamente eficiente y de bajo presupuesto. Está diseñado para convertir el voltaje desde 2V hasta un máximo de 28V con una corriente de salida máxima de 2 amperios. El voltaje que se obtiene en la salida se lo puede ajustar con la resistencia variable que está incluida en dicho módulo.

Este módulo contiene una cantidad muy baja de componentes y, el componente principal de este módulo es MT3608 IC. El MT3608 es un circuito integrado de refuerzo de voltaje altamente eficiente con muchas características integradas como protección contra sobrecalentamiento, corriente de reposo baja, función de arranque suave, componentes externos bajos y apagado por sobrecalentamiento. En la Figura 12 se visualiza el regulador a emplear (AEROSEMI, 2022).

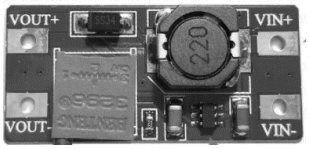


Figura 12. Regulador Boost Step-Up MT3608 (Components info).

2.10. Convertidor de voltaje DC-DC Step-Down LM2596.

Los reguladores de la serie LM2596 son circuitos integrados monolíticos que brindan toda la funcionalidad activa de los reguladores de conmutación reductores que pueden impulsar cargas 3A con una excelente regulación de línea y carga.

Estos dispositivos están disponibles en voltajes de salida fijos de 3,3 V, 5 V y 12 V y versiones de salida ajustable. Estos controles requieren componentes externos mínimos, son fáciles de usar y tienen compensación de frecuencia interna y un oscilador de frecuencia fija. En la Figura 13 se observa el componente que se empleó (Texas instruments, 1999).

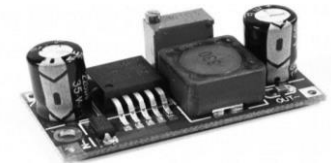


Figura 13. Convertidor Buck Step-Down LM2596 (Naylamp Mechatronics, 2021).

2.11. Modulo Bluetooth HC-05.

El módulo Bluetooth HC05 es uno de los dispositivos más populares para agregar capacidades de comunicación Bluetooth a Arduino o a otros microcontroladores. Estos son dispositivos relativamente económicos que se pueden conectar directamente a la placa prototipo y a cualquier microcontrolador sin soldar.

El módulo HC05 tiene dos modos de funcionamiento, modo maestro-esclavo. Esto significa que no solo puede actuar como un esclavo esperando órdenes de su computadora o teléfono inteligente, sino que también puede actuar como un maestro y cuidar de sí mismo. Conecte un dispositivo Bluetooth a otro dispositivo. La Figura 14 muestra el componente que se empleó (Biendicho Lletí, 2015).

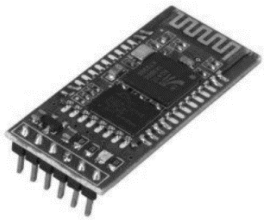


Figura 14. Módulo Bluetooth HC-05 (Biendicho Lletí, 2015).

2.12. Motor CHR-GM25-370.

Este motor de CC con codificador y caja de engranajes es perfecto para el control de posición de bucle cerrado o de velocidad como: péndulo invertido, robot móvil autónomo, servomotor de CC, transportador y más. Este dispositivo consta de tres partes: el motor DC, la caja reductora y el encoder de cuadratura. Los motores de CC funcionan con un voltaje nominal de 12 V. La caja reductora de metal funciona para reducir la velocidad de entrada y aumentar el par de salida. En la Figura 15 se muestra la estructura del encoder (Naylamp, 2021).

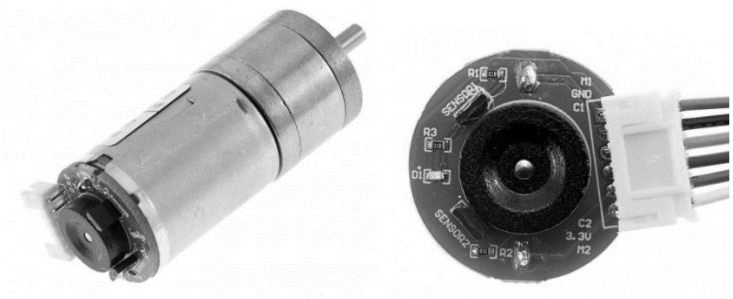


Figura 15. Motor DC con caja reductora y encoder 25GA (Naylamp, 2021).

2.13. Arduino Nano.

Arduino Nano es una placa de diseño clásica y minimalista compatible con el IDE de Arduino. Este microcontrolador viene con pines para facilitar la conexión a la placa de pruebas y tiene un conector USB Mini-B. En la Figura 16 se muestra el microcontrolador que se ocupó (Arduino, 2022).

Figura 16. Arduino Nano (Kurniawan, 2019).

2.14. Puente H L293D.

El dispositivo es un controlador monolítico integrado de cuatro canales de alto voltaje y alta corriente diseñado para admitir niveles lógicos DTL o TTL estándar y controlar cargas inductivas como relés, solenoides, CC y relés, motores paso a paso y transistores de potencia de conmutación. Proporciona una entrada de alimentación independiente para la lógica, lo que permite el funcionamiento a voltajes más bajos y en combinación con diodos terminales internos. El dispositivo es adecuado para aplicaciones de conversión de hasta 5 kHz. En la Figura 17 se visualiza el driver que se utilizó para el proyecto (STMicroelectronics, 1996).

Figura 17. Driver de baja potencia para motores L293D (Nomada, 2016).

2.15. Buzzer.

Un zumbador o comúnmente conocido como buzzer es un pequeño sensor que convierte la energía eléctrica en sonido. Para que funcione, todo lo que necesita hacer es conectar el positivo y el negativo a una batería o cualquier fuente de CC. En la Figura 18 se observa el zumbador que se utilizó en el proyecto (La enciclopedia de la ingeniería, 2018).

Figura 18. Componente electrónico buzzer (La enciclopedia de la ingeniería, 2018).

2.16. Diodo Led

Un LED es un dispositivo que permite que la corriente fluya en una dirección y emite un haz de luz polarizada. Funciona como un diodo regular, pero brilla cuando recibe corriente. Los diodos funcionan a unos 2V. Para conectarlos con diferentes voltajes necesitas usar resistencias. En la Figura 19 se puede apreciar la estructura del componente (Déleg, 2010).

Figura 19. Estructura de un LED (Déleg, 2010).

2.17. Batería recargable.

Estas se denominan baterías de litio o baterías de iones de litio. Son muy ligeros, tienen una gran capacidad energética y son resistentes a la descarga. No tienen efecto mnemotécnico y sus principales desventajas son su mayor costo en comparación con el níquel, su rápida descomposición y su alta sensibilidad a las altas temperaturas que pueden dañarlos. En la Figura 20 se muestra la batería que se utilizó para este proyecto (Ferrer, 2020).

Figura 20. Batería recargable Li-ion de 4000mAh (Ferrer, 2020).

2.18. Impresión 3D.

Este es un grupo tecnológico fundado en 1984 y consiste en construir objetos de menor escala capa por capa utilizando resinas ópticas líquidas. El progreso de esta tecnología fue lento hasta que salió la primera impresora de código abierto en el Reino Unido en 2014. Desde entonces, la impresión 3D se ha acercado más al consumidor, lo que le ha ayudado a crecer hasta el punto de que hoy en día se reproducen réplicas de automóviles e incluso casas pequeñas. construido a escala natural. En la Figura 21 se visualiza la impresora con la que se generaron los soportes de los motores del prototipo (Gil, 2015).

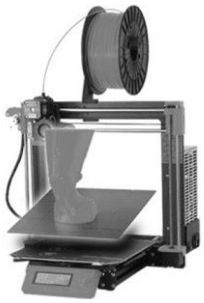


Figura 21. Impresora Prusa I3 MK2S (3Dnatives, 2022).

Estas impresoras utilizan dos métodos para crear objetos, que pueden ser la compactación capa por capa de bloques de polvo o la inyección de polímero capa por capa. Su estructura consiste en una tarjeta de control electrónico, muy probablemente un Arduino MEGA, con la adición de SHIELDS y otros componentes como una pantalla LCD, un lector de tarjetas SD y un panel de control de botones; es importante elegir el firmware a utilizar, algunas alternativas podrían ser Sprinter, Marlin. (Lambán & Domínguez, 2013).

2.19. Corte por láser.

Esta técnica digital de grabado de materiales mediante un láser. Emplea tecnología CNC en la que un ordenador traza un camino de corte y hace que el láser se desplace sobre el material, y contaba con una pérdida pequeña de 0,5 mm. En la Figura 22 se visualiza un corte de láser sobre acrílico (Sculpteo, 2020).

Figura 22. Corte realizado por láser en acrílico (Salvador, 2019)

Una de sus principales ventajas es la versatilidad en cuanto a la cantidad de materiales que puede cortar, esto nos beneficia, ya que, se generan menos gastos en logística en cuanto a posteriores ya que casi en todos los materiales el calor del láser además de cortar, los sella y es de alta precisión (TROTEC, 2020).

2.20. EasyEDA.

Es una aplicación muy poderosa de diseño esquemático, diseño de PCB y simulación de circuitos mixtos. Estos diseños son fáciles de implementar gracias a la biblioteca de componentes que contienen. Es una aplicación súper completa para la creación de esquemas eléctricos. Se puede visualizar en la Figura 23 el logo del software mencionado (Bastero Huarte, 2018).

Figura 23. Logo del software EasyEDA (EasyEDA, 2021).

2.21. Autodesk Inventor

Autodesk Inventor permite diseñar un prototipo ya sea en 3D o en 2D, con la finalidad de visualizar las diferentes vistas del proyecto. Se puede ajustar a criterio de cada usuario todo el producto que se realice. Este software agiliza los procesos de trabajo en equipo o individuales. En la Figura 24 se muestra el logo del software Inventor (Sanson & Dalila, 2022).

Figura 24. Logo del software Autodesk Inventor (Sanson & Dalila, 2022).

2.22. Circuito Impreso.

Entre los programas para el diseño de tarjetas electrónicas en placas de circuitos impresos se encuentran AutoDesk Eagle, Labcenter Electronics Proteus los cuales son muy potentes y competitivos en el mercado, pero también requiere licencia para su uso comercial. Para lograr los objetivos del proyecto, se utilizó un software de diseño con licencia libre llamado EasyEDA. Se puede observar el Figura 25 el circuito impreso del proyecto de titulación (J. Charras, 2020).

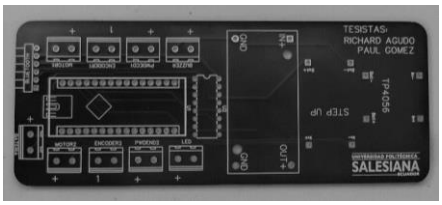


Figura 25. Circuito impreso para el proyecto de titulación.

2.23. Robótica.

La robótica ahora se puede considerar como uno de los campos de la tecnología que se está desarrollando más dinámicamente, basado en el estudio de la robótica, es decir, el sistema de mecanismos que les permite utilizar conceptos de los campos del conocimiento para navegar y realizar acciones específicas, programables y accionables. Tareas inteligentes, como electrónica, mecánica, física, matemáticas, electricidad e informática.

Dependiendo de la aplicación, los robots pueden extenderse no solo a la industria y los servicios, sino también a las aulas, lo que permite el desarrollo de entornos de aprendizaje innovadores (Pinto Salamanca, Barrera Lombana, & Pérez Holguín, 2010).

2.24. Controlador PID.

El controlador PID (Proporcional, Integral y Derivativo) es un controlador de retroalimentación diseñado para producir un error de estado estable de cero asintótico entre la señal de referencia y la salida del dispositivo a lo largo del tiempo logrado de manera integrada. Además, el auditor tiene la capacidad de predecir el futuro a través de operaciones derivadas que llevan a predecir el resultado del proceso. El controlador PID es suficiente para resolver problemas de control en muchas aplicaciones industriales, especialmente cuando la dinámica de los procesos lo permite, con requerimientos de desempeño moderados y respuesta rápida a los cambios en la señal de referencia. En la Figura 26 se puede visualizar cómo funciona un sistema de control (Améstegui Moreno, 2001).

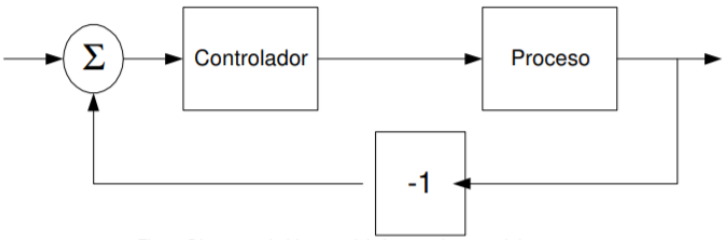


Figura 26. Diagrama de bloques del sistema de control de un proceso (Améstegui Moreno, 2001).

2.24.1. Método de Sintonía Lambda

El método de sintonía lambda es ampliamente usado en la industria de pulpa y papel, donde hay una conexión entre la uniformidad del papel y la eficiencia manufacturera.

Cuando fue introducido por Dahlin en 1968, el ajuste de lambda ofreció una nueva forma de coordinar la sintonía de los bucles de la fábrica de papel para ganar estabilidad del proceso junto con un producto uniforme.

Para la obtención del diseño PID, considere la función de transferencia PID serie o interactuante:

$$C'(s) = K'_c\left(\frac{(1 + sT'_i)(1 + sT'_d)}{sT'_i}\right)$$

Donde:

K'_c= Ganancia Proporcional

T'_i= Tiempo Integral

T'_d= Tiempo Derivativo

Este método consiste en cancelar los polos del proceso con los ceros del controlador. A diferencia de los demás métodos tradicionales, el método de sintonía lambda logra una estabilización sin presentar sobre impulsos, a pesar que conlleva más tiempo, pero es mucho más eficiente que los otros métodos como López (IAE), Haalman y Ziegler y Nichols. A continuación, en la Figura 27 se muestra un ejemplo de un sistema con diferentes tipos de respuesta con cada método (Pruna, Edison, & Mullo , 2017).

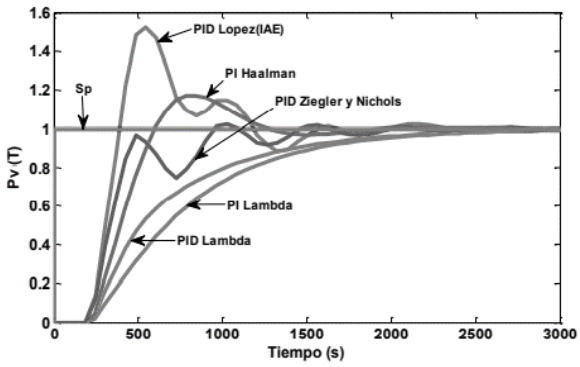


Figura 27. Comparación de respuesta de un sistema con diferentes métodos de sintonía (Pruna, Edison, & Mullo , 2017).

2.25. Coordenadas de la cámara.

El sistema de referencia de las coordenadas de la cámara se planteó de la siguiente manera, se tiene el eje X que está en la parte superior y se incrementa hacia la derecha, mientras que el eje Y está en la parte inferior y se incrementa hacia abajo, tal cual como se muestra en la Figura 28 referenciando un plano cartesiano.

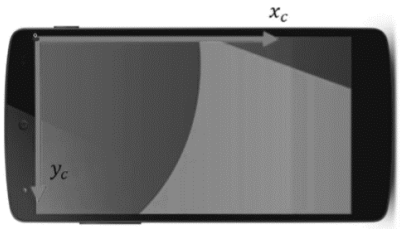


Figura 28. Referencia de coordenadas en el plano X y Y.

El prototipo desarrollado lleva un similar sistema de referencia a como se observó en el párrafo anterior, en este caso el eje X se mantiene, mientras que el eje Y se incrementa para arriba, tal como se muestra en la Figura 29.

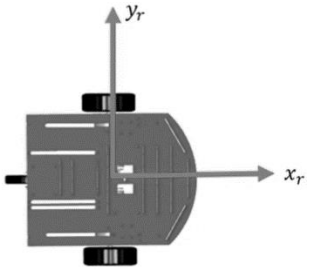


Figura 29. Referencia del robot en el plano X y Y.

Entonces, para el sistema anterior se determina que Xr es igual a:

$$x_r = x_1 + \alpha * cos\varphi \qquad (2)$$

Por consiguiente, su derivada viene a ser:

$$\dot{x_r} = \mu * cos\varphi - \alpha\omega * sen\varphi \qquad (3)$$

Y para el caso de Yr, es igual a:

$$y_r = y_1 + \alpha * sen\varphi \qquad (4)$$

Por consiguiente, su derivada viene a ser:

$$\dot{y}_r = \mu * sen\varphi + \alpha\omega * cos\varphi \qquad (5)$$

Llevando esto a una matriz 2x2, se tiene como resultado la siguiente agrupación:

$$\begin{bmatrix}\dot{x}_r \\ \dot{y}_r\end{bmatrix} = \begin{bmatrix}cos\varphi & \alpha sen\varphi \\ sen\varphi & \alpha cos\varphi\end{bmatrix} * \begin{bmatrix}u \\ \omega\end{bmatrix} \qquad (6)$$

Pero con los dos sistemas de referencia explicados recientemente, se observa que no coinciden y eso se aprecia en la Figura 30.

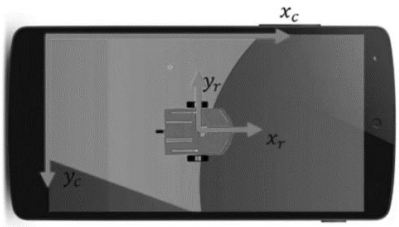


Figura 30. Sistemas de referencias unidos.

Para ello se requiere girar el robot 90°, de esta manera se puede llevar los dos sistemas a uno solo, tanto el de la cámara como el del robot. En la Figura 31 se aprecia el robot girado 90°, logrando así un solo sistema de referencia.

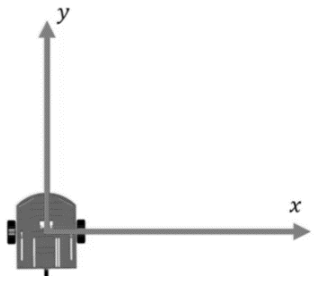


Figura 31. Robot móvil girado 90°.

Como el robot está girado 90°, se debe aplicar dichos grados a las derivadas anteriormente vistas:

$$\dot{x}_r = \mu * \cos\left(\varphi + \frac{\pi}{2}\right) - \alpha\omega * sen\left(\varphi + \frac{\pi}{2}\right) \qquad (7)$$

$$\dot{y}_r = \mu * sen\left(\varphi + \frac{\pi}{2}\right) + \alpha\omega * cos\left(\varphi + \frac{\pi}{2}\right) \qquad (8)$$

Y al final se puede agrupar dichas expresiones en una matriz 2x2; entonces este sistema se debe agregarlo más adelante en la programación del prototipo:

$$\begin{bmatrix}\dot{x}_r \\ \dot{y}_r\end{bmatrix} = \begin{bmatrix}-sen\varphi & -\alpha cos\varphi \\ cos\varphi & -\alpha sen\varphi\end{bmatrix} * \begin{bmatrix}u \\ \omega\end{bmatrix} \qquad (9)$$

Se observa que se necesita obtener el ángulo de orientación del robot, además de la velocidad lineal y también la velocidad angular, las cuales se obtuvieron más adelante en el modelamiento del mismo.

En la Figura 32 se contempla que se rotó el robot como también se desplazó en el primer cuadrante de la visualización de la cámara. En este punto se procedió a determinar el centroide junto con el ancho y alto de la imagen, para ello se ubicó el robot en el centro del cuadrante y a partir de ahí hacer uso de las siguientes fórmulas para hallar las dimensiones mencionadas:

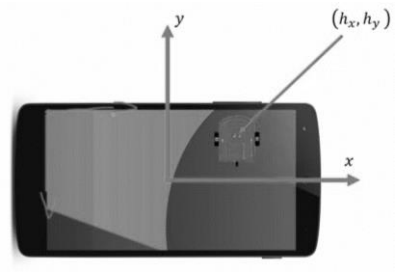


Figura 32. Determinación de centroide del robot.

Se debe hallar hx, para ello considere hacer la diferencia entre el centroide en el eje X y la mitad del ancho de la imagen:

$$h_x = C_{x0} - \frac{width}{2} \qquad (10)$$

Y, por último, también se necesita hallar hy, para ello se realiza la diferencia entre la mitad del alto de la imagen y el centroide en el eje Y:

$$h_y = \frac{heigth}{2} - C_{y0} \qquad (11)$$

Dichas fórmulas también se requieren agregar a la programación del prototipo más adelante.

2.26. Modelado Matemático del robot.

Para el modelamiento del robot, se plantean algunas variables y determinaciones como se puede observar en la Figura 33. Además, se plantea en un plano cartesiano la trayectoria que el robot realiza al momento de estar en funcionamiento.

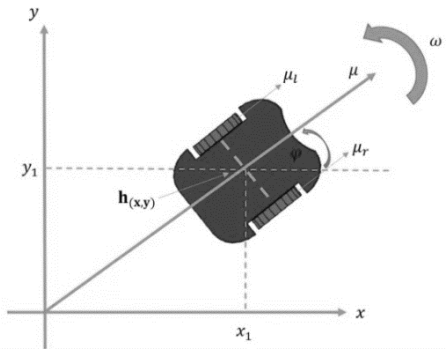


Figura 33. Trayectoria del robot en el plano cartesiano.

Donde:

u_l=Velocidad lineal del motor izquierdo

u_r=Velocidad lineal del motor derecho

u=Velocidad lineal global del robot

$\dot{\varphi}$=Ángulo de orientación

ω=Velocidad angular

Entonces se empieza declarando las variables hx y hy, las cuales van ser x1 y y1 respectivamente:

$$h_x = x_1 \qquad (12)$$

$$h_y = y_1 \qquad (13)$$

Donde, por consiguiente, sus derivadas vienen a resultar las siguientes expresiones:

$$\dot{h}_x = \mu * cos\varphi \qquad (14)$$

$$\dot{h}_y = \mu * sen\varphi \qquad (15)$$

Se deduce que la velocidad angular va a ser igual a la derivada del ángulo de orientación del robot:

$$\dot{\varphi} = \omega \qquad (16)$$

No obstante, también hay que definir la velocidad lineal global, la cual es el resultado de las sumas de las velocidades lineales de cada llanta dividido entre dos:

$$\mu = \frac{u_r + u_l}{2} \qquad (17)$$

La velocidad angular, en cambio, tiene como resultado la diferencia entre la velocidad de llanta derecho y la llanta izquierda, divide para la distancia que existe entre el punto de origen y el punto de referencia:

$$\omega = \frac{u_r - u_l}{d} \qquad (18)$$

Una vez determinadas las velocidades lineales y la angular, se pueden agrupar en una matriz 2x2, esta asociación de fórmulas se las inserta en la programación de desarrollo. Esta matriz sirve para hallar la velocidad lineal y angular del robot al momento de estar operativo:

$$\begin{bmatrix} \mu \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{d} & -\frac{1}{d} \end{bmatrix} * \begin{bmatrix} u_r \\ u_l \end{bmatrix} \qquad 19)$$

En la derivada de hx, se tenía de incógnito la velocidad lineal global, pero como ya se determinó dicha velocidad, pues, se la reemplaza y se obtiene la siguiente expresión:

$$\dot{h}_x = \frac{u_r + u_l}{2} * cos\varphi \qquad (20)$$

De la misma manera se reemplaza la velocidad lineal global, en la derivada de hy:

$$\dot{h}_y = \frac{u_r + u_l}{2} * sen\varphi \qquad (21)$$

También se determinó la velocidad angular, entonces se procede a reemplazarla en la derivada del ángulo de orientación del robot:

$$\dot{\varphi} = \frac{u_r - u_l}{d} \qquad (22)$$

Con las tres últimas fórmulas determinadas, se las ubica en una matriz 3x3 y dicha agrupación también se debe añadirla a la programación del prototipo; esto con la finalidad de determinar el ángulo de orientación del robot referenciando las velocidades lineales de cada llanta, como también la posición del mismo en el plano X y Y.

$$\begin{bmatrix} \dot{h}_x \\ \dot{h}_y \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \frac{cos\varphi}{2} & \frac{cos\varphi}{2} \\ \frac{sen\varphi}{2} & \frac{sen\varphi}{2} \\ \frac{1}{d} & -\frac{1}{d} \end{bmatrix} * \begin{bmatrix} u_r \\ u_l \end{bmatrix} \qquad (23)$$

3. MARCO METODOLOGICO.

3.1. Funcionalidad.

Este proyecto ofrece el desarrollo de un robot móvil con gestión multitarea manejado con el uso de visión artificial y programado en Python. En la Figura 34 se presenta el diagrama de funcionamiento del prototipo.

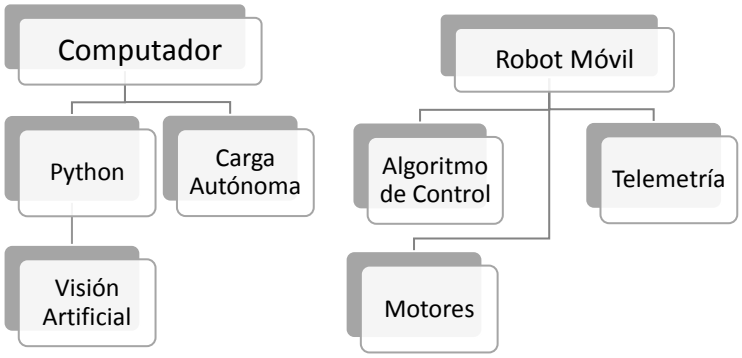


Figura 34. Diagrama de funcionamiento.

Para empezar, se carga la programación del IDE de Arduino en la tarjeta microcontroladora NANO para el control de los motores, esto se lo logra gracias a las librerías motorControl (Control en lazo abierto y cerrado de motores) y PinChangeInterrupt (Habilita todos los puertos del microcontrolador como interrupciones), luego se abre el IDE de Atom, se inicializa la programación principal en la cual se muestra una ventana de la librería Tkinter con 5 botones para que al momento de que el usuario/operador de clic en en cada uno de ellos, aparezca la interfaz respectiva de cada práctica para el desarrollo del prototipo. Una vez teniendo la pantalla principal, se procede a encender el robot y colocarlo en el piso para visualizar la operación y funcionamiento en cada práctica.

La práctica #1 consiste en la configuración y manejo del robot inalámbricamente, aquí se puede manipular el prototipo mediante el uso del mouse o teclas del computador, en diferentes direcciones (adelante, atrás, izquierda, derecha, paro), todo esto gracias a la comunicación serial haciendo uso del módulo HC-05. Con la configuración del control PID se logra una correcta alineación de los motores para que al momento en que el robot esté activo, este último no se desvíe o tome direcciones bruscas.

La práctica #2 consiste en telemetría robótica, el operador realiza la comunicación bidireccional, así como adquisición de datos entre Arduino y Python, todo esto puede ser visto en la interfaz correspondiente a esta práctica. El robot tiene implementado un potenciómetro el cual sirve para una batería simulada en un rango de 0 a 5v, esto a su vez para ser adquisición de datos ya que Arduino es quien envía este último dato hacia Python para ser mostrado en la pantalla. También se visualiza la orientación del prototipo en radianes/segundos para saber si está girando hacia la izquierda o hacia la derecha, y a su vez se visualiza un texto que me indique hacia que dirección está yendo el robot. Por último, también como adquisición de datos, el usuario puede activar y desactivar mediante dos teclas desde el computador un diodo led y un buzzer, estos últimos se muestran en la pantalla de esta práctica e irán cambiando dependiendo si están o no activados.

La práctica #3 se trata de la configuración del sistema de visión artificial, aquí se centra en la determinación de colores, la ubicación del robot en el plano y también del punto de carga. Con la ayuda de la cámara USB y el uso de la librería OpenCV, se procesan distintos colores y coordenadas, en este caso se tiene en el plano varios colores, entre ellos las tareas que el robot debe seguir, el centroide de este último y el punto de carga autónoma implementado, los cuales la cámara detecta cada uno de ellos, esto lo hace de forma individual dividiéndolos por su máscara y el color, deduciendo pues que cada color tiene un distinto factor HSV (Hue-Saturation-Value).

La práctica #4 realiza la gestión multitarea autónoma, el robot aparte de determinar los colores, debe hacer una trayectoria entre varios puntos asignados. Aquí se obtiene 3 colores, los cuales son amarillo, verde y naranja; pueden ser de cualquier color. Cada color tiene un distinto factor HSV y por ende se debe parametrizar, una vez parametrizado cada uno de los colores, pues son distintas tareas que el robot tiene que realizar. Con esta práctica el prototipo realiza una trayectoria cíclica entre los distintos colores asignados y a su vez de manera individual. Al final, también existe la opción de manipular un punto en el plano mediante el computador, el cual el robot debe de seguir acorde vaya movilizando dicho punto en el plano. Adicional se visualiza en la pantalla la velocidad lineal, velocidad angular y orientación que el prototipo realice.

La práctica #5 consiste en el desarrollo de carga de batería del robot y movilización autónoma a punto de carga entre tareas. Para finalizar con el desarrollo del prototipo, en esta práctica el robot debe movilizarse de forma autónoma hacia el punto de carga indiferentemente si está realizando alguna tarea asignada. Se tiene la opción de cambiar de

batería simulada, es decir la que se manipula con la ayuda del potenciómetro, o utilizar la batería real, es decir la que brindan las baterías con las que el robot funciona. El punto de carga está asignado con un botón el cual debe de estar activo para que cuando la batería este menor a 2 voltios, el robot deje de realizar la tarea que este haciendo y se dirija hacia el punto de carga, y una vez que ya este cargado, pues, vuelve a realizar la tarea que estaba realizando.

3.2. Materiales necesarios.

Para llevar a cabo el proyecto se necesitaron de los siguientes materiales. A continuación, se adjuntan los valores de cada uno de los componentes en la Tabla 1.

Tabla 1
Listado de materiales

Descripción	Cantidad	Valor unitario ($)	Valor final ($)
Impresión de tarjeta PCB			
Placa PCB	5	4	20
Materiales electrónicos de robot			
Arduino Nano	1	7.50	7.50
Módulo Bluetooth HC-05	1	5.50	5.50
Puente H L293D	1	1.50	1.50
Step-Down LM2596	1	2.50	2.50
Step-Up MT3608	1	2.40	2.40
Cargador de batería TP4056	1	1.60	1.60
Receptor carga inalámbrica “Qi”	1	3.50	3.50
Motor CHR-GM25-370	2	10.60	21.20
Buzzer	1	1	1
Diodo Led	1	0.40	0.40

Materiales computador visión artificial			
Computadora portátil	1	800	800
Cámara USB	1	80	80

Estructura, visión artificial y carga de batería			
Base del robot	2	5	10
Soporte del motor	2	3	6
Acople del motor	2	3	6
Llanta	2	1	2
Rueda loca	1	1.35	1.35
Porta pilas	1	1	1
Batería recargable	2	1.80	3.60
Módulo receptor de bobinas de carga	1	12	12
Estructura de la cámara	1	30	30

Materiales varios			
Borneras	9	0.15	1.35
Porta IC 16 pines	1	1	1
Conector pin macho	12	0.05	0.60
Tira pin hembra 15	2	1	2
Tira pin hembra 6	1	1	1
Cable de conexión jumper	12	0.08	0.96
		Total:	1025.96

Para este proyecto, se requiere una computadora portátil con las siguientes especificaciones: Sistema operativo de 64 bits en adelante, 4GB de RAM, procesador de doble núcleo a 2 GHz o más y Core™ i5. En este caso se utilizó una computadora con características superiores.

Además, se necesitan los softwares adecuados para el diseño e implementación del robot para la correcta comunicación, análisis y procesamiento de datos que nos proporcionan los dispositivos.

3.2.1. Python

Para descargar el software se busca en la página oficial de Python, “https://python.org/”, tal como se aprecia en Figura 35.

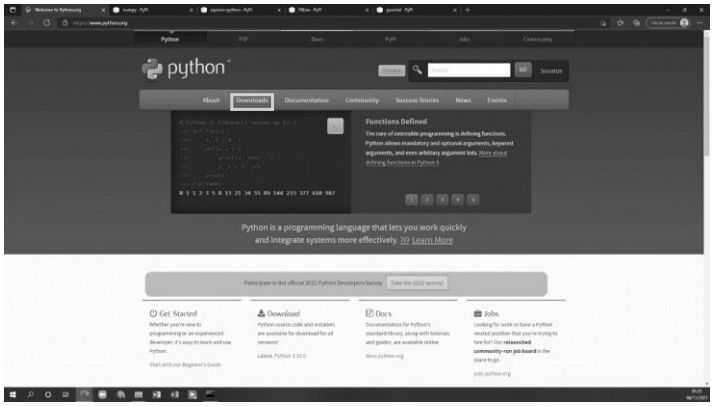


Figura 35. Página de Python.

En el inicio de la página de Python, hay que dirigirse a la pestaña de descargas y se busca la versión 3.7.7, tal como se observa en la Figura 36.

Figura 36. Página de descargas de Python.

Se da clic en la versión 3.7.7, como se había mencionado, y se selecciona el ejecutable de 64 bits, así como se visualiza en la Figura 37.

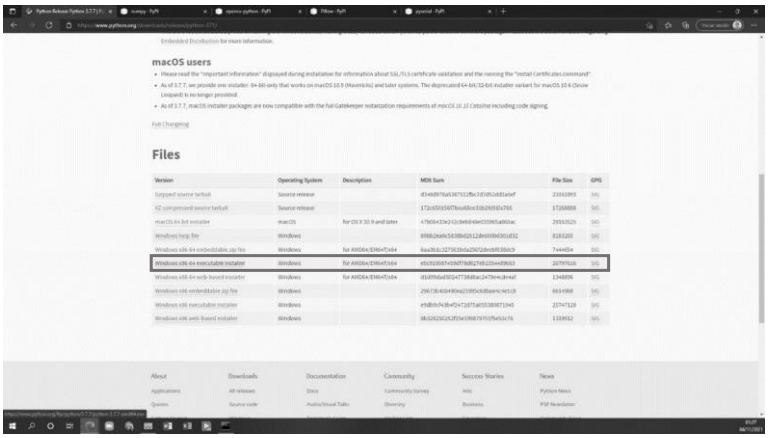


Figura 37. Apartado de ejecutables de la versión 3.7.7.

El proceso de instalación es inmediato, hay que dirigirse a la carpeta de descargas, se procede a dar clic derecho sobre el archivo descargado y se lo ejecuta como administrador. En la Figura 38 se puede apreciar al momento de ejecutar el archivo.

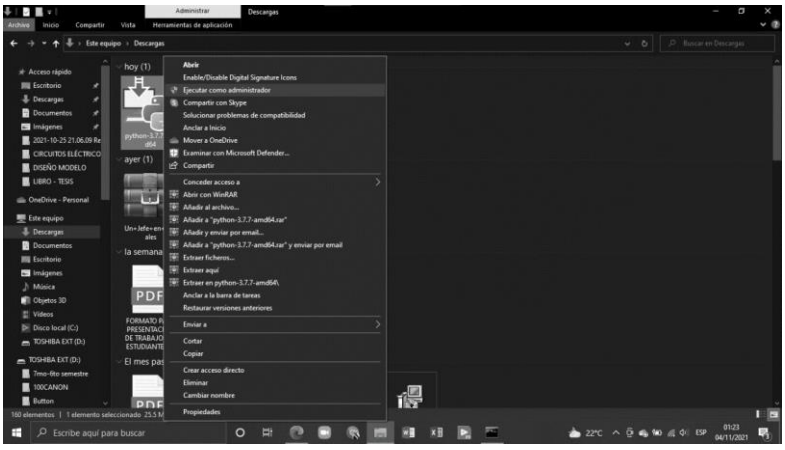


Figura 38. Ejecutando el archivo en la carpeta de descargas.

Una vez ejecutado el archivo, se muestra una ventana de instalación de Python, se marca con un visto en el apartado de abajo donde dice “Add Python 3.7 to PATH” y se da clic en “Install Now”. En la Figura 39 se visualiza la ventana de instalación mencionada.

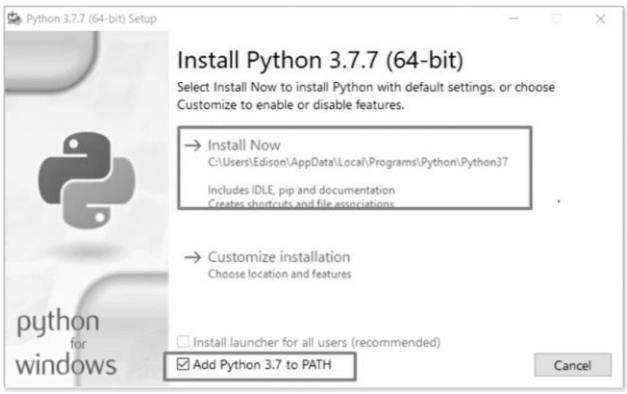


Figura 39. Ventana de instalación de Python.

Luego de haberse instalado, se muestra la ventana donde dice “Setup was successful”, indicando que la instalación fue todo un éxito y se procede a dar clic en “Close”., tal como se observa en la Figura 40.

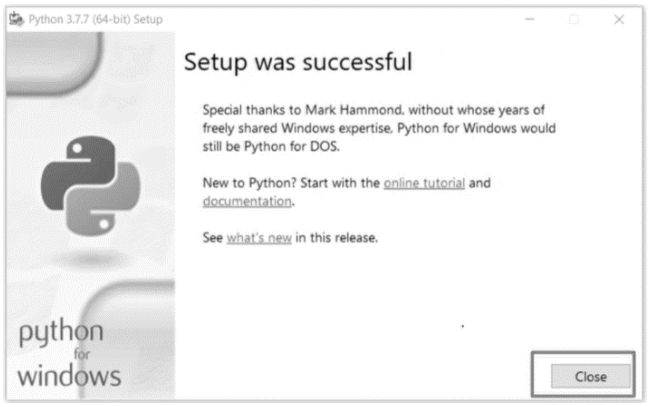


Figura 40. Ventana de finalización de la instalación de Python.

Se verifica que Python se haya instalado correctamente ingresando al cmd de Windows y colocando “python”. En la Figura 41 se aprecia lo mencionado.

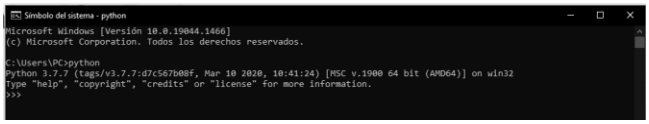


Figura 41. Ventana cmd de Windows.

3.2.1.1. Librería NumPy.

Numpy es el paquete fundamental para el manejo de matrices o arrays en Python. Para poder instalar esta librería se debe de ingresar a “https://pypi.org/project/numpy/”, tal como se observa en la Figura 42.

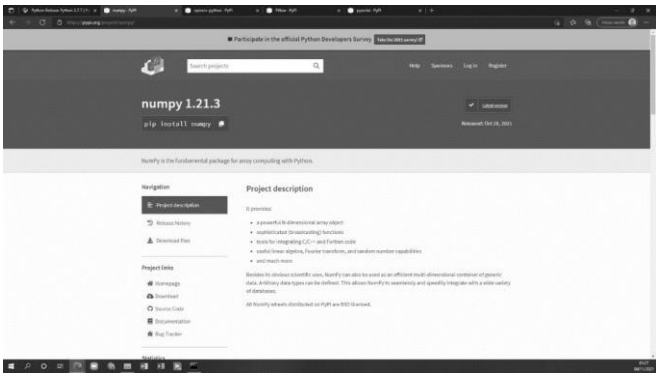


Figura 42. Ventana de la librería NumPy para Python.

Una vez ingresado al enlace, se copia el comando de instalación de la librería. En la Figura 43 se puede ver aquello.

Figura 43. Comando de instalación de la librería NumPy.

Se abre el cmd de Windows, y se pega el comando previamente copiado “pip install numpy”, se da clic en Enter y empieza a descargarse todo el paquete de la librería NumPy, tal como se puede comprobar en la Figura 44.

Figura 44. Instalación de librería NumPy en el cmd de Windows.

3.2.1.2. Librería OpenCV.

Para tener una visión óptima de los puntos de referencia se usa la librería OpenCV. Para implementarlo en Python se lo debe buscar ingresando a “https://pypi.org/project/opencv-python/”, así como se muestra en la Figura 45.

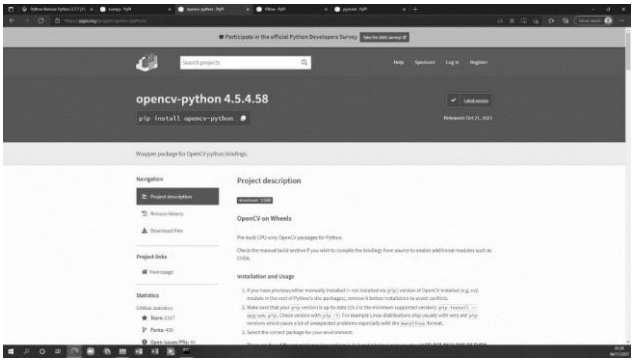


Figura 45. Ventana de la librería OpenCV para Python.

Una vez ingresado al enlace, se copia el comando de instalación de la librería. En la Figura 46 se puede ver aquello.

Figura 46. Comando de instalación de la librería OpenCV.

Se abre el cmd de Windows, y se pega el comando previamente copiado “pip install opencv-python”, se da clic en Enter y empieza a descargarse todo el paquete de la librería OpenCV, tal como se puede comprobar en la Figura 47.

Comentado [VAPI1]:

Figura 47. Instalación de librería OpenCV en el cmd de Windows.

3.2.1.3. Librería Pillow.

Uno de los módulos también a utilizar es Pillow, el cual es una librería de imágenes propia de Python. En este caso permite manejar imágenes que posteriormente se va a visualizar en una interfaz GUI. No se puede utilizar directamente la librería de OpenCV para trabajar con imágenes entonces para ello se debe de instalar esta librería. Para implementarlo en Python se ingresa a “https://pypi.org/project/Pillow/”, tal como se muestra en la Figura 48.

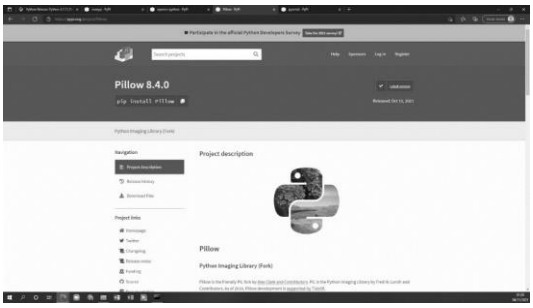


Figura 48. Ventana de búsqueda de la librería Pillow para Python.

Una vez ingresado al enlace, se copia el comando de instalación de la librería. En la Figura 49 se puede ver aquello.

Figura 49. Comando de instalación de la librería Pillow.

Se abre el cmd de Windows, y se pega el comando previamente copiado “pip install numpy”, se da clic en Enter y empieza a descargarse todo el paquete de la librería NumPy, tal como se puede comprobar en la Figura 50.

Figura 50. Instalación de librería Pillow en el cmd de Windows.

3.2.1.4. Librería Pyserial.

Otro módulo muy importante es el Pyserial, esta librería permite acceder al puerto serial. Para implementarlo en Python se debe de ingresar a “https://pypi.org/project/pyserial/”, tal como se muestra en la Figura 51.

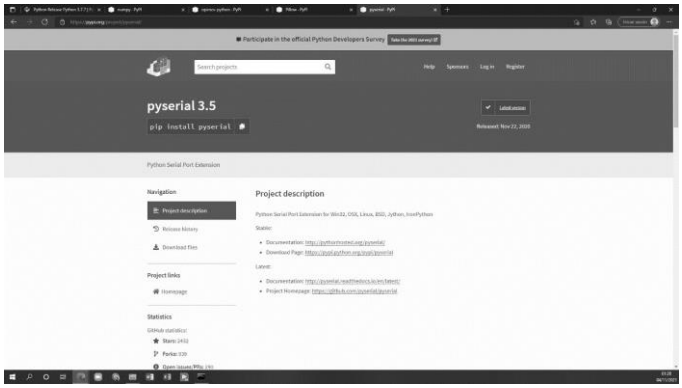


Figura 51. Ventana de búsqueda de la librería Pillow para Python.

Una vez ingresado al enlace, se copia el comando de instalación de la librería. En la Figura 52 se puede ver aquello.

Figura 52. Comando de instalación de la librería Pyserial.

Se abre el cmd de Windows, y se pega el comando previamente copiado “pip install pyserial”, se da clic en Enter y empieza a descargarse todo el paquete de la librería Pyserial, tal como se puede comprobar en la Figura 53.

Figura 53. Instalación de librería Pyserial en el cmd de Windows.

3.2.1.5. LibreríaMatplotlib.

Para la visualización de imágenes del PID, se necesita también la librería de Matplotlib, la cual permite crear gráficos en dos dimensiones. Para implementarlo en Python se debe de ingresar a “https://pypi.org/project/matplotlib/”, tal como se muestra en la Figura 54.

Figura 54. Ventana de búsqueda de la librería Matplotlib para Python.

Una vez ingresado al enlace, se copia el comando de instalación de la librería. En la Figura 55 se puede ver aquello.

Figura 55. Comando de instalación de la librería Matplotlib.

Se abre el cmd de Windows, y se pega el comando previamente copiado “pip install matplotlib”, se da clic en Enter y empieza a descargarse todo el paquete de la librería Matplotlib, tal como se puede comprobar en la Figura 56.

Figura 56. Instalación de librería Matplotlib en el cmd de Windows.

3.2.1.6. Librería Os-system.

Por último, se instala la librería de Os-System, la cual ayuda a ejecutar los programas a partir de una sola ventana, esto se lo realiza con el fin de no hacer la ejecución de cada práctica de manera individual. Para implementarlo en Python se debe de ingresar a “https://pypi.org/project/os-sys/”, tal como se muestra en la Figura 57.

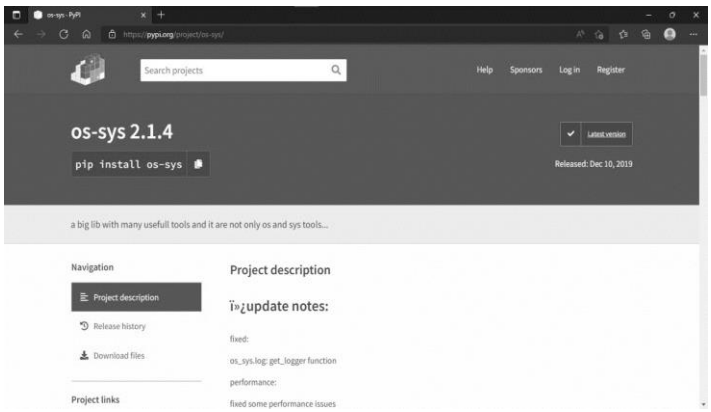


Figura 57. Ventana de búsqueda de la librería Os-System para Python.

Una vez ingresado al enlace, se copia el comando de instalación de la librería. En la Figura 58 se aprecia aquello.

Figura 58. Comando de instalación de la librería Os-System.

Se abre el cmd de Windows, y se pega el comando previamente copiado “pip install os-sys”, se da clic en Enter y empieza a descargarse todo el paquete de la librería Os-System, tal como se puede comprobar en la Figura 59.

Figura 59. Instalación de librería Os-System en el cmd de Windows.

3.2.2. Arduino.

Para su descarga se debe de ingresar a la página oficial de Arduino, “https://www.arduino.cc”, tal como se aprecia en Figura 60.

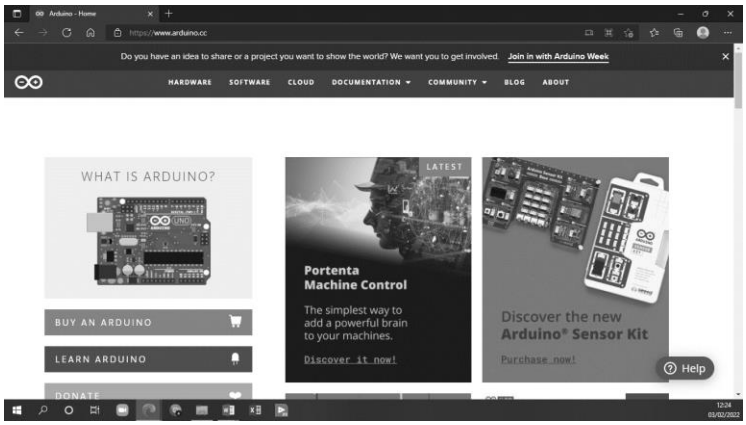


Figura 60. Página oficial de Arduino.

Una vez en la página, hay que dirigirse a la pestaña de descargas y se escoge la opción para el sistema operativo que se necesite, tal como se muestra en la Figura 61.

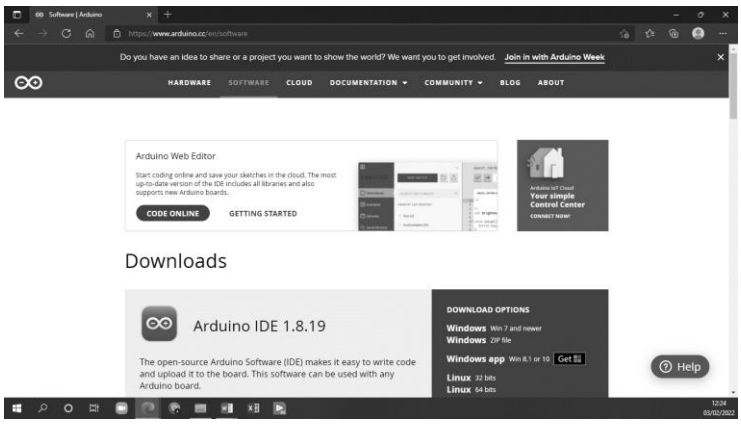


Figura 61. Página de descarga de Arduino.

Una vez descargado se procede a ejecutar el archivo para empezar a instalar. Se acepta los términos y condiciones del software y se da clic en siguiente, eso en todas las ventanas que nos salgan y finalmente se escoge el lugar de instalación. En la Figura 62 y en la Figura 63 se observa el proceso de instalación.

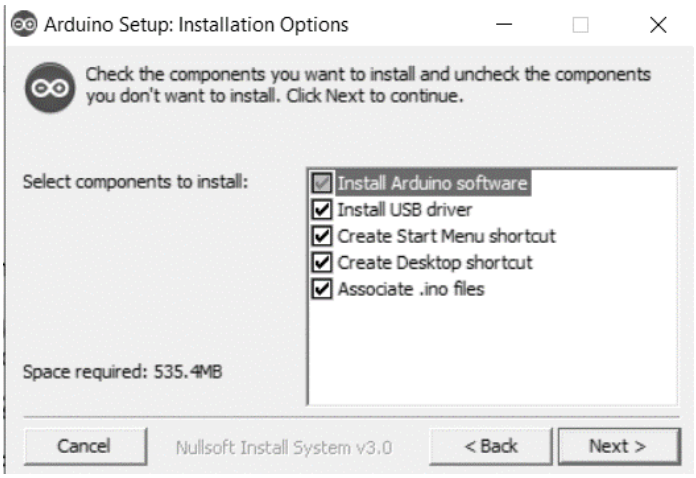


Figura 62. Ventana de instalación de Arduino.

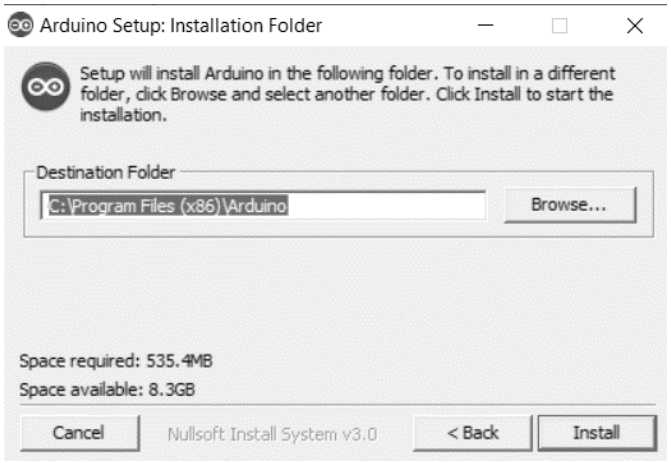


Figura 63. Ventana final de instalación de Arduino.

Una vez concluido se puede trabajar en el software, en la Figura 64 se aprecia la ventana del IDE de Arduino.

Figura 64. IDE de Arduino.

3.2.2.1. Librería PinChangeInterrupt.

El Arduino tradicional dispone de interrupciones especiales llamadas PINCHANGE INTERRUPTIONS, las cuales se pueden utilizar en todos los pines incluso en las entradas analógicas, pero son un poco complejas de administrar, sin embargo, gracias la librería PinChangeInterrupt.h se pueden utilizar estas interrupciones de la misma manera que una interrupción normal.

Para su instalación en el IDE Arduino hay que dirigirse a la sección de “Programa”, seguido de esto a la opción “Incluir Librería” y se selecciona “Administrar Bibliotecas”. En la Figura 65 se puede observar el proceso de instalación.

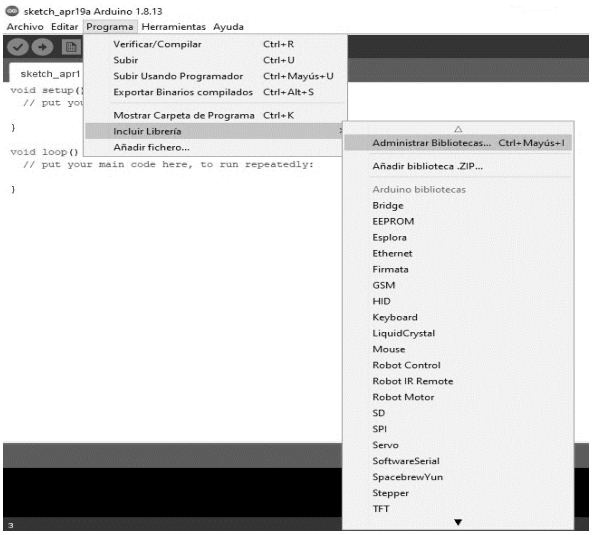


Figura 65. Proceso de instalación de la librería PinChangeInterrupts.

Luego de dar clic en “Administrar Bibliotecas”, aparece una ventana de gestor de librerías. En la Figura 66 se puede apreciar la ventana mencionada.

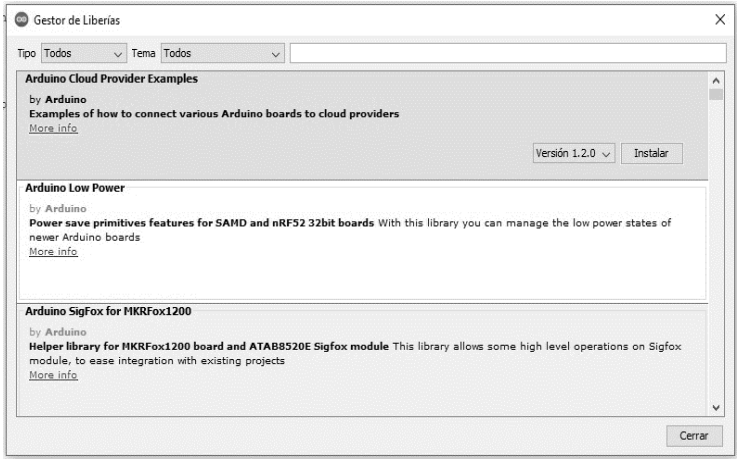


Figura 66. Gestor de librerías.

Finalmente, se coloca el nombre de la librería que se desee, en este caso “PinChangeInterrupts”, se selecciona la versión adecuada y se procede a añadir en la programación. En la Figura 67 se visualiza la librería que seleccionamos.

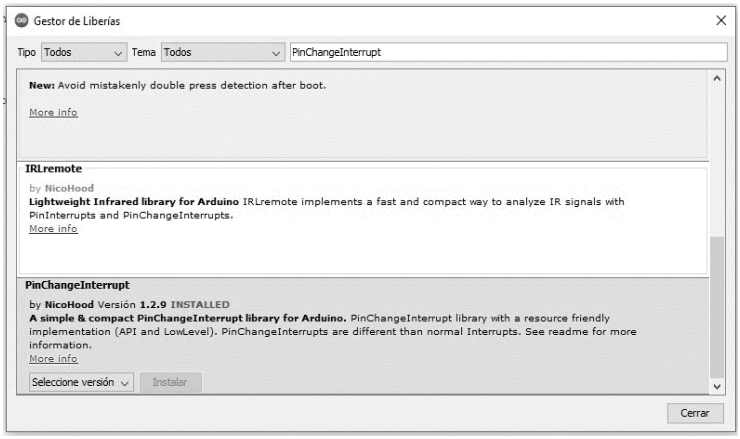


Figura 67. Librería PinChangeInterrupts.

3.2.2.2. Librería MotorControl.

La Librería MotorControl permite realizar un buen control en motores DC, ya sea, en lazo abierto o en lazo cerrado, indiferentemente de cuál sea la aplicación que se requiera.

Se empieza dirigiéndose hacia la sección de “Programa”, se selecciona “Incluir Librería” y se procede a dar clic a “Añadir biblioteca”. En la Figura 68 se encuentra el paso a paso de este proceso.

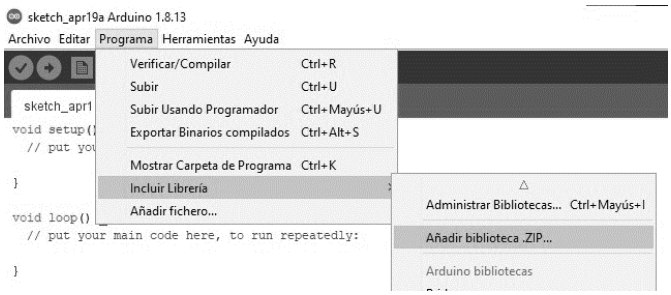


Figura 68. Proceso para incluir librería motorControl.

Luego de dar clic en “Añadir Biblioteca”, aparece una ventana en la cual se selecciona el fichero que se desee para añadir en el programa. En la Figura 69 y en la Figura 70 se aprecia el proceso de instalación culminado una vez importada la librería requerida.

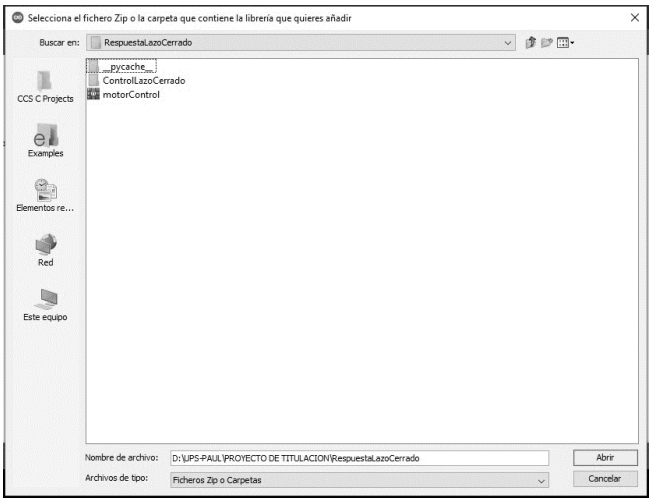


Figura 69. Ventana para seleccionar el fichero deseado.

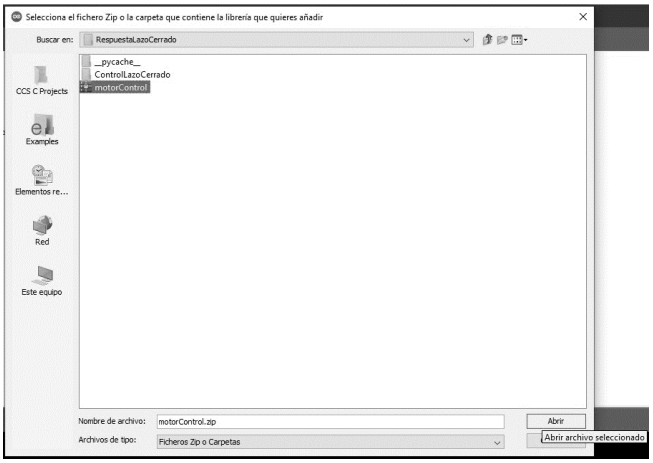


Figura 70. Selección del fichero.

3.2.3. Atom.

Para descargar el software se busca en la página oficial de Atom, “https://atom.io”, al momento de entrar detecta el sistema operativo del ordenador y sugiere el instalador adecuado, tal como se aprecia en la Figura 71.

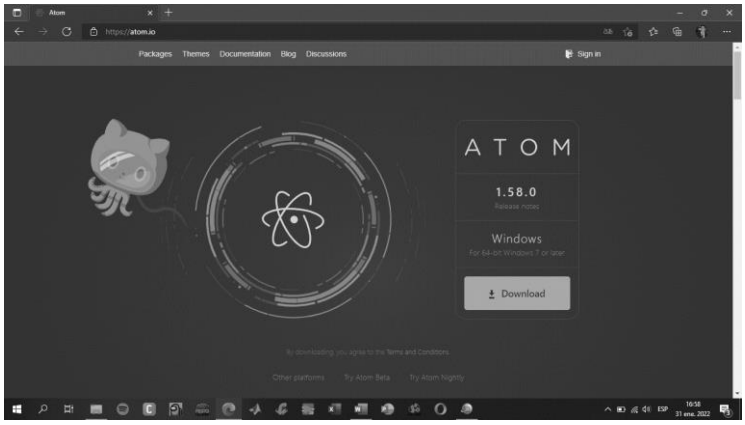


Figura 71. Página oficial de Atom.

Se procede a dar clic en “Download” y se comienza a descargar el instalador. En el apartado de descargas del buscador se da clic y aparece una ventana emergente para empezar a ejecutar el software, se da clic en “Ejecutar” y se espera a que termine de instalarse. En la Figura 72 y en la Figura 73 se muestra lo recientemente escrito.

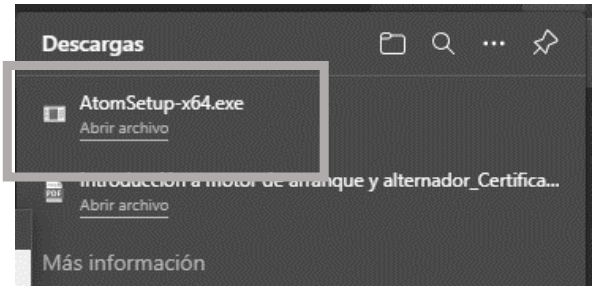


Figura 72. Instalador descargado en el apartado de descargas.

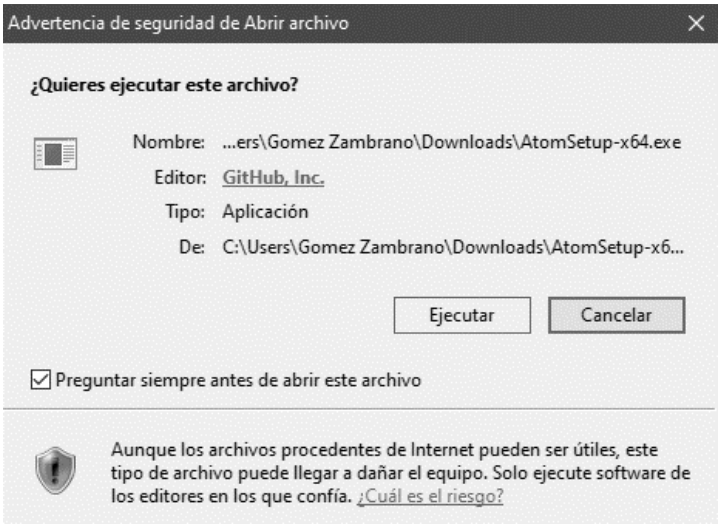


Figura 73. Ventana emergente para ejecución del instalador del software.

Como se puede observar, en la Figura 74 ya se está empezando a abrir el software Atom.

Figura 74. Ventana emergente del software instalado.

En la Figura 75 ya está instalado el software Atom, listo para empezar a programar.

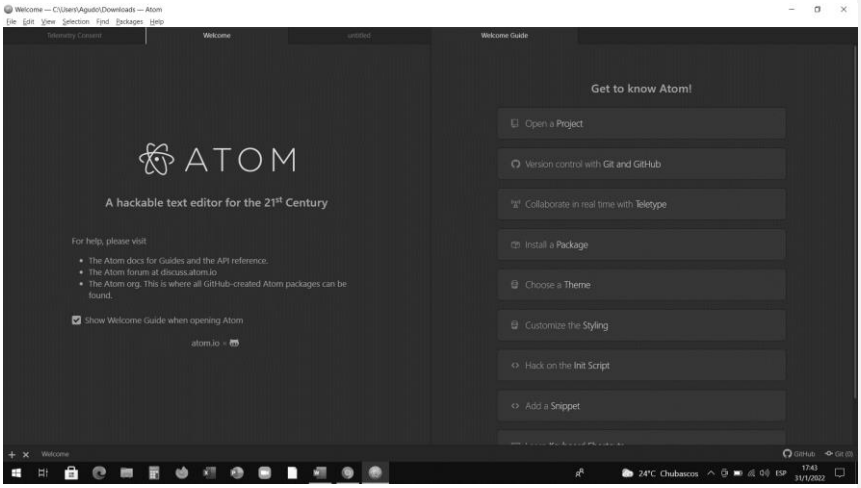


Figura 75. Pantalla Principal del software Atom.

3.2.4. EasyEDA.

Para el diseño del circuito electrónico del robot se necesitó la plataforma EasyEDA, para ello se ingresa a la página oficial “https://easyeda.com/”, tal como se aprecia en la Figura 76.

Figura 76. Pantalla Principal de EasyEDA.

En la parte superior derecha se da clic en “Register”, luego se abre una pestaña en el navegador y procede a registrarse. En la Figura 77 y en la Figura 78 se observa aquello.

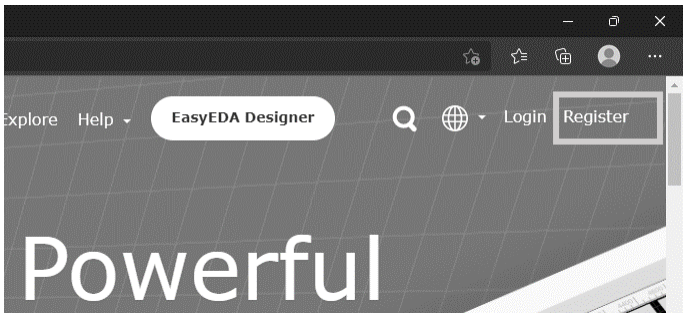


Figura 77. Opción de registro.

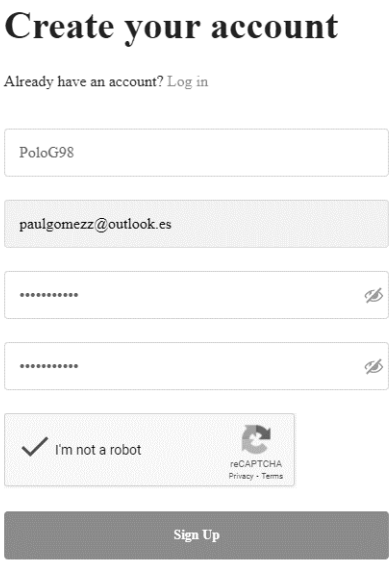


Figura 78. Pantalla Principal de EasyEDA.

Una vez registrado, la página redirige hacia la cuenta oficial y para empezar a diseñar se da clic en EasyEDA Designer. En la Figura 79 se aprecia aquello.

Figura 79. Página de inicio de EasyEDA Designer.

Luego de dar clic en “EasyEDA Designer”, aparece esta pestaña en la cual se puede empezar a diseñar el circuito esquemático, así como lo muestra la Figura 80 a continuación.

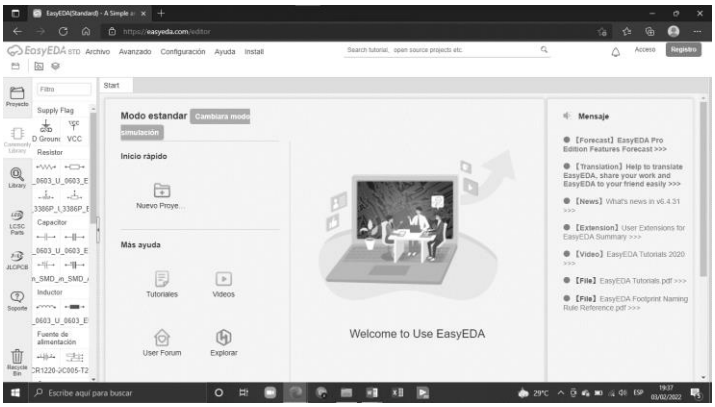


Figura 80. Pestaña de diseño en EasyEDA.

3.2.5. Autodesk Inventor

Hay que dirigirse hacia la página de Autodesk inventor en Academic Software y se procede a dar clic en “Get started” arriba a la derecha para solicitar tu cuenta Education de Autodesk. En la Figura 81 se observa la página oficial de Autodesk.

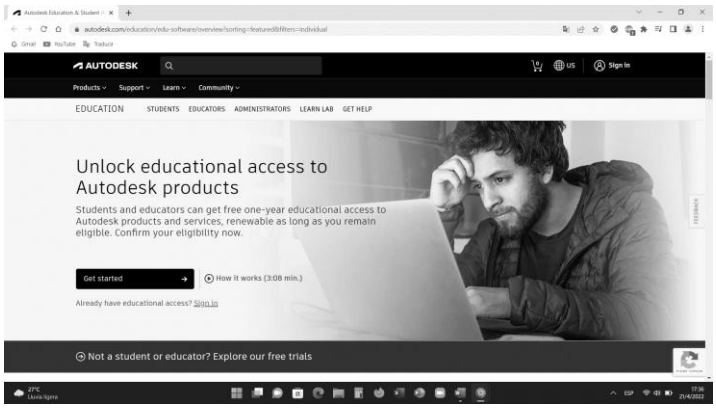


Figura 81. Página de Autodesk Inventor.

Se da clic en “Create Account” y se procede a llenar los datos solicitados. En la Figura 82 se visualiza la ventana para ingresar los datos para adquirir una cuenta en Autodesk Inventor.

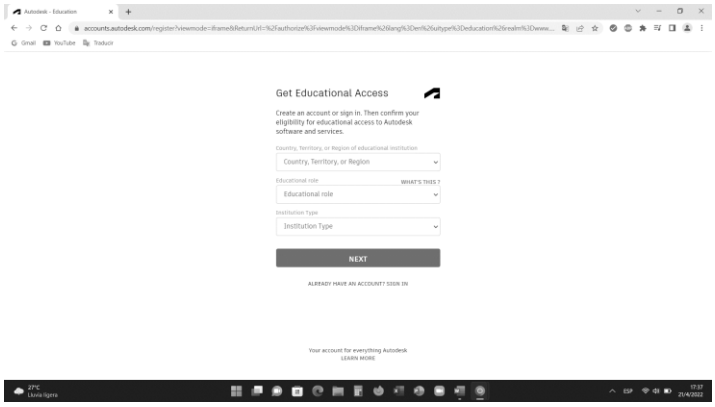


Figura 82. Ventana para la creación de cuenta.

En la siguiente en pestaña se procede a colocar los nombres y apellidos junto con un email. A continuación, en la Figura 83 se visualiza la ventana para colocar los datos del usuario.

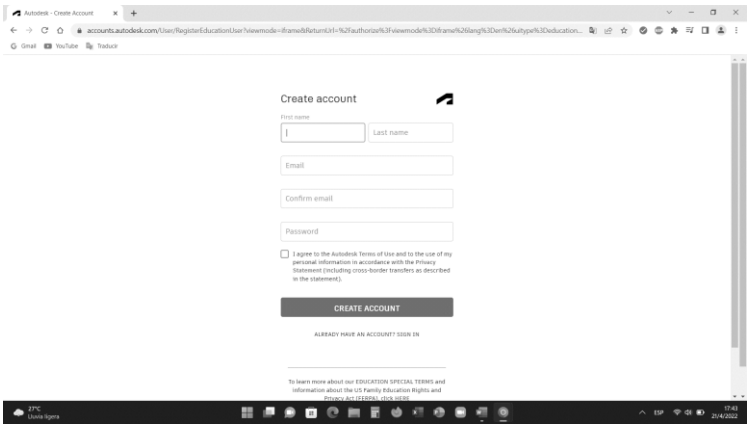


Figura 83. Ventana para ingresar datos personales.

A continuación, se recepta un correo electrónico para confirmar la cuenta. En la Figura 84 se puede ver como se realiza la verificación del correo electrónico.

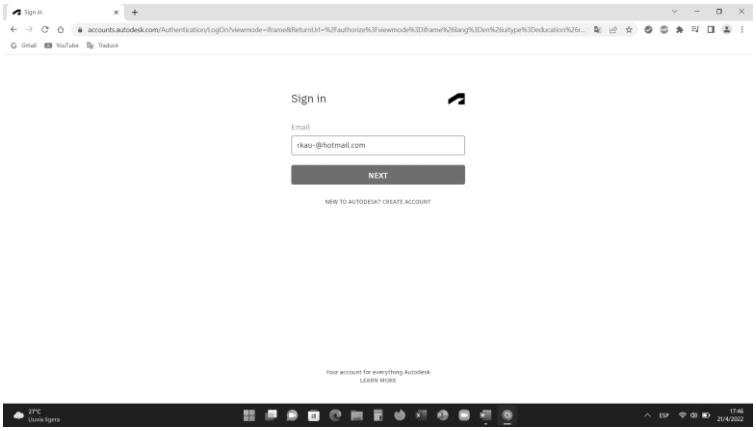


Figura 84. Verificación de correo electrónico.

Una vez obtenida la cuenta, hay que dirigirse a la página de Autodesk Inventor en Academic Software de nuevo. En la Figura 85 ya se puede observar que se ingresa con éxito a la página de Autodek Inventor.

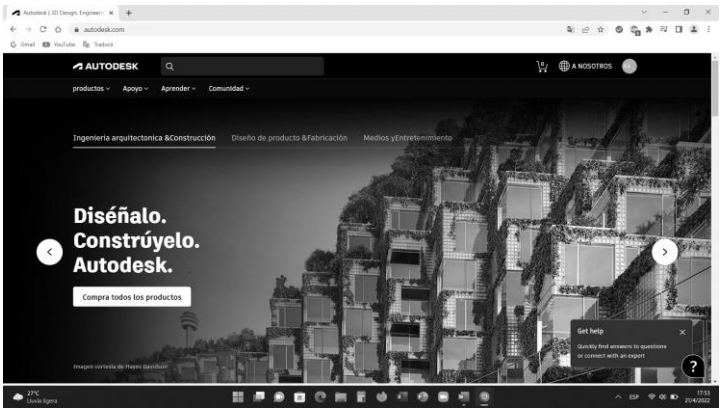


Figura 85. Inicio de sesión en Autodesk Inventor.

Se hace clic en el botón descargar Autodesk Inventor Profesional. En la Figura 86 se visualiza el producto escogido.

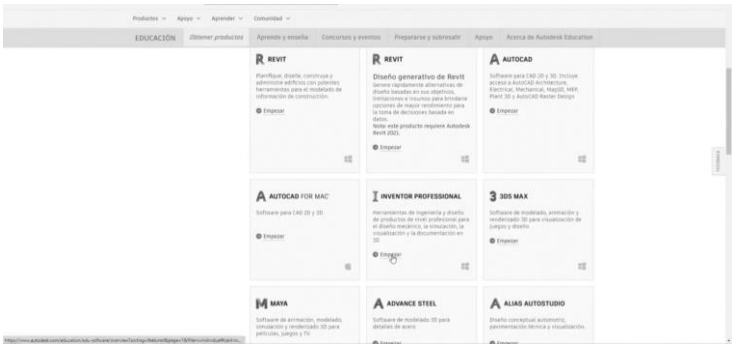


Figura 86. Selección de Producto.

Se da clic en insttall y después en Accept. En la Figura 87 se observa la ventana del año e idioma que tiene el software.

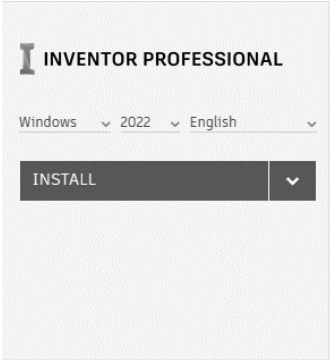


Figura 87. Ventana de instalación.

Se procede a escoger el espacio donde se instala el software. En la Figura 88, se aprecia la parte de selección de espacio para Autodesk.

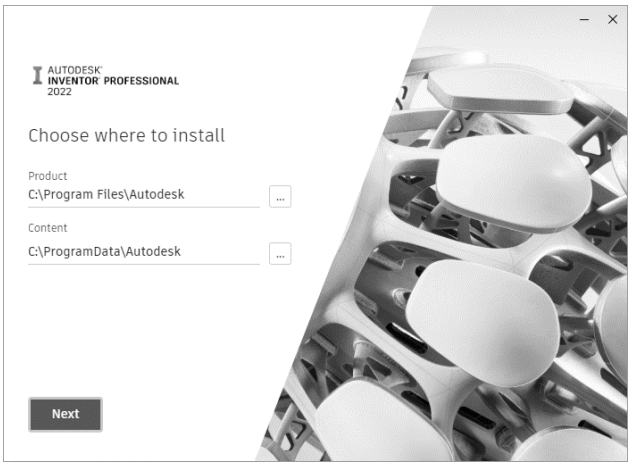


Figura 88. Selección de ubicación de instalación del software.

Se selecciona los componentes que se quiere incluir en el programa. En la Figura 89 se muestra cómo se escogen componentes que mejoran la experiencia en Autodesk.

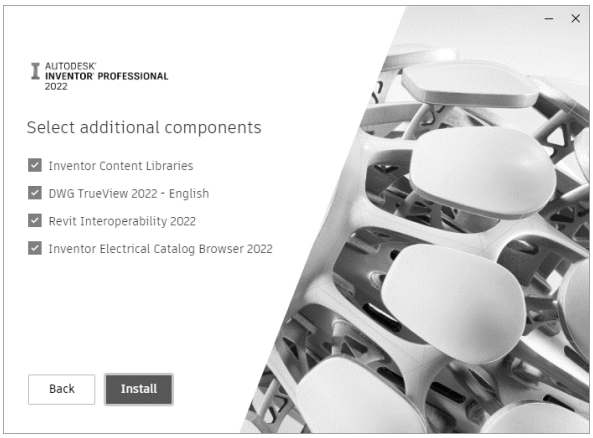


Figura 89. Selección de Componentes adicionales.

La duración de esta etapa depende de la velocidad de cada ordenador, como se puede apreciar en la Figura 90.

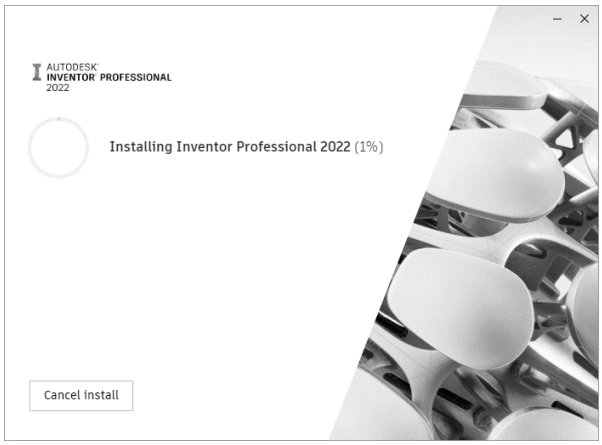


Figura 90. Inicio de instalación.

Una vez finalizada la instalación, se procede a iniciar Inventor Profesional 2022, como se observa en la Figura 91.

Figura 91. Instalación finalizada.

Pantalla de inicio de autodesk Inventor. Por último, en la Figura 92, se puede visualizar como ya se tiene disponibles todas las funciones de Autodesk Inventor.

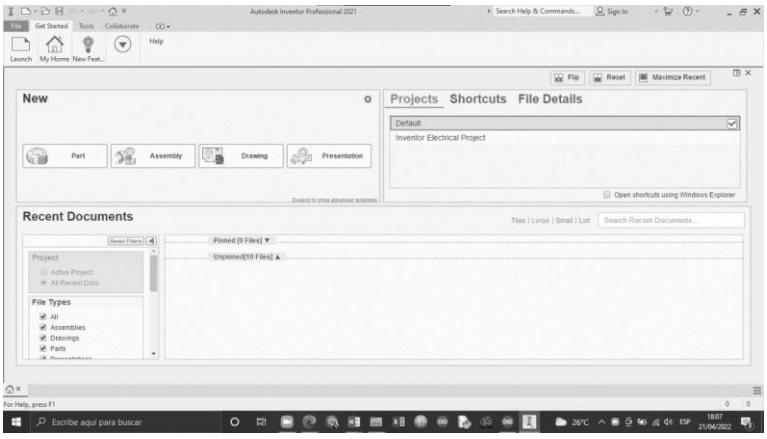


Figura 92. Autodesk Inventor habilitado.

3.3. Diseño del Circuito Impreso.

Se diseñó el circuito mediante el software EasyEDA, siendo este un programa de uso libre y en línea. Beneficia en lo que es la elaboración de fabricación de PCB a bajo precio para ingenieros electrónicos y aficionados. Esta tarjeta consta distintos componentes como las borneras para las entradas de voltaje y también para la conexión de los motores, además, contamos con un led indicador de carga y un buzzer para la verificación de telemetría.

Contiene un Arduino nano el cual es la tarjeta de adquisición de datos entre Arduino y Python, esto lo logra mediante el módulo Bluetooth HC-05. Para el control de motores se utilizó un puente H, el cual, indica el sentido de giro de cada uno.

Para la alimentación del circuito, se empleó dos baterías de litio que suman 8 voltios, este voltaje llega a un módulo de carga, seguido se suministra al módulo Step-up para que el voltaje se eleve a 12 voltios y este voltaje alimente a los motores. Y para finalizar, el voltaje suministrado llega al módulo step-Down para disminuir el voltaje y que alimente de manera correcta el Arduino junto con los demás componentes. En la Figuras 93 y en la Figura 94 se encuentran los diseños tanto del circuito esquemático y el circuito PCB.

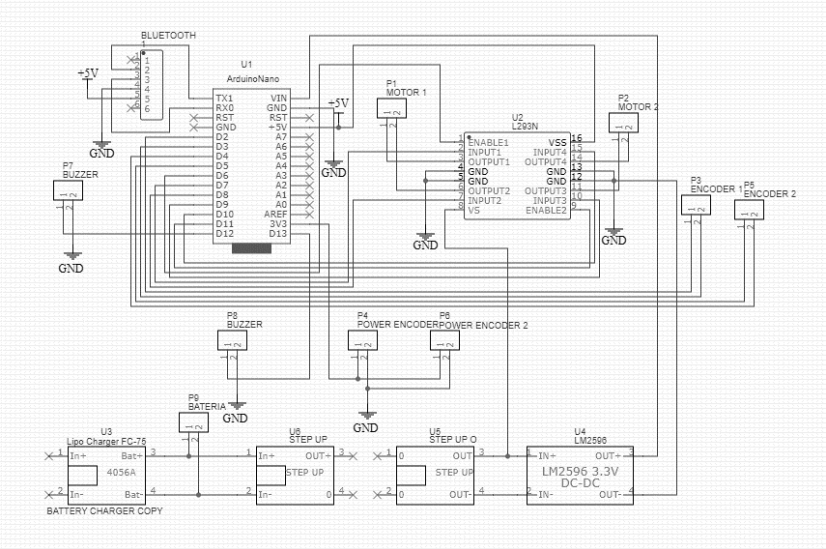


Figura 93. Circuito esquemático.

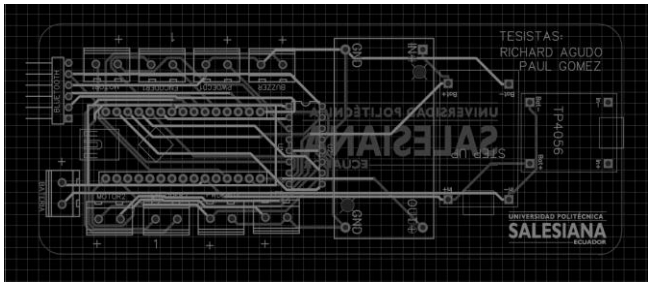


Figura 94. Circuito PCB.

Se puede ver a continuación el resultado de la fabricación de la tarjeta PCB. En la Figura 95 se observa la tarjeta con sus componentes soldados, mientras que, en las Figura 96 y en la Figura 97 se muestra la tarjeta previa a la colocación de cada uno de materiales.

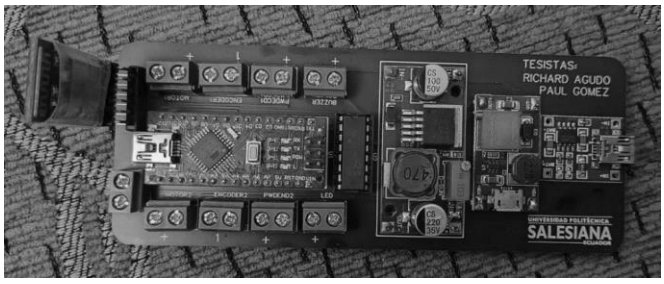


Figura 95. Tarjeta electrónica impresa con sus componentes.

Figura 96. Tarjeta previa a soldar sus componentes (parte inferior).

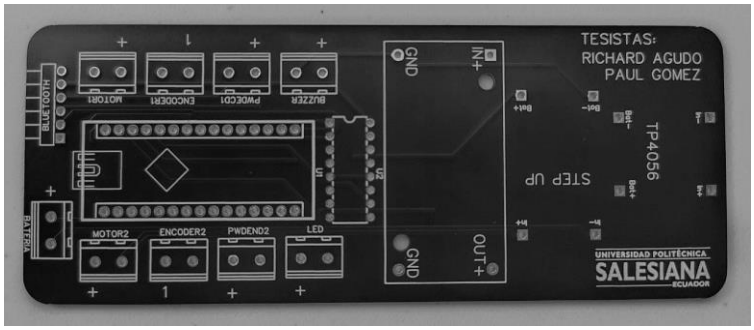


Figura 97. Tarjeta previa a soldar sus componentes (parte superior).

3.4. Diseño de controlador PID para el control de motores.

Para el diseño del sistema de control se utilizó el software Python y el IDE de Arduino. Se empezó armando el circuito con un solo motor, tal como se muestra en la Figura 98, para que, de esta manera se pueda realizar el control PID y una vez configurado uno de los motores, se utilizó el mismo método para hallar el PID del otro motor y, al momento de realizar la conexión, las entradas de alimentación y los canales del encoder se invierten para que las trayectorias del robot sean sincronizadas.

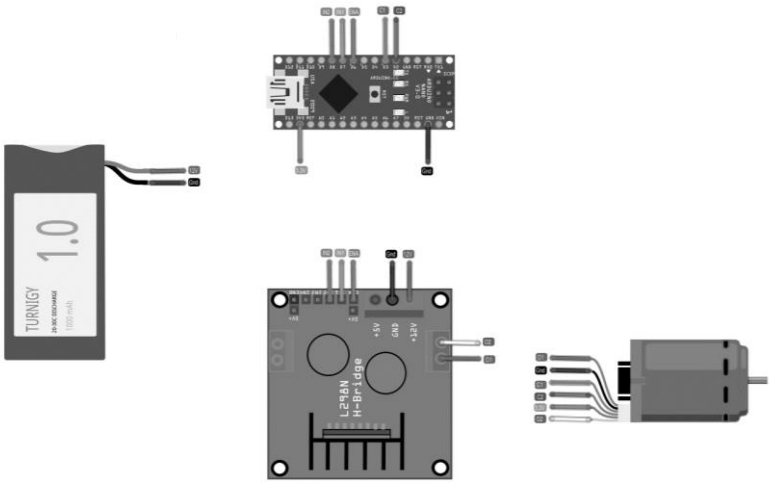


Figura 98. Esquema de circuito para encontrar el control PID.

3.4.1. Control en lazo abierto.

Una vez armado el circuito, como se muestra en la Figura 99, se debe de conocer cómo se comporta el motor en lazo abierto. Para ello, se utilizaron bases de varios proyectos encontrados de investigaciones previas para complementar información sobre el comportamiento de los motores.

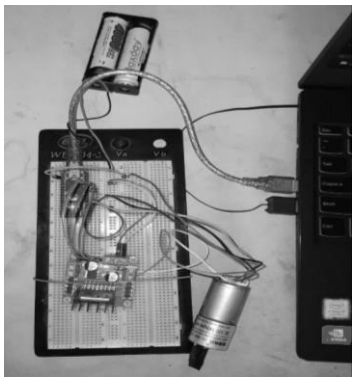


Figura 99. Elaboración de circuito para encontrar el control PID.

Se comienza cargando el código en lazo abierto al microcontrolador Arduino nano, no obstante verificar el puerto serial para que no exista error de compilación. En la Figura 100, Figura 101, Figura 102, Figura 103 y Figura 104 se muestra toda la programación en Arduino para el control en lazo abierto.

```
#include "motorControl.h"

//////////////////////////// COMUNICACION SERIAL ////////////////////
String inputString = "";
bool stringComplete = false;
const char separator = ',';
const int dataLength = 1;
double data[dataLength]; // Valor regula ciclo de trabajo (PWM)

////////////////////////// Control Lazo Abierto ///////////////
unsigned long lastTime = 0, sampleTime = 100;  // Tiempo de muestreo
motorControl motor(sampleTime);

//////////////////////////// ENCONDER ////////////////////
const byte    C1 = 3; // Entrada de la señal A del encoder (Cable amarillo).
const byte    C2 = 2; // Entrada de la señal B del encoder (Cable verde).

//////////////////////////// PUENTE H ///////////////////////
const byte    in1 = 7;
const byte    in2 = 8;
const byte    enA = 6;

volatile int count = 0;
volatile byte ant  = 0;
volatile byte act  = 0;
```

Figura 100. Segmento 1 de programación para visualizar comportamiento del motor.

```
//////////////////// Variables Motor ////////////////

double w = 0.0;  // Velocidad angular en rad/s.
int outValue = 0; //Variable de control (pwm)
double constValue = 3.1733; //(1000*2*pi)/R ---> R = 1980 Resolucion encoder cuadruple

void setup()
{
  ////////////////// CONFIGURACION PUERTO SERIAL ////////////////
  Serial.begin(9600);

  ////////////////// Limites de señales //////////////////
  motor.setCvLimits(255,20);
  motor.setPvLimits(11,0);

  ////////////////// CONFIGURACION DE PINES ////////////////
  pinMode(C1, INPUT);
  pinMode(C2, INPUT);

  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);

  ////////////////// MOTOR APAGADO //////////////////////
  digitalWrite(in1, false);
  digitalWrite(in2, false);

  analogWrite(enA,outValue);
```

Figura 101. Segmento 2 de programación para visualizar comportamiento del motor.

```
  if (millis() - lastTime >= sampleTime)
  {
      if (outValue > 0) anticlockwise(in2,in1,enA,outValue); else clockwise(in2,in1,enA,abs(outValue));
      w =(constValue*count)/(millis()-lastTime); // Calculamos velocidad rad/s
      lastTime = millis(); // Almacenamos el tiempo actual.
      count = 0;  // Reiniciamos los pulsos.
      w = motor.scalePv(w); // Escalar del 0 al 100%
      Serial.println(w);
   }
}

void serialEvent() {
  while (Serial.available()) {
    char inChar = (char)Serial.read();
    inputString += inChar;
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

Figura 102. Segmento 3 de programación para visualizar comportamiento del motor.

```
// Encoder precisión cuádruple.
void encoder(void)
{

    ant=act;
    act=PIND & 12;

    if(ant==0  && act== 4)  count++;
    if(ant==4  && act==12)  count++;
    if(ant==8  && act== 0)  count++;
    if(ant==12 && act== 8)  count++;

    if(ant==0 && act==8)  count--;
    if(ant==4 && act==0)  count--;
    if(ant==8 && act==12) count--;
    if(ant==12 && act==4) count--;

}
```

Figura 103. Segmento 4 de programación para visualizar comportamiento del motor.

```
void clockwise(int pin1, int pin2,int analogPin, int pwm)
{
  digitalWrite(pin1, LOW);
  digitalWrite(pin2, HIGH);
  analogWrite(analogPin,pwm);
}

void anticlockwise(int pin1, int pin2,int analogPin, int pwm)
{
  digitalWrite(pin1, HIGH);
  digitalWrite(pin2, LOW);
  analogWrite(analogPin,pwm);
}
```

Figura 104. Segmento 5 de programación para visualizar comportamiento del motor.

Ya compilada la programación a la tarjeta de Arduino, se procede a ejecutar el código del control en lazo abierto en Python, para que, de esta manera se pueda visualizar el comportamiento del motor mediante una gráfica. En la Figura 105, Figura 106 y Figura 107, se visualiza la programación realizada en el IDE de Atom, mientras que, en la Figura 108 se observan las curvas de la variable de control y la variable de proceso.

```
from pyArduino import *
import matplotlib.pyplot as plt

ts = 0.1 # Tiempo de muestreo
tf = 10  # Tiempo de simulacion
t = np.arange(0,tf+ts,ts) # Array de tiempo
N = len(t) # Numero de muestras

####################### Comunicacion Serial ##############

port = 'COM3'  # Com Arduino
baudRate = 9600 # Baudios

arduino = serialArduino(port,baudRate)# Objeto serial

arduino.readSerialStart() # Inicia lectura de datos

####################### Señales ###################

y  = np.zeros(N) # Variable de proceso (Pv)
u  = np.zeros(N) # Variable de control (Cv)
```

Figura 105. Segmento 1 de programación en Python para el control de lazo abierto.

```
######################### Loop ######################

for k in range(N):

      start_time = time.time() # Tiempo actual

      # Escalon
      if k*ts   > 3:  # Escalon a los 3 segundos
           u[k] = 40 # Valor escalon del 0 al 100% (40%)
      else:
           u[k] = 0;

      arduino.sendData([u[k]]) # Enviar Cv (debe ser una lista)

      y[k] = arduino.rawData[0] # Recibir Pv

      elapsed_time = time.time() - start_time # Tiempo transcurrido

      time.sleep(ts-elapsed_time) # Esperar hasta completar el tiempo de muestreo


arduino.sendData([0]) # Detener motor
arduino.close() # Cerrar puerto serial
```

Figura 106. Segmento 2 de programación en Python para el control de lazo abierto.

```
###################### Guardar señales ########################
with open('firstResponse.npy', 'wb') as f:
      np.save(f,u)
      np.save(f,y)
      np.save(f,t)
      np.save(f,ts)

#################### Mostrar figuras ####################

plt.plot(t,y,label='Pv')
plt.plot(t,u,label='Cv')
plt.legend(loc='upper left')
plt.show()
```

Figura 107. Segmento 3 de programación en Python para el control de lazo abierto.

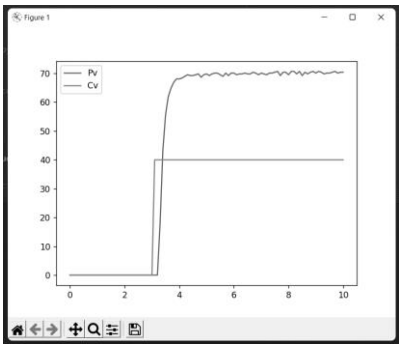


Figura 108. Curvas características de variable de proceso y variable de control.

Se contempló el comportamiento de la variable de proceso, se puede deducir que es un sistema de primer orden con retardo. El modelo de la planta correspondiente al sistema mencionado tiene como forma:

$$G(s) = \frac{K}{1 + sT} e^{-sL}$$

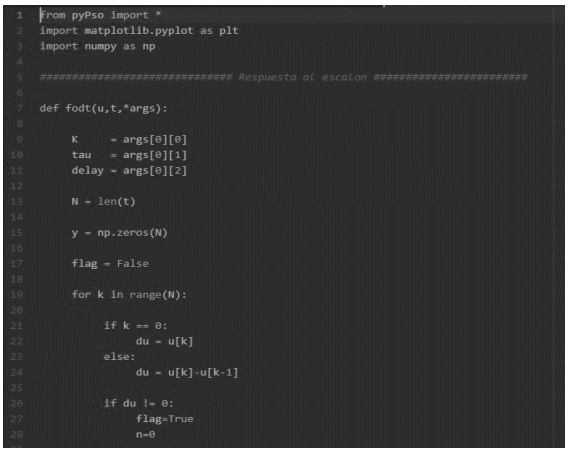
Donde:

K = ganancia del proceso

T = constante de tiempo en lazo abierto

L = retardo o delay

Con todo eso, se utilizó como base una plantilla de programación para encontrar los valores de un sistema de primer orden del modelo de la planta. Se ejecuta el código y automáticamente arrojan los valores deseados. En las Figuras 109, Figura 110, Figura 111 y Figura 112, se visualiza el código para un sistema de primer orden con retardo.



```
from pyPso import *
import matplotlib.pyplot as plt
import numpy as np

############################## Respuesta al escalon ######################

def fodt(u,t,*args):

      K     = args[0][0]
      tau   = args[0][1]
      delay = args[0][2]

      N = len(t)

      y = np.zeros(N)

      flag = False

      for k in range(N):

            if k == 0:
                  du = u[k]
            else:
                  du = u[k]-u[k-1]

            if du != 0:
                  flag=True
                  n=0
```

Figura 109. Segmento 1 de programación en Python para un sistema de primer orden con retardo.

```
            if flag:
                if n*ts < delay:
                    y[k] = y[k]
                else:
                    y[k] = K*u[k]*(1-np.exp(-(n*ts-delay)/tau))
                n = n+1

        return y

    ############################### Funcion de costo ##########################

    def costFunction(x):

        K     = x[:,0]
        tau   = x[:,1]
        delay = x[:,2]

        cost = np.zeros(swarmsize)

        for particle in range(swarmsize):
            ye = fodt(u,t,[K[particle],tau[particle],delay[particle]])
            error = y-ye
            cost[particle] = error.reshape(-1,1).T@error # sum(error^2)

        return cost
```

Figura 110. Segmento 2 de programación en Python para un sistema de primer orden con retardo.

```
########################## Cargar señales medidas ######################
with open('firstResponse.npy', 'rb') as f:
    u  = np.load(f)
    y  = np.load(f)
    t  = np.load(f)
    ts = np.load(f)


########################## Encontrar parametros ########################
swarmsize = 20
variables = 3
maxGen = 200

p = pso(costFunction,swarmsize,variables)

p.run(maxGen)

print(f"K = {np.round(p.globalp[0],4)}")
print(f"tau = {np.round(p.globalp[1],4)}")
print(f"delay = {np.round(p.globalp[2],4)}")

###################### Respuesta con valores optimos ###############
ye = fodt(u,t,p.globalp)
```

Figura 111. Segmento 3 de programación en Python para un sistema de primer orden con retardo.

```
##################### Mostrar figuras ####################
plt.figure()
plt.plot(p.summary,label='Global error')
plt.legend(loc='upper left')

plt.figure()
plt.plot(t,ye,label='Pv_estimated')
plt.plot(t,y,label='Pv_real')
plt.plot(t,u,label='Cv')
plt.legend(loc='upper left')

plt.show()
```

Figura 112. Segmento 4 de programación en Python para un sistema de primer orden con retardo.

Una vez ejecutado el código, se muestran los valores deseados para el modelo de la planta, tal como se muestra en la Figura 113. Por otro lado, en la Figura 114 se aprecia como el error disminuye de a poco hasta llegar a cero y a su vez como la variable de proceso real se ajusta con pocas perturbaciones a la variable de proceso estimada.

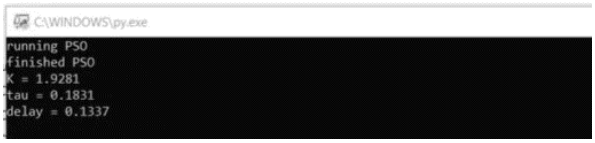


Figura 113. Valores del modelo de la planta de un sistema de primer orden.

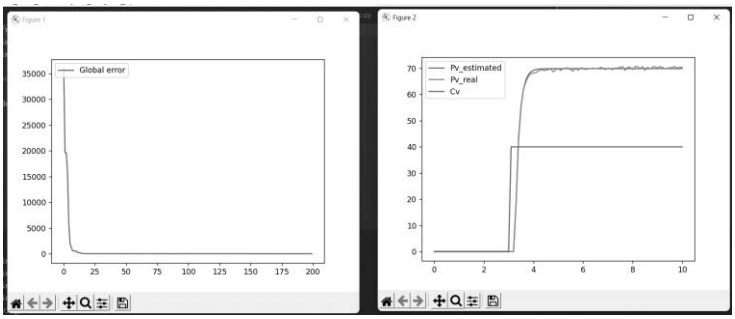


Figura 114. Gráficas de variable de proceso estimado, variable de proceso real y variable de control.

3.4.2. Control en lazo cerrado.

Ahora que ya se sabe que el modelo de la planta es un sistema de primer orden con retardo, se procede a cargar el código en lazo cerrado en el Arduino nano. Como se puede observar en las Figuras 115, Figura 116, Figura 117, Figura 118, Figura 119, Figura 120 y Figura 121, se utilizó una plantilla base de diferentes proyectos investigados de la misma manera que se realizó con el control en lazo abierto. En este código se introducen los valores que arrojó la ventana de comandos en Python de la sección anterior, los cuales son K, tau y delay, para que así cuando sea compilada la programación se muestre mediante el monitor serial los valores del PID, tal como se puede apreciar en la Figura 122.

```
#include "motorControl.h"

//////////////////////////// COMUNICACION SERIAL ////////////////////
String inputString = "";
bool stringComplete = false;
const char separator = ',';
const int dataLength = 1;
double data[dataLength]; // Velocidad de referencia (Sp)

//////////////////////////// CONTROLADOR PID ////////////////////
unsigned long lastTime = 0, sampleTime = 100;  // Tiempo de muestreo
motorControl motor(sampleTime);

//////////////////////////// ENCONDER ////////////////////
const byte    C1 = 3; // Entrada de la señal A del encoder (Cable amarillo).
const byte    C2 = 2; // Entrada de la señal B del encoder (Cable verde).

//////////////////////////// PUENTE H ////////////////////
const byte    in1 = 7;
const byte    in2 = 8;
const byte    enA = 6;

volatile int count = 0;
volatile byte ant  = 0;
volatile byte act  = 0;
```

Figura 115. Segmento 1 de la programación en Arduino para el control en lazo cerrado.

```
//////////////////// Variables Motor ////////////////

double w = 0.0;     // Velocidad angular en rad/s.
double wRef = 0.0;  // Velocidad angular de referencia en rad/s.
int outValue = 0;   //Variable de control (pwm)
double constValue = 3.1733; //(1000*2*pi)/R ---> R = 1980 Resolucion encoder cuadruple

void setup()
{
  ////////////////// CONFIGURACION PUERTO SERIAL ////////////////
  Serial.begin(9600);

  ////////////////// SINTONIA LAMBDA PID ////////////////

//   motor.lambdaTunning(1.9281,0.1831,0.1337);  //(K,tau,delay)
//
//  Serial.print(motor.getKc());
//  Serial.print(", ");
//  Serial.print(motor.getTi());
//  Serial.print(", ");
//  Serial.println(motor.getTd());
```

Figura 116. Segmento 2 de la programación en Arduino para el control en lazo cerrado.

```
  ////////////////// SU PROPIA SINTONIA //////////////////

  motor.setGains(0.21, 0.07, 0.05); //(Kc,Ti,Td)

  ////////////////// Limites de señales //////////////////
  motor.setCvLimits(255,20); // Limites de Cv (0-255), considerar zona muerta
  motor.setPvLimits(11,0);   // Limites de Pv (rad/s)

  ////////////////// CONFIGURACION DE PINES ////////////////
  pinMode(C1, INPUT);
  pinMode(C2, INPUT);

  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);

  ////////////////////// MOTOR APAGADO //////////////////////
  digitalWrite(in1, false);
  digitalWrite(in2, false);

  analogWrite(enA,outValue);

  ////////////////////////// INTERRUPCIONES //////////////////////
  attachInterrupt(digitalPinToInterrupt(C1), encoder, CHANGE);
  attachInterrupt(digitalPinToInterrupt(C2), encoder, CHANGE);

  lastTime = millis();
}
```

Figura 117. Segmento 3 de la programación en Arduino para el control en lazo cerrado.

```
void loop() {

  ////////// SI RECIBE DATOS //////////////
  if (stringComplete)
  {
    for (int i = 0; i < dataLength ; i++)
    {
      int index = inputString.indexOf(separator);
      data[i] = inputString.substring(0, index).toFloat();
      inputString = inputString.substring(index + 1);
     }

    wRef=data[0];

    inputString = "";
    stringComplete = false;
  }
```

Figura 118. Segmento 4 de la programación en Arduino para el control en lazo cerrado.

```
  if (millis() - lastTime >= sampleTime)
  {
      w =(constValue*count)/(millis()-lastTime); // Calculamos velocidad rad/s
      lastTime = millis(); // Almacenamos el tiempo actual.
      count = 0;  // Reiniciamos los pulsos.
      outValue = motor.compute(wRef,w);  // Control PID
      if (outValue > 0) anticlockwise(in2,in1,enA,outValue); else clockwise(in2,in1,enA,abs(outValue));
      Serial.println(w);
      Serial.println(outValue);
   }
}

void serialEvent() {
  while (Serial.available()) {
    char inChar = (char)Serial.read();
    inputString += inChar;
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

Figura 119. Segmento 5 de la programación en Arduino para el control en lazo cerrado.

```
    void serialEvent() {
      while (Serial.available()) {
        char inChar = (char)Serial.read();
        inputString += inChar;
        if (inChar == '\n') {
          stringComplete = true;
        }
      }
    }

    // Encoder precisión cuádruple.
    void encoder(void)
    {

        ant=act;
        act=PIND & 12;

        if(ant==0  && act== 4)  count++;
        if(ant==4  && act==12)  count++;
        if(ant==8  && act== 0)  count++;
        if(ant==12 && act== 8)  count++;

        if(ant==0 && act==8)  count--;
        if(ant==4 && act==0)  count--;
        if(ant==8 && act==12) count--;
        if(ant==12 && act==4) count--;
    }
```

Figura 120. Segmento 6 de la programación en Arduino para el control en lazo cerrado.

```
void clockwise(int pin1, int pin2,int analogPin, int pwm)
{
  digitalWrite(pin1, LOW);
  digitalWrite(pin2, HIGH);
  analogWrite(analogPin,pwm);
}

void anticlockwise(int pin1, int pin2,int analogPin, int pwm)
{
  digitalWrite(pin1, HIGH);
  digitalWrite(pin2, LOW);
  analogWrite(analogPin,pwm);
}
```

Figura 121. Segmento 7 de la programación en Arduino para el control en lazo cerrado.

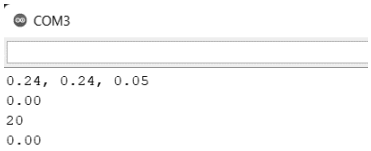


Figura 122. Valores de PID visualizados en el monitor serial.

Ya con los resultados de la programación de lazo cerrado en Arduino, hay que dirigirse al IDE de Atom para ejecutar el código de lazo cerrado, se selecciona un tiempo de simulación de entre 10 a 15 segundos según la preferencia del usuario. Si se coloca más tiempo, se observa un estudio más eficiente y amplio de la variable de proceso. En la Figura 123, Figura 124 y Figura 125, se visualiza la programación realizada en el IDE de Atom.

```
from pyArduino import *
import matplotlib.pyplot as plt
import numpy as np

ts = 0.1 # Tiempo de muestreo
tf = 15  # Tiempo de simulacion
t = np.arange(0,tf+ts,ts) # Array de tiempo
N = len(t) # Numero de muestras

######################## Comunicacion Serial ###############

port = 'COM3'  # Com Arduino
baudRate = 9600 # Baudios

arduino = serialArduino(port,baudRate,2)# Objeto serial

arduino.readSerialStart() # Inicia lectura de datos

######################### Señales ####################

y  = np.zeros(N) # Variable de proceso (Pv)
u  = np.zeros(N) # Variable de control (Cv)
sp = np.zeros(N) # Variable de deseada (Sp)
```

Figura 123. Segmento 1 de programación en Python para el control de lazo cerrado.

```
######################## Setpoint Escalon ###############

for k in range(N):
     if k*ts > 3:
          sp[k] = 10
     else:
          sp[k] = 0

#################### Setpoint Trayectoria ###############
sp = 4*np.cos(0.5*t)+6

######################### Loop ######################

for k in range(N):

     start_time = time.time() # Tiempo actual

     arduino.sendData([sp[k]]) # Enviar Sp (debe ser una lista)

     y[k] = arduino.rawData[0] # Recibir Pv
     u[k] = arduino.rawData[1] # Recibir Cv

     elapsed_time = time.time() - start_time # Tiempo transcurrido

     time.sleep(ts-elapsed_time) # Esperar hasta completar el tiempo de muestreo
```

Figura 124. Segmento 2 de programación en Python para el control de lazo cerrado.

```
arduino.sendData([0]) # Detener motor
arduino.close() # Cerrar puerto serial

#################### Mostrar figuras ####################
plt.figure()
plt.plot(t,sp,label='Sp')
plt.plot(t,y,label='Pv')
plt.legend(loc='upper left')

plt.figure()
plt.plot(t,u,label='Cv')
plt.legend(loc='upper left')

plt.show()
```

Figura 125. Segmento 3 de programación en Python para el control de lazo cerrado.

Para el estudio del motor en lazo cerrado, se debe de visualizar que la variable de proceso se pueda estabilizar de manera eficiente y segura sin que presente cualquier tipo de perturbaciones, para ello se requieren dos tipos de set point. El primero, es un set point de tipo escalón, el cual tiene una simulación de 12 segundos cuando sea mayor a 3 segundos este mismo. En la Figura 126 se aprecia que se está utilizando el set point de tipo escalón en primera instancia, mientras que en la Figura 127 se muestra como la variable de proceso logra estabilizarse al set point que en este caso es de 10.

```
25  ######################## Setpoint Escalon ################
26
27  for k in range(N):
28       if k*ts > 3:
29            sp[k] = 10
30       else:
31            sp[k] = 0
32
33  ###################### Setpoint Trayectoria ###############
34  #sp = 4*np.cos(0.5*t)+6
```

Figura 126. Segmento de programación en Python con set point escalón.

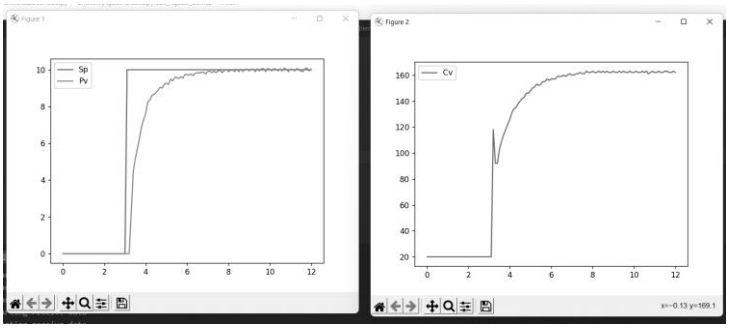


Figura 127. Curva característica con el set point escalón.

El segundo, es un set point de trayectoria, el cual tiene una simulación de 15. El set point de trayectoria se utiliza debido a que se está trabajando en el ámbito de la robótica y, pues, el robot mientras este haciendo las tareas tiene distintas trayectorias y velocidades a la par. En la Figura 128 se aprecia que se está utilizando el set point de tipo escalón en primera instancia, mientras que en la Figura 129 se muestra como la variable de proceso logra estabilizarse al set point que en este caso es de 10.

```
32
33  ###################### Setpoint Trayectoria ###############
34  sp = 4*np.cos(0.5*t)+6
```

Figura 128. Segmento de programación en Python con set point de trayectoria.

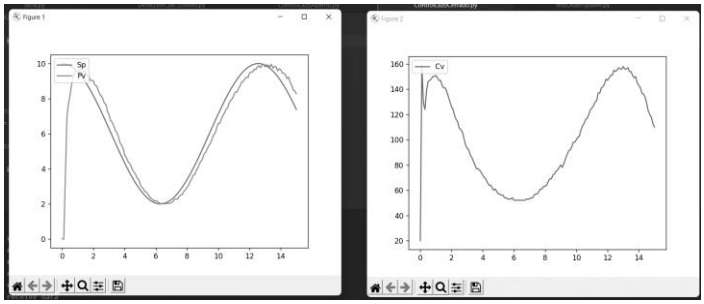


Figura 129. Curva característica con el set point trayectoria.

3.5. Estructura de la cámara.

La estructura de la cámara está compuesta por 2 trípodes de metal, los cuales permiten ubicar la cámara USB de manera centrada en una tubería de tipo PVC, dicho material es de ½". Todo esto es realizado, ya que, cumple el requisito de espacio necesaria para que traspase el cable USB que transmite y envía los frames de la imagen al computador. En la Figura 130 se observa la estructura armada de la cámara con los materiales mencionados, mientras que, en la Figura 131 se visualiza la cámara USB centrada de manera fija a la tubería PVC.

Figura 130. Estructura armada de la cámara USB.

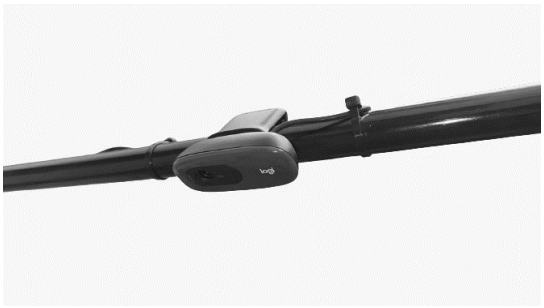


Figura 131. Cámara USB centrada en la tubería PVC.

3.6. Estructura del robot móvil.

En esta sección se puede observar las diferentes vistas del robot y poder contemplar alguno de sus componentes superficialmente. En la Figura 132 se puede apreciar la vista en 3D del prototipo, en la Figura 133 se tiene la vista lateral derecha, en la Figura 134 se tiene la vista superior del robot y en la Figura 135 se tiene la vista frontal del prototipo.

Figura 132. Vista 3D del robot móvil.

Figura 133. Vista lateral derecha del robot móvil.

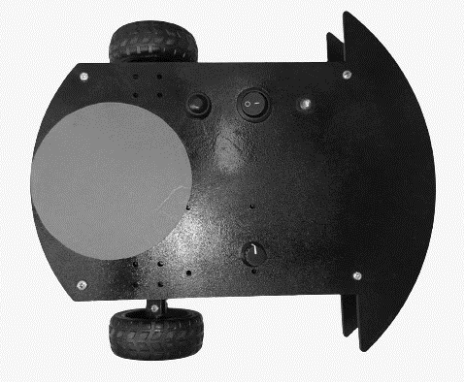


Figura 134. Vista superior del robot móvil.

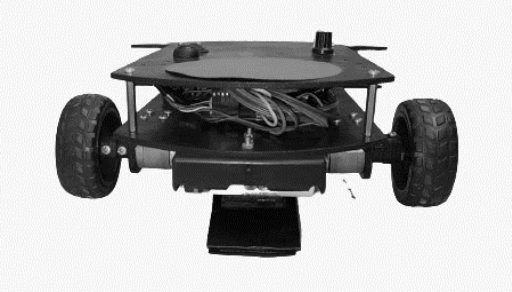


Figura 135. Vista frontal del robot móvil.

Las piezas de la estructura del prototipo se realizaron en el programa Inventor, como primera pieza se diseñó la estructura base del robot, es decir, el cuerpo del mismo. En la Figura 136 se observa el diseño en 3D de la base del proyecto, dicha base se fabricó en madera prensada con un espesor de 2 mm y elaborado en corte CNC a través de láser.

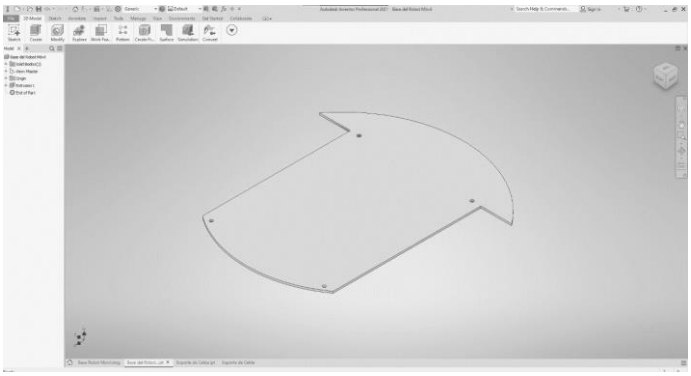


Figura 136. Diseño de la base del robot realizada en Inventor.

Para el encaje de los motores con sus respectivas llantas, se diseñó un soporte para el motor DC, como se muestra en la Figura 137, y un acople de la llanta, como se aprecia en la Figura 138. Estas piezas fueron realizadas con una impresión 3D.

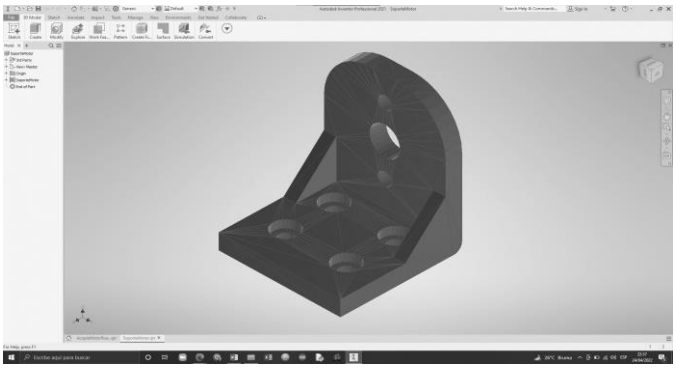


Figura 137. Diseño del soporte del motor realizado en Inventor.

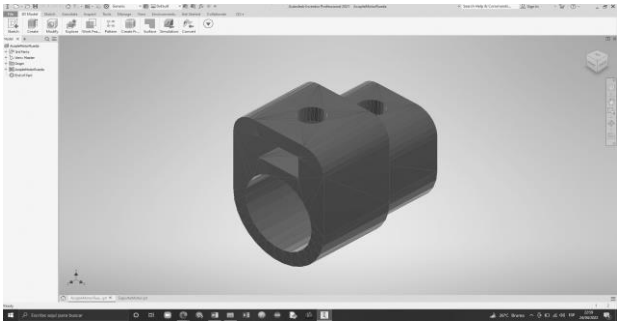


Figura 138. Diseño del acople del motor realizado en Inventor.

Por último, como contiene una celda de carga para la recepción de voltaje hacia el circuito, se diseñó un soporte para este componente en forma de "L" para que la celda se ajuste y no interfiera con los demás elementos. En la Figura 139 se visualiza el diseño de esta pieza, y esta última también fue realizada en impresión 3D.

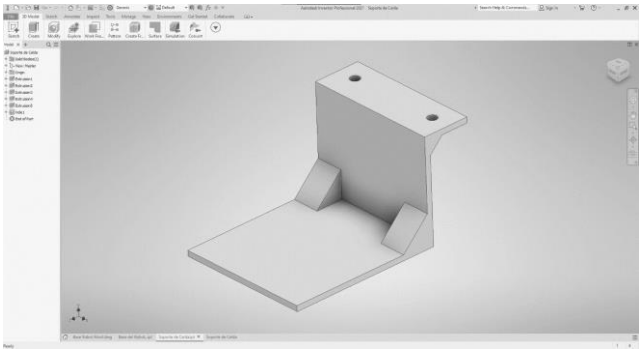


Figura 139. Diseño del soporte de la celda de carga realizada en Inventor.

3.7. Desarrollo de la programación en Arduino.

Se empieza la programación del control de motores llamando a las librerías PinChangeInterrupt y en este caso motorControl, que fue obtenida gracias a diversas plantillas que se encontraron en distintos artículos para hallar el control PID en lazo cerrado para ambos actuadores. Para ello se utilizó la sentencia “include”, más el nombre de la carpeta que conlleva la librería. En la Figura 140 se observa cómo se importaron las librerías.

Motor_Control

```
#include "PinChangeInterrupt.h" //Librería para poder utilizar todos los pines de arduino como interrupciones
#include "motorControl.h" //Librería para controlar en lazo cerrado los motores
```

Figura 140. Importación de librerías en el IDE de Arduino.

Se tiene el tiempo de muestreo el cual va a ser de 100 milisegundos y está declarada como una variable sin signo, seguido de esto se crean 2 objetos para ambos motores y se envía el tiempo de muestreo para cada uno de ellos. En la Figura 141 se encuentra el fragmento del código explicado.

```
unsigned long lastTime, sampleTime = 100; //Variable sin signo para utilizar en tiempo de muestreo y declaro variable de tiempo de muestreo

motorControl motor1(sampleTime);  //Creamos un objeto y enviamos el tiempo de muestreo al motor 1
motorControl motor2(sampleTime);  //Creamos un objeto y enviamos el tiempo de muestreo al motor 2
```

Figura 141. Declaración del tiempo de muestreo para ambos motores.

Se realiza la comunicación serial, como pueden observar en la Figura 142, se tienen 2 datos, aquí lo que se procede a recibir del puerto serial va a ser la velocidad lineal de referencia y la velocidad angular de referencia para que el robot pueda moverse. Al final, el tipo de dataLength recibe el valor del ciclo de trabajo el cual regula la velocidad de los actuadores al set point.

```
//////////////////// COMUNICACION SERIAL ////////////////

String inputString = "";  //Variable donde almacena la cadena que empieza con valor vacio
bool stringComplete = false;  //Bandera para verificar si los datos llegaron completos
const char separator = ',';
const int dataLength = 2; //Recibe 2 datos: la velocidad lineal de referencia y la velocidad angular de referencia
double data[dataLength];  //Recibe el valor que regula el ciclo de trabajo (PWM) con esto regulamos la velocidad del motor osea el Sp
```

Figura 142. Comunicación Serial para las velocidades lineales y angulares.

En esta sección, se declaran las variables para los encoders, para el puente H y también las demás variables. Entonces, como se puede visualizar en la Figura 143, se tienen las variables para el motor derecho, en donde los canales han sido invertidos y declarados como constantes enteras, en este caso, el canal A y canal B. Luego la variable de control se la declara como entero y se la inicializa en cero. Seguido de esto, se tienen los pines que se utilizaron para el puente H, además, se tienen las variables para la lectura de los encoder, las cuales también están inicializadas en cero. Por último, se declara como dato de tipo double la variable de proceso y el setPoint, ambas se las inicializa en cero.

De la misma manera, como pueden ver en la Figura 144, las líneas de código del motor izquierdo se repiten, simplemente se cambian de pines de cada variable, como lo son los canales respectivamente y también los nombres al momento de cada declaración.

```
//////////////////////MOTOR DERECHO//////////////////////////////
//// Ojo se ha invertido canales////////

const int    C1R = 2;    // Entrada de la señal A del encoder.
const int    C2R = 3;    // Entrada de la señal B del encoder.
int outValueR = 0;       //Variable de control, por defecto empieza en cero

//// Puente H L293D ////
const int    in1 = 7;    //Pin para el sentido de giro del motor (derecho)
const int    in2 = 8;    //Pin para el sentido de giro del motor (derecho)
const int    enA = 6;    //Pin de activación del enable de la señal A en el punte H

//////////////Variables para lectura de encoder////////////

volatile int countR = 0; //Numero de cuentas que va a tener el encoder
volatile int antR = 0;  //Estado anterior
volatile int actR = 0;  //Estado actual
double wR = 0;  //Variable proceso (Velocidad angular en rad/s)
double w1Ref = 0;  //Setpoint (Velocidad angular de referencia en rad/s)
```

Figura 143. Fragmento de programación del motor derecho.

```
//////////////////////MOTOR IZQUIERDO//////////////////////////////

const int    C1L = 5;    // Entrada de la señal A del encoder.
const int    C2L = 4;    // Entrada de la señal B del encoder.
int outValueL = 0;      //Variable de control, por defecto empieza en cero

//// Puente H L293D ////
const int    in3 = 9;    //Pin para el sentido de giro del motor (izquiedo)
const int    in4 = 10;   //Pin para el sentido de giro del motor (izquierdo)
const int    enB = 11;   //Pin de activación del enable de la señal A en el punte H

//////////////Variables para lectura de encoder////////////

volatile int countL = 0;  //Numero de cuentas que va a tener el encoder
volatile int antL = 0;  //Estado anterior
volatile int actL = 0;  //Estado actual
double wL = 0;  //Variable proceso (Velocidad angular en rad/s)
double w2Ref = 0;  //Setpoint (Velocidad angular de referencia en rad/s)
```

Figura 144. Fragmento de programación del motor izquierdo.

En esta sección, se declara como constante el valor global de la velocidad del robot, que, en este caso es 3.1733. Seguido de esto, se tienen las variables de las distintas velocidades que tiene el robot, como se puede visualizar en la Figura 145, se declaran como dato de tipo "double" a la velocidad lineal, velocidad angular y al ángulo de orientación. Al final, se declara como constantes de tipo "double" al radio de la llanta y a la distancia entre cada una de ellas.

```
//////// VARIABLES PARA CALCULAR VELOCIDADES ANGULARES /////////

double constValue = 3.1733; // (1000*2*pi)/R ---> R = 1980 Resolucion encoder cuadruple

////////////////////////// ROBOT ////////////////////////////
double uRobot  = 0;  //Velocidad lineal mt/s
double wRobot  = 0;  //Velocidad angular mt/s
double phi = 0;  //Ángulo de orientación
const double R = 0.0335; // Radio de la llanta
const double d = 0.205; // Distancia entre llantas
```

Figura 145. Declaración de velocidades lineales y angulares del robot.

Aquí se declaran 2 pines digitales y 1 pin analógico, como se puede observar en la Figura 146 se tiene la declaración como dato de tipo entero de la batería el cual es para la simulación de un rango de voltaje entre 0 - 5V. Después, se declaran como constantes enteras los pines digitales 12 y 13, los cuales son de un buzzer y de un diodo led respectivamente.

```
//////////////////////////// DECLARACIÓN DE PINES ////////////////////////
int batteryPin = A4;   //Pin análogo de señal de la bateria simulada
const int buzzer = 12;       //Pin digital de señal del buzzer (zumbador)
const int LED = 13;       //Pin digital de señal del diodo led
```

Figura 146. Declaración del diodo led, buzzer y señal análoga para la batería del robot.

Una vez declarada las variables que se utilizaron para el desarrollo del control en ambos motores, se procede a realizar el "void setup". Se comienza colocando los baudios al puerto serial, que, en este caso son 9600; seguido se declara el voltaje como dato de tipo "double" inicializado en 0. Para finalizar en esta parte, se introduce la fórmula para obtener el voltaje simulado en el pin previamente declarado. En la Figura 147 se visualiza el fragmento del código mencionado.

```
void setup()
{
  Serial.begin(9600);  //Iniciar el puerto serial a 9600 baudios
  double voltage = 0.0;  //Declaro la batería como dato double
  voltage  = analogRead(batteryPin)*(5.0/1023.0);  //Obtener el voltaje simulado en pin A4

  /*////////////////// SINTONIA LAMBDA PID //////////////////

    motor.lambdaTunning(1.9281,0.1831,0.1337);  //(K,tau,delay)
    Serial.print()motor.getKc());
    Serial.print(", ");
    Serial.print()motor.getTi());
    Serial.print(", ");
    Serial.print()motor.getTd());
    Serial.print(", "); */
```

Figura 147. Inicialización del puerto serial a 9600 baudios y declaración de voltaje.

Previamente, cuando se realizó el control en lazo cerrado, se habían encontrado los valores del PID, en este caso refieren a Kc, Ti y Td; dichos valores se introducen con la sentencia "motor.setGains" para ambos motores. También se configuran los límites de las señales; la variable de control se la limita de 20 a 255 considerando una zona muerta y la variable de proceso se la limita de 0 a 11 rad/s. En la Figura 148 se aprecia la sección del código con la sintonía fina para ambos actuadores explicada recientemente.

```
////////////////// SINTONIA FINA MOTOR 1 /////////////////
motor1.setGains(0.21, 0.07, 0.05); // (Kc,Ti,Td)

////////////////// Limites de señales ////////////////////
motor1.setCvLimits(255, 20); //Limitamos señal de control de 20 a 255 (Considerar zona muerta)
motor1.setPvLimits(11, 0); //Velocidad angular max 11 rad/s y min 0 rad/s

////////////////// SINTONIA FINA MOTOR 2 ////////////////////
motor2.setGains(0.21, 0.07, 0.05); // (Kc,Ti,Td)

////////////////// Limites de señales ////////////////////
motor2.setCvLimits(255, 20); //Limitamos señal de control de 20 a 255 (Considerar zona muerta)
motor2.setPvLimits(11, 0); //Velocidad angular max 11 rad/s y min 0 rad/s
```

Figura 148. Valores del PID y límites de señales colocados en ambos motores.

En el siguiente fragmento se procede a hacer la declaración de pines E/S (Entradas/Salidas) como INPUT u OUTPUT respectivamente. Como se muestra en la figura 149, primero se declara como entrada las señales A y B de ambos encoder; seguido se declara como salida los pines de los sentidos de giro de cada motor; luego los enable de cada canal (A y B) como salida, y por último se inicializa en falso los pines del sentido de giro de ambos actuadores previamente declarados.

```
////////////////// DECLARACIÓN DE PINES E/S ////////////////////

pinMode(C1R, INPUT);  //Declaro como entrada la señal A del encoder derecho
pinMode(C2R, INPUT);  //Declaro como entrada la señal B del encoder derecho
pinMode(C1L, INPUT);  //Declaro como entrada la señal A del encoder izquierdo
pinMode(C2L, INPUT);  //Declaro como entrada la señal B del encoder izquierdo

pinMode(in1, OUTPUT);  //Declaro como salida el pin del sentido de giro del motor derecho
pinMode(in2, OUTPUT);  //Declaro como salida el pin del sentido de giro del motor derecho
pinMode(in3, OUTPUT);  //Declaro como salida el pin del sentido de giro del motor izquierdo
pinMode(in4, OUTPUT);  //Declaro como salida el pin del sentido de giro del motor izquierdo

pinMode(enA, OUTPUT);  //Declaro como salida el pin del sentido de giro del motor izquierdo
pinMode(enB, OUTPUT);  //Declaro como salida el pin del sentido de giro del motor izquierdo


digitalWrite(in1, false);  //Inicializo IN1 en falso
digitalWrite(in2, false);  //Inicializo IN2 en falso
digitalWrite(in3, false);  //Inicializo IN3 en falso
digitalWrite(in4, false);  //Inicializo IN4 en falso
```

Figura 149. Declaración de pines como entrada y salida.

Los pines de interrupción es un factor fundamental en este proyecto, ya que es mediante ellos que se pueden activar los canales A y B en cada encoder. En la Figura 150 se puede apreciar que se utilizó la sentencia "attachInterrupt" para las variables C1R y C2R que tienen como pines digitales 2 y 3 respectivamente para el encoder derecho; en cambio para el encoder izquierdo se debe añadir una palabra a la sentencia mencionada, en este caso es "attachPinChangeInterrupt" esto gracias a la librería importada al inicio y aquí se ubican las variables C1L y C2L que contiene los pines 4 y 5 respectivamente.

```
////////////////// PINES DE INTERRUPCION //////////////////

attachInterrupt(digitalPinToInterrupt(C1R), encoderR, CHANGE); //Declaro C1R como interrupción (encoder derecho)
attachInterrupt(digitalPinToInterrupt(C2R), encoderR, CHANGE); //Declaro C2R como interrupción (encoder derecho)

attachPinChangeInterrupt(digitalPinToPinChangeInterrupt(C1L), encoderL, CHANGE); //Declaro C1L como interrupción (encoder izquierdo)
attachPinChangeInterrupt(digitalPinToPinChangeInterrupt(C2L), encoderL, CHANGE); //Declaro C2L como interrupción (encoder izquierdo)
```

Figura 150. Declaración de pines de interrupción para ambos encoders.

Con respecto al buzzer, se lo declara como salida y se inicializa en cero o en bajo para que, al momento de activar el robot, éste no emita algún sonido. También se ubica un "lasTime", como se observa en la Figura 151, el cual actualiza el tiempo anterior de del prototipo.

```
//////////////////////////// BUZZER ////////////////////////

pinMode(buzzer, OUTPUT); //Declaro como salida el pin donde se conecta el buzzer
digitalWrite(buzzer, LOW); //Incializo el buzzer en bajo

lastTime = millis();  //Actualiza el tiempo anterior
```

Figura 151. Declaración del buzzer como salida.

Después del "void setup" hay que dirigirse al "void loop", aquí se tiene una condición "if" que conlleva otra condición "for", donde si el dato que recibe es menor al valor inicial "i", éste último se incrementa, y por ende recibe los datos declarados. Estos datos que se recibe aquí, van a ser la velocidad lineal de referencia y la velocidad angular de referencia, entonces se procede a llamar a la función "velocityMotor" para convertir esas velocidades de referencia globales del robot en velocidades para cada motor y se pueda aplicar el controlador PID. En la Figura 152 se aprecia el desarrollo de la recepción de datos.

```
void loop() {

  ////////// SI RECIBE DATOS /////////////
  if (stringComplete)
  {
    for (int i = 0; i < dataLength ; i++)
    {
      int index = inputString.indexOf(separator);
      data[i] = inputString.substring(0, index).toFloat();
      inputString = inputString.substring(index + 1);
    }

    velocityMotor(data[0], data[1]);

    inputString = "";
    stringComplete = false;
  }
```

Figura 152. Fragmento del void loop para la recepción de datos.

En la Figura 153 se tiene el controlador PID, para empezar, se debe colocar una condición "if" donde se hace la comparación del tiempo actual con el tiempo anterior en milisegundos, si es mayor al tiempo de muestro pues procede a calcular las velocidades angulares de cada actuador del robot y ese tiempo se lo almacena en una variable "lasTime". Esas mismas velocidades angulares están almacenadas en la función "velocityMotor" la cual más abajo del código está su proceso.

También se debe de obtener la orientación del robot en radianes, esto para la orientación del mismo y se lo manda a imprimir en el monitor serial del IDE de Arduino. Al final del desarrollo de este fragmento, se debe devolver la variable de control a cada motor con la sentencia "motor.Compute"; y si es mayor a cero cada motor pues el sentido de giro pasa de horario a antihorario dependiendo del motor al que se esté refiriendo.

```
  ////////////////// CONTROLADOR PID ////////////////

  if (millis() - lastTime >= sampleTime) //Compara el tiempo actual con el tiempo anterior
  { //Y si es >= al tiempo de muestreo hacemos lo sgte
    wR = constValue * countR / (millis() - lastTime); //Calculamos velocidad angular rad/s motor der.
    wL = constValue * countL / (millis() - lastTime); //Calculamos velocidad angular rad/d motor izq.

    lastTime = millis();  //Almacenamos el tiempo actual
    countR = 0;  //Reiniciamos los pulsos motor der.
    countL = 0; //Reiniciamos los pulsos motor izq.

    velocityRobot(wR, wL); //Velocidades angulares del robot

    phi = phi + wRobot * 0.1; //Aqui obtenemos la orientación del robot en radianes
    Serial.println(phi, 3);

    battery();
    outValueR = motor1.compute(w1Ref, wR); //Devuelve la variable de control a motor der.
    outValueL = motor2.compute(w2Ref, wL); //Devuelve la variable de control a motor izq.

    if (outValueR > 0) clockwise(in2, in1, enA, outValueR); else anticlockwise(in2, in1, enA, abs(outValueR));
    if (outValueL > 0) anticlockwise(in3, in4, enB, outValueL); else clockwise(in3, in4, enB, abs(outValueL));
  }                                                                                          //"ab
}
```

Figura 153. Fragmento del controlador PID en la programación.

En lo que corresponde a la recepción de datos, como se puede observar en la Figura 154, esto se hace para el envío de datos a Python y que éste último los imprima en una interfaz gráfica y sea de mejor manejo para la activación de variables por el usuario. Se comienza creando un evento para que mientras existan datos, lea cada uno de los caracteres, concatena cada uno de los valores en el string y compara la bandera si es igual a un salto de línea. Luego de esto se tiene la activación del diodo led y el buzzer, se inicia declarando como dato de tipo "inChar" y se coloca una letra, en este caso con la letra "F" se activan ambos componentes y con la letra "B" los desactivan. También se cuenta con el cambio de batería simulada a batería real con el cambio de letra, en este caso de "G" a "H".

```
/////////////// RECEPCION DE DATOS ////////////////////
void serialEvent()
{
  while (Serial.available()) //Mientras exista datos
  {
    char inChar = (char)Serial.read(); //Leer cada uno de los caracteres
    inputString += inChar;  //Concatenar cada uno de los valores en el string
    if (inChar == '\n')  //Comparala bandera, si este es igual a un saltode linea
    {
      stringComplete = true;  //Levantar la bandera, recibe datos
    }
    ////////////LED Y BUZZER///////////////
    if (inChar == 'F')
    {
      digitalWrite(13, HIGH);
      digitalWrite(12, HIGH);
      velocityMotor(2, 0);
    }
    if (inChar == 'B')
    {
      digitalWrite(13, LOW);
      digitalWrite(12, LOW);
      velocityMotor(0, 0);
    }
    ////////////BATERÍA REAL Y BATERÍA SIMULADA///////////////
    if (inChar == 'G')
    {
      batteryPin = A4;
    }
    if (inChar == 'H')
    {
      batteryPin = A0;
    }
  }
}
```

Figura 154. Sección de cambio de variable con teclas del computador.

Las funciones para los encoder de precisión cuádruple sirven para comparar el estado anterior y actual e incrementar o decrementar un contador según la tabla. Para esta tarea, se debe leer un puerto completo que dependiendo en donde se encuentre conectado los pines puede variar. En la Figura 155 tenemos para el encoder derecho, en donde se utilizan los pines 2 y 3 del puerto D, por lo tanto, para leer el puerto completo usaremos PIND. Entonces considerando que los puertos de Arduino son de 8 bits, se puede aplicar una operación "and" entre el valor de PIND y 00001100 (12 decimal), entonces se debe hacer la comparación sin olvidar que los valores también cambian en base a la conexión de los pines, es decir, si hay un flanco en el canal A (pin 3) y nada en el canal B, (pin 2) el valor es 10 (2 decimal) se debe de cambiar a 00001000 (8 decimal) y así en todas las combinaciones 00, 01 y 11. En la Figura 156 se tiene la función para el encoder izquierdo, donde es similar a la función anterior, cambiando solo la filtración de puertos en el PIND, en este caso sería 00110000 (48 decimal).

```
void encoderR(void) //Función del encoder der. para pecisión cuádruple
{
  antR = actR;  //Actualizamos los estados
  actR = PIND & 12; //Filtramos los pines del puerto D (2 y 3)

  if (antR == 0  && actR == 4)  countR++;
  if (antR == 4  && actR == 12)  countR++;
  if (antR == 8  && actR == 0)  countR++;
  if (antR == 12 && actR == 8)  countR++;

  if (antR == 0 && actR == 8)  countR--;
  if (antR == 4 && actR == 0)  countR--;
  if (antR == 8 && actR == 12) countR--;
  if (antR == 12 && actR == 4) countR--;

}
```

Figura 155. Función del encoder derecho para precisión cuádruple.

```
void encoderL(void)  //Función del encoder izq. para pecisión cuádruple
{
  antL = actL;  //Actualizamos los estados
  actL = PIND & 48; //Filtramos los pines del puerto D (4 y 5)

  if (antL == 0  && actL == 16)  countL++;
  if (antL == 16 && actL == 48)  countL++;
  if (antL == 32 && actL == 0)  countL++;
  if (antL == 48 && actL == 32)  countL++;

  if (antL == 0  && actL == 32)  countL--;
  if (antL == 16 && actL == 0)  countL--;
  if (antL == 32 && actL == 48)  countL--;
  if (antL == 48 && actL == 16)  countL--;

}

void clockwise(int pin1, int pin2, int analogPin, int pwm)
{
  digitalWrite(pin1, LOW);
  digitalWrite(pin2, HIGH);
  analogWrite(analogPin, pwm);
}
```

Figura 156. Función del encoder izquierdo para precisión cuádruple.

Se tiene ahora las funciones para el sentido de giro, como se puede ver en la Figura 157 para el sentido horario el pin1 se inicializa en bajo, el pin2 en alto y se declaran los PWM en un pin análogo. Para el sentido antihorario es similar, solo cambia que en el pin1 inicializa en alto y el pin2 en bajo.

```
void clockwise(int pin1, int pin2, int analogPin, int pwm)
{
  digitalWrite(pin1, LOW);
  digitalWrite(pin2, HIGH);
  analogWrite(analogPin, pwm);
}

void anticlockwise(int pin1, int pin2, int analogPin, int pwm)
{
  digitalWrite(pin1, HIGH);
  digitalWrite(pin2, LOW);
  analogWrite(analogPin, pwm);
}
```

Figura 157. Función para el sentido de giro horario y antihorario de cada motor.

Como penúltimo paso, vemos en la Figura 158 que se tiene la función de "velocityMotor" en donde se introducen las fórmulas de la velocidad angular, velocidad lineal y las velocidades referenciales de cada llanta en específico.

```
void velocityRobot(double w1, double w2)
{
  uRobot = (R * (w1 + w2)) / 2;
  wRobot = (R * (w1 - w2)) / d;
}
void velocityMotor(double u, double w)
{
  w1Ref = (u + (d * w / 2)) / R; //Velocidad llanta derecha
  w2Ref = (u - (d * w / 2)) / R; //Velocidad llanta izquierda
}
```

Figura 158. Función para las distintas velocidades del robot.

Para finalizar, se tiene la función de la batería en donde, como se muestra en la Figura 159, se declara la batería como dato de tipo "double" nuevamente, luego se obtiene el voltaje simulado en el pin A4 que previamente también se colocó arriba y al final se manda un print para poder visualizar en el monitor serial del IDE de Arduino.

```
void battery()  //Función batería
{
  double voltage = 0.0;  //Declaro la batería como dato double
  voltage  = analogRead(batteryPin)*(5.0/1023.0);  //Obtener el voltaje en pin A4
  Serial.println(voltage, 3);
}
```

Figura 159. Declaración de la función para la batería.

3.8. Desarrollo de la pantalla principal en Python.

La pantalla principal en Python es una gran ventaja y beneficia al usuario al momento de querer visualizar las demás prácticas con solo hacer un clic, esto se lo realiza para evitar ejecutar cada programa de manera individual, ya que esto a veces suele ser tedioso y a la larga puede generar complicaciones. Para ello, en una pestaña, se comienza importando las librerías necesarias para la visualización de las ventanas de Tkinter, la captura de imágenes a través de OpenCV y, la más importante, la libería Os-system. Esta útlima ayuda a ejecutar de manera eficaz en la misma pantalla cada práctica. En la Figura 160 se observa la importación de las librerías a utilizar.

Main.py

```
1 from tkinter import *
2 import cv2
3 from PIL import Image, ImageTk
4 import tkinter as tk
5 import tkinter.messagebox
6 from tkinter import filedialog
7 import os
```

Figura 160. Importación de librerías en el IDE de Atom.

Para una interfaz amigable con el usuario, se le puede añadir algunas imágenes de preferencia y así conseguir una buena impresión. Dichas imágenes son una fotografía del proyecto armado por completo y el logo de la universidad, tal como se aprecia en la Figura 161. Continuando con la programación de la pantalla principal, como se observa en la Figura 162, las imágenes mencionadas se las debe de llamar a través de la sentencia “path” y colocando su ubicación en la carpeta de archivos de su ordenador. Luego se tiene una función llamada “onClossing” creada con la sentencia “def”, esta tarea es la encargada de cerra la ventana creada y salir del bucle de eventos.

Figura 161. Imágenes a utilizar en la pantalla.

```
9  path = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/logoUPS.png'
10 path1 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/robot4.jpeg'
11
12 def onClossing():
13     root.quit()           #Salir del bucle de eventos
14     root.destroy()        #Destruye la ventana creada
```

Figura 162. Importación de imágenes y función de cierre de ventana.

Aquí se empieza a diseñar la pantalla principal, primero se define la variable “root” enlazándola con Tkinter mediante la sentencia “Tk()”, luego con esa misma variable se crean 3 diferentes objetos; el primero “root.protocol” es para poder cerrar la ventana de una manera correcta una vez que ya no esté siendo utilizada, segundo “root.title” es para ponerle título a la pestaña de la ventana, y tercero “root.geometry” es para dimensionar en pixeles el tamaño de la pantalla principal.

Luego, para ponerle texto dentro de la ventana se va a hacer uso de la sentencia “lbltext=Label”, con ello se imprime el título del proyecto junto con los autores del mismo, no obstante, se puede editar con algún tipo de letra en específico y ubicarlo en la ventana. Las dos últimas secciones, que van desde la línea 29 hasta la 37, son para mostrar las imágenes previamente llamadas en un inicio, esto lo hace con la sentencia “Image.PhotoImage(Image.open(path))”, y de igual manera se necesita ubicarla en la pantalla de manera que quede ordenada y no exista un cruce entre imágenes y textos. En la Figura 163 se observa el diseño de la pantalla principal utilizando objetos que brinda la librería Tkinter.

```
16 root = Tk()
17 root.protocol("WM_DELETE_WINDOW",onClossing)
18 root.title("PANTALLA PRINCIPAL")
19 root.geometry("510x650")
20
21 lbltext=Label(text="'Diseño e Implementación de Robot Móvil Multitarea con",font=("Siemens AD Sans",17,"italic")).place(x=8,y=160)
22 lbltext1=Label(text="Visión Artificial y Programación en Lenguaje Python",font=("Siemens AD Sans",17,"italic")).place(x=25,y=190)
23 lbltext1=Label(text="para la Universidad Politécnica Salesiana'",font=("Siemens AD Sans",17,"italic")).place(x=60,y=220)
24
25 lblautores=Label(text="Autores:",font=("Arial black",11, "italic")).place(x=220,y=250)
26 lblagudo=Label(text="Est. Richard Kevin Agudo Ube",font=("Ebrima",11,"italic")).place(x=150,y=270)
27 lblgomez=Label(text="Est. Franklin Paúl Gómez Zambrano",font=("Ebrima",11,"italic")).place(x=130,y=290)
28
29 img = ImageTk.PhotoImage(Image.open(path))
30 panel = tk.Label(root, image = img)
31 panel.pack(side = "top", fill = "both", expand = "yes")
32 panel.place(x=0, y=10)
33
34 img1 = ImageTk.PhotoImage(Image.open(path1))
35 panel1 = tk.Label(root, image = img1)
36 panel1.pack(side = "top", fill = "both", expand = "yes")
37 panel1.place(x=220, y=320)
```

Figura 163. Diseño de la pantalla principal.

Se tienen 5 funciones para poder ejecutar desde la misma pantalla cada práctica con solo un clic. Como se muestra en la Figura 164, aquí se definen 5 tareas, cada una realiza la misma función, la cual es llamar a la práctica requerida mediante la ubicación de esta última en la carpeta de archivos del ordenador. Esto se hace gracias a la librería Os-system mediante la sentencia “os.sytem”.

```
39  ##########################Funciones para llamar practicas#######################
40  def p1():
41      pr1 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/Practica1.py'
42      os.system('"%s"' % pr1)
43
44  def p2():
45      pr2 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/Practica2.py'
46      os.system('"%s"' % pr2)
47
48  def p3():
49      pr3 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/Practica3.py'
50      os.system('"%s"' % pr3)
51
52  def p4():
53      pr4 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/Practica4.py'
54      os.system('"%s"' % pr4)
55
56  def p5():
57      #notepad5 = 'D:/UPS-PAUL/PROYECTO DE TITULACION/Tesis_Agudo_Gómez/Practica5.py'
58      pr5 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/Practica5.py'
59      os.system('"%s"' % pr5)
```

Figura 164. Declaración de funciones para cada una de las prácticas realizadas.

Por último, una vez configuradas las funciones, lo que queda por realizar son los botones para cada unas de las prácticas del proyecto. Como se puede visualizar en la Figura 165, utilizando la sentencia “boton=Button” se puede agregar un texto a cada botón, como también editar el tipo de letra, devolviendo su variable con el método “command” y, también importante, la ubicación de cada botón para que aparezcan de manera ordenada en la ventana. Y al final, la función “root.mainloop()” es fundamental debido a que sin ella la pantalla principal no se puede ejecutar de manera eficaz.

```
61  ##################Botones de practicas#####################################
62  boton1=Button(root,text='PRACTICA #1',font=("Britannic Bold",18), command=p1).place(x=35, y=360, width=150, height=30)
63  boton2=Button(root,text='PRACTICA #2',font=("Britannic Bold",18), command=p2).place(x=35, y=410, width=150, height=30)
64  boton3=Button(root,text='PRACTICA #3',font=("Britannic Bold",18), command=p3).place(x=35, y=460, width=150, height=30)
65  boton4=Button(root,text='PRACTICA #4',font=("Britannic Bold",18), command=p4).place(x=35, y=510, width=150, height=30)
66  boton5=Button(root,text='PRACTICA #5',font=("Britannic Bold",18), command=p5).place(x=35, y=560, width=150, height=30)
67
68  root.mainloop()
```

Figura 165. Fragmento de programación del motor izquierdo.

3.9. Desarrollo de la programación de las prácticas en Python.

3.9.1. Desarrollo de la práctica #1.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**GUIA DE PRÁCTICAS**	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 1**
NOMBRE PRÁCTICA:	Configuración y manejo del robot inalámbricamente (Prueba de conexión, manejo a través de comandos en el computador.)	

1. Objetivo.

Configurar y manejar el robot de forma inalámbrica

2. Objetivos Específicos.

- Cargar el microcontrolador con el IDE de Arduino.
- Aprender a manejar las bases de Python para una conexión serial con el microcontrolador.
- Manejar el robot a través de comandos en el computador.
- Diseñar una interfaz GUI para los controles del robot.

3. Instrucciones.

- Abrir el IDE de Arduino.
- Compilar la programación “Motor_Control.ino” al Arduino NANO.
- Abrir el IDE de Atom.
- Desarrollar una programación de comandos en el computador en lenguaje Python.
- Utilizar la librería Tkinter para el diseño de controles del prototipo.

4. Desarrollo.

Paso N.º 1:

Se comienza importando las librerías necesarias en Python para realizar la programación de la práctica establecida. En la Figura 1 se observa la importación de cada una de las librerías requeridas.

Practica1.py

```
from pyArduino import *
from tkinter import *
from PIL import Image, ImageTk
import sys
import cv2
import numpy as np
import imutils
import tkinter as tk
import tkinter.messagebox
```

Figura 1. Importación de librerías a utilizar en Python.

Paso N.º 2:

Se crea una función, en este caso se la llama “onClosing()”, la cual permite cerrar de forma correcta las ventanas en Tkinter y a su vez que no quede ningún valor guardado en arduino. Seguido de esto se establecen las variables, las cuales son para insertar imágenes a la interfaz, en este caso el logo de la universidad y las flechas direccionales. En la Figura 2 se muestra la función mencionada y el directorio de cada imagen a utilizar.

```
def onClossing():
    arduino.sendData([0,0])
    arduino.close()
    root.quit()         #Salir del bucle de eventos.
    root.destroy()      #Destruye la ventana creada

path = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/logoUPS.png'
path_flecha_abajo_1 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/abajo_off.jpeg'
path_flecha_abajo_2 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/abajo_on.jpeg'
path_flecha_arriba_1 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/arriba_off.jpeg'
path_flecha_arriba_2 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/arriba_on.jpeg'
path_flecha_derecha_1 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/derecha_off.jpeg'
path_flecha_derecha_2 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/derecha_on.jpeg'
path_flecha_izq_1 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/izquierda_off.jpeg'
path_flecha_izq_2 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/izquierda_on.jpeg'
path_boton_gris = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/paro_off.png'
path_boton_azul = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/paro_on.png'
```

Figura 2. Función de cierre de ventana de Tkinter e importación de imágenes.

Paso N.º 3:

Ahora se establecen varias funciones para accionar el robot mediante teclas en el computador, esto se lo hace para cada orientación (derecha, izquierda, adelante, atrás y paro). Dentro de cada función se llama a la imagen de la flecha correspondiente para una buena visualización en la interfaz gráfica. En la Figura 3 y Figura 4 se visualiza cada una de las funciones establecidas.

```
def adelante(w):
    arduino.sendData([0.3,0.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_2)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    print("adelante")

def derecha(d):
    arduino.sendData([0.0,-2.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_2)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    print("derecha")

def izquierda(a):
    arduino.sendData([0.0,2.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_2)
    btn_boton_parar.config( image=img_path_Boton_gris)
    print("izquierda")
```

Figura 3. Función adelante, derecha e izquierda desarrollada en Python.

```
def parar(s):
    arduino.sendData([0.0,0.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_azul)
    print("parar")

def atras(x):
    arduino.sendData([-0.3,0.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_2)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    print("atras")
```

Figura 4. Función parar y atrás desarrollada en Python.

Paso N.º 4:

Para poder accionar el robot de una manera diferente, en este caso con el control del mouse, de igual manera, se crea una función para cada orientación y a su vez se llama a la imagen de flecha correspondiente para una buena visualización en la interfaz gráfica. En la Figura 5 y Figura 6 se muestra la configuración de las funciones para poder accionar el robot mediante el mouse.

```
def toggle_flecha_abajo():
    btn_flecha_abajo.config(image=img_path_flecha_abajo_2)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    arduino.sendData([-0.3,0.0])

def toggle_flecha_derecha():
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_2)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    arduino.sendData([0.0,-2.0])

def toggle_flecha_arriba():
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_2)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    arduino.sendData([0.3,0.0])

def toggle_flecha_izq():
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_2)
    btn_boton_parar.config( image=img_path_Boton_gris)
    arduino.sendData([0.0,2.0])

```

Figura 5. Función abajo, derecha, arriba e izquierda para control por mouse.

```
def toggle_boton_parar():
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_azul)
    arduino.sendData([0.0,0.0])
```

Figura 6. Función parar mediante el control del mouse.

Paso N.º 5:

En esta sección se define el puerto serial para la comunicación con Arduino, tal como se muestra en la Figura 7.

```
122  ############################ Serial communication ############
123  port = 'COM7'
124  arduino = serialArduino(port)
125  arduino.readSerialStart()
```

Figura 7. Puerto de comunicación serial.

Paso N.º 6:

Se procede a diseñar la interfaz gráfica, como se observa en la Figura 8, aquí se define el tamaño en pixeles de nuestra ventana, de la misma manera se coloca un título. También se crean diferentes label para colocar textos, ajustar el tipo y tamaño de letra, además de su ubicación en la pantalla. Por último, declaro una variable con la librería de Tkinter para de esta manera llamar al path en donde está la imagen del logo de la universidad.

```
127  ############################ HMI design ################
128  root = Tk()
129  root.protocol("WM_DELETE_WINDOW",onClossing)
130  root.title("PRACTICA #1") # título de la ventana
131  root.geometry("525x600")    # Definir el tamaño de la ventana principal
132
133  lblcomandos=Label(text="Control del Robot",font=("Arial Narrow",12)).place(x=208,y=555)
134
135  lbltitle=Label(text="Práctica #1:",font=("Bahnschrift SemiBold SemiConden",14)).place(x=28,y=150)
136  lbltext=Label(text="Configuración y manejo del robot inalámbricamente",font=("Bahnschrift SemiCondensed",14)).place(x=115,y=150)
137  lbltext1=Label(text="(Prueba de conexión, manejo a través de comandos en el computador)",font=("Bahnschrift SemiCondensed",14)).place(x=2,y=180)
138
139  img = ImageTk.PhotoImage(Image.open(path))
140  panel = tk.Label(root, image = img)
141  panel.pack(side = "top", fill = "both", expand = "yes")
142  panel.place(x=10, y=0)
143
```

Figura 8. Configuración de tamaño y texto dentro de la pantalla.

Paso N.º 7:

En esta sección se llaman a las imágenes de las flechas direccionales para que sean visualizadas en la ventana de Tkinter, y se ajusta el tamaño en pixeles para que se ajusten y no ocupen tanto espacio. En la Figura 9 se tiene la adquisición de cada imagen de las flechas direccionales.

```
img_path_flecha_abajo_1 = ImageTk.PhotoImage(Image.open(path_flecha_abajo_1).resize((150,100)))
panel_flecha_abajo_1 = tk.Label(root, image = img_path_flecha_abajo_1)

img_path_flecha_abajo_2 = ImageTk.PhotoImage(Image.open(path_flecha_abajo_2).resize((150,100)))
panel_flecha_abajo_2 = tk.Label(root, image = img_path_flecha_abajo_2)

img_path_flecha_arriba_1 = ImageTk.PhotoImage(Image.open(path_flecha_arriba_1).resize((150,100)))
panel_flecha_arriba_1 = tk.Label(root, image = img_path_flecha_arriba_1)

img_path_flecha_arriba_2 = ImageTk.PhotoImage(Image.open(path_flecha_arriba_2).resize((150,100)))
panel_flecha_arriba_2 = tk.Label(root, image = img_path_flecha_arriba_2)

img_path_flecha_derecha_1 = ImageTk.PhotoImage(Image.open(path_flecha_derecha_1).resize((100,150)))
panel_flecha_derecha_1 = tk.Label(root, image = img_path_flecha_derecha_1)

img_path_flecha_derecha_2 = ImageTk.PhotoImage(Image.open(path_flecha_derecha_2).resize((100,150)))
panel_flecha_derecha_2 = tk.Label(root, image = img_path_flecha_derecha_2)

img_path_flecha_izq_2 = ImageTk.PhotoImage(Image.open(path_flecha_izq_2).resize((100,150)))
panel_flecha_izq_2 = tk.Label(root, image = img_path_flecha_izq_2)

img_path_flecha_izq_1 = ImageTk.PhotoImage(Image.open(path_flecha_izq_1).resize((100,150)))
panel_flecha_izq_1 = tk.Label(root, image = img_path_flecha_izq_1)

img_path_Boton_azul = ImageTk.PhotoImage(Image.open(path_boton_azul).resize((150,100)))
panel_Boton_azul = tk.Label(root, image = img_path_Boton_azul)

img_path_Boton_gris = ImageTk.PhotoImage(Image.open(path_boton_gris).resize((150,100)))
panel_Boton_gris = tk.Label(root, image = img_path_Boton_gris)
```

Figura 9. Adquisición de imágenes de las flechas.

Paso N.º 8:

Antes de culminar, aquí se importan los path de cada flecha en cada variable para que cuando sea pulsada active la variable y haga el cambio de imagen correspondiente. Además, se posiciona cada una de las imágenes en el plano X y Y, además, de poder ajustar el ancho y alto de los botones, como se contempla en la Figura 10 en el HMI.

```
btn_flecha_abajo = Checkbutton(root, width=12, indicator=False,
                   command=toggle_flecha_abajo, image=img_path_flecha_abajo_1)
btn_flecha_abajo.place(x=190, y=440, width=150, height=100)

btn_flecha_derecha = Checkbutton(root, width=12, indicator=False,
                   command=toggle_flecha_derecha, image=img_path_flecha_derecha_1)
btn_flecha_derecha.place(x=340, y=315, width=100, height=150)

btn_flecha_arriba = Checkbutton(root, width=12, indicator=False,
                   command=toggle_flecha_arriba, image=img_path_flecha_arriba_1)
btn_flecha_arriba.place(x=190, y=242, width=150, height=100)

btn_flecha_izq= Checkbutton(root, width=12,  indicator=False,
                   command=toggle_flecha_izq, image=img_path_flecha_izq_1)
btn_flecha_izq.place(x=90, y=315, width=100, height=150)

btn_boton_parar= Checkbutton(root, width=12,  indicator=False,
                   command=toggle_boton_parar, image=img_path_Boton_gris)
btn_boton_parar.place(x=190, y=342, width=150, height=100)
```

Figura 10. Configuración de los botones para cada flecha.

Paso N.º 9:

Para finalizar, aquí se devuelven las variables mediante el comando “root.bind()”, para que siempre se pueda direccionar el robot en distintas posiciones y al final con la ayuda de la función “root.mainloop()” permite que se habilite la ventana de esta práctica, como se puede apreciar en la Figura 11.

```
195  root.bind('<w>',adelante)
196  root.bind('<a>',izquierda)
197  root.bind('<d>',derecha)
198  root.bind('<s>',parar)
199  root.bind('<x>',atras)
200  root.mainloop()
201
```

Figura 11. Devolución de variables direccionales.

3.9.2. Desarrollo de la práctica #2.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**GUIA DE PRÁCTICAS**	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 2**
NOMBRE PRÁCTICA:	Telemetría robótica (Comunicación bidireccional, adquisición de datos del prototipo, configuración e implementación en microcontrolador e interfaz en computador).	

1. Objetivo.

Ejecutar una telemetría robótica para adquisición de datos.

2. Objetivos Específicos.

- Cargar el microcontrolador con el IDE de Arduino.
- Agregar el módulo bluetooth HC-05 envío de datos.
- Desarrollar un programa en el IDE de Atom para adquisición de datos.
- Agregar un buzzer y un diodo led para verificar telemetría.
- Observar parámetro de batería simulada y orientación a través de GUI.

3. Instrucciones.

- Abrir el IDE de Arduino.
- Compilar la programación “Motor_Control.ino” al Arduino NANO.
- Abrir el IDE de Atom.
- Importar las librerías necesarias para visualización en la interfaz HMI.
- Identificar la cámara USB a utilizar.

4. Desarrollo.

Paso N.º 1:

Se comienza importando las librerías necesarias en Python para realizar la programación de la práctica establecida, tal como se muestra en la Figura 1.

```
from pyArduino import *
from tkinter import *
from PIL import Image, ImageTk
import sys
import cv2
import numpy as np
import imutils
import tkinter as tk
import tkinter.messagebox
```

Figura 1. Importación de librerías requeridas en Python.

Paso N.º 2:

Se crea una función, en este caso se la llama “onClosing()”, la cual permite cerrar de forma correcta las ventanas en tkinter, automáticamente se cierra la cámara que se esté utilizando y a su vez que no quede ningún valor guardado en Arduino, para ello con el comando “arduino.sendData()” se inicializa en cero para lograr lo mencionado. Seguido de esto, se establecen las variables, las cuales son para insertar imágenes a la interfaz, todo esto, para poder visualizar de una mejor manera el encendido/apagado del diodo led y el buzzer. En la Figura 2 se puede visualizar la importación de las imágenes que se utilizan para el HMI.

```
def onClossing():
    arduino.sendData([0,0])
    arduino.close()
    root.quit()         #Salir del bucle de eventos.
    cap.release()       #Cerrar camara
    print("Camara USB desconectada")
    root.destroy()      #Destruye la ventana creada

path = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/logoUPS.png'
path_flecha_abajo_1 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/abajo_off.jpeg'
path_flecha_abajo_2 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/abajo_on.jpeg'
path_flecha_arriba_1 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/arriba_off.jpeg'
path_flecha_arriba_2 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/arriba_on.jpeg'
path_flecha_derecha_1 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/derecha_off.jpeg'
path_flecha_derecha_2 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/derecha_on.jpeg'
path_flecha_izq_1 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/izquierda_off.jpeg'
path_flecha_izq_2 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/izquierda_on.jpeg'
path_boton_gris = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/paro_off.png'
path_boton_azul = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/paro_on.png'
path_led_buzzer_on = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/led_buzzer_on.jpeg'
path_led_buzzer_off = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/led_buzzer_off.jpeg'
```

Figura 2. Función de cierre de ventana de Tkinter e importación de imágenes.

Paso N.º 3:

De igual manera que en la práctica #1, como se puede ver en la Figura 3, Figura 4 y Figura 5, una vez establecidas las funciones para accionar el robot mediante teclas en el computador, se añade la sentencia “def” para cada comando para poder llamar a la imagen correspondiente, en este caso la del diodo led y el buzzer. Dentro de cada función, se añade el comando “arduino.sendData()” para la adquisición de datos en Python y dependiendo de la orientación, pues, se agregan los números para direccionar , además se crea un label que permite visualizar dichas direcciones, y por último se colocan las coordenadas de ubicación en el HMI de este mensaje.

```
def adelante(w):
    arduino.sendData([0.3,0.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_2)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    btn_led_buzzer.config(image=img_path_led_buzzer_off)
    label_adelante = Label(root,text="Adelante")
    label_adelante.place(x=760, y=70)

def derecha(d):
    arduino.sendData([0.0,-2.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_2)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    btn_led_buzzer.config(image=img_path_led_buzzer_off)
    label_der = Label(root,text="Derecha")
    label_der.place(x=760, y=70)

def izquierda(a):
    arduino.sendData([0.0,2.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_2)
    btn_boton_parar.config( image=img_path_Boton_gris)
    btn_led_buzzer.config(image=img_path_led_buzzer_off)
    label_izq = Label(root,text="Izquierda")
    label_izq.place(x=760, y=70)
```

Figura 3. Función adelante, derecha e izquierda desarrollada en Python.

```
def parar(s):
    arduino.sendData([0.0,0.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_azul)
    btn_led_buzzer.config(image=img_path_led_buzzer_off)
    label_parar = Label(root,text="Paro         ")
    label_parar.place(x=760, y=70)

def atras(x):
    arduino.sendData([-0.3,0.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_2)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    btn_led_buzzer.config(image=img_path_led_buzzer_off)
    label_atras = Label(root,text="Atrás       ")
    label_atras.place(x=760, y=70)

def letraF(f):
    arduino.sendData("F")
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    btn_led_buzzer.config(image=img_path_led_buzzer_on)
    label_led_buzzer = Label(root,text="Led y Buzzer On")
    label_led_buzzer.place(x=710, y=215)
```

Figura 4. Función parar, atrás y letraf desarrollada en Python.

```
def letraB(b):
    arduino.sendData("B")
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    btn_led_buzzer.config(image=img_path_led_buzzer_off)
    label_led_buzzer = Label(root,text="Led y Buzzer Off")
    label_led_buzzer.place(x=710, y=215)
```

Figura 5. Función letrab desarrollada en Python.

Paso N.º 4:

Complementando el código para realizar esta práctica, en las funciones para accionar mediante el mouse el robot, se añade un label que permite visualizar en qué dirección se moviliza el robot, y por último se colocan las coordenadas de ubicación en el HMI de este mensaje. En la Figura 6 y Figura 7 se observa las funciones de los controles mediante el mouse.

```
def toggle_flecha_abajo():
    btn_flecha_abajo.config(image=img_path_flecha_abajo_2)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    arduino.sendData([-0.3,0.0])
    label_atras = Label(root,text="Atrás        ")
    label_atras.place(x=760, y=70)

def toggle_flecha_derecha():
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_2)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    arduino.sendData([0.0,-2.0])
    label_der = Label(root,text="Derecha")
    label_der.place(x=760, y=70)
```

Figura 6. Función abajo y derecha para control por mouse.

```
130
131 def toggle_flecha_arriba():
132     btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
133     btn_flecha_arriba.config( image=img_path_flecha_arriba_2)
134     btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
135     btn_flecha_izq.config( image=img_path_flecha_izq_1)
136     btn_boton_parar.config( image=img_path_Boton_gris)
137     arduino.sendData([0.3,0.0])
138     label_adelante = Label(root,text="Adelante")
139     label_adelante.place(x=760, y=70)
140
141 def toggle_flecha_izq():
142     btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
143     btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
144     btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
145     btn_flecha_izq.config( image=img_path_flecha_izq_2)
146     btn_boton_parar.config( image=img_path_Boton_gris)
147     arduino.sendData([0.0,2.0])
148     label_izq = Label(root,text="Izquierda")
149     label_izq.place(x=760, y=70)
150
151 def toggle_boton_parar():
152     btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
153     btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
154     btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
155     btn_flecha_izq.config( image=img_path_flecha_izq_1)
156     btn_boton_parar.config( image=img_path_Boton_azul)
157     arduino.sendData([0.0,0.0])
158     label_parar = Label(root,text="Paro          ")
159     label_parar.place(x=760, y=70)
```

Figura 7. Función arriba, izquierda y parar, para control por mouse.

Paso N.º 5:

Se crea una función, la cual es denominada “def callback()” en donde se hace la adquisición de datos de las variables globales que se tiene y que a su vez Arduino posee, en este caso la batería y la orientación del prototipo. Se añade una condición en donde se puede llamar a la imagen que está siendo procesada, todo esto a través de la librería OpenCV y Tkinter. Caso contrario, se cierra automáticamente la ventana con todas sus variables a través de la función “onClosing()”. En la Figura 8 se muestra la adquisición de la imagen de la cámara USB.

```python
def callback():

    ########### Adquisición de la Imagen ###########
    global var1, bateria, phi

    ret, frame = cap.read() # Leer Frame
    phi.set(arduino.rawData[0])
    bateria.set(arduino.rawData[1])


    varPhi.set("Orientacion: "+str(phi.get())+" [rad/s]")
    varBateria.set("Batería Simulada: "+str(bateria.get())+" [V]")

    if ret:
        img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        img = Image.fromarray(img)
        img.thumbnail((420,420))
        tkimage = ImageTk.PhotoImage(img)
        label.configure(image = tkimage)
        label.image = tkimage
        root.after(10,callback)

    else:
        onClossing()
```

Figura 8. Función para procesamiento y adquisición de la imagen.

Paso N.º 6:

En esta sección, como se puede ver en la Figura 9, se crea una condición en la cual, si la cámara USB es detectada, mediante la librería OpenCV es visualizada en la ventana de Tkinter, caso contrario se muestra un mensaje en donde indica que la cámara está desconectada. Además, se puede seleccionar la cámara de la laptop (0) o la cámara USB (1); otro tipo de cámara también puede ser la que brinda un dispositivo móvil, en este caso un Android.

Al final, se realiza la comunicación serial para la adquisición de datos de Arduino a Python, todo esto a través del COM7.

```python
cap = cv2.VideoCapture(1)
if cap.isOpened():
    print("Camara USB inicializada")
else:
    sys.exit("Camara USB desconectada")

#cap.open(url)
ret, frame = cap.read()

########################## Serial communication ###########

port = 'COM7'
sizeData = 2
arduino = serialArduino(port,sizeData=sizeData)
arduino.readSerialStart()
```

Figura 9. Configuración de la cámara USB y puerto serial.

Paso N.º 7:

Para complementar con el código de la práctica #1, tal como se observa en la Figura 10, se procede a diseñar la interfaz gráfica, aquí se define el tamaño en pixeles de nuestra ventana, de la misma manera se coloca un título. También, se crean diferentes label para colocar textos dentro de la interfaz mencionada. Con el comando "ImageTk.PhotoImage(image.open(path))" de la librería de Tkinter se llama al path en donde está la imagen del logo de la universidad. Por último, se crean dos label de tipo "String", para poder observar el nivel de batería simulada junto con la orientación del prototipo, y a su vez poder ubicarlas en la pantalla HMI de manera ordenada.

```
############################ HMI design ################

root = Tk()
root.protocol("WM_DELETE_WINDOW",onClossing)
root.title("PRACTICA #2") # titulo de la ventana
root.geometry("1000x620")    # Definir el tamaño de la ventana principal


lblvideo=Label(text="Visualización en Cámara",font=("Arial Narrow",12)).place(x=688,y=580)
lblcomandos=Label(text="Control del Robot",font=("Arial Narrow",12)).place(x=208,y=580)
lbldir=Label(text="Dirección:").place(x=705,y=70)

lbltitle=Label(text="Práctica #2:",font=("Bahnschrift SemiBold SemiConden",14)).place(x=25,y=150)
lbltext=Label(text=" Telemetría robótica (Comunicación bidireccional,  ",font=("Bahnschrift SemiCondensed",14)).place(x=115,y=150)
lbltext1=Label(text="adquisición de datos del prototipo, configuración e implementación",font=("Bahnschrift SemiCondensed",14)).place(x=4,y=180)
lbltext2=Label(text="en microcontrolador e interfaz en computador).",font=("Bahnschrift SemiCondensed",14)).place(x=80,y=210)
lbltext3=Label(text="Adquisición de Datos",font=("Bahnschrift SemiBold SemiConden",16)).place(x=670,y=35)

img = ImageTk.PhotoImage(Image.open(path))
panel = tk.Label(root, image = img)
panel.pack(side = "top", fill = "both", expand = "yes")
panel.place(x=5, y=0)

#Labels
phi = DoubleVar(root,0)
varPhi = StringVar(root,"Batería Simulada : 0.00")
labelPhi = Label(root, textvariable = varPhi)
labelPhi.place(x=690, y=90)

bateria = DoubleVar(root,0)
varBateria = StringVar(root,"Orientación : 0.00")
labelBateria = Label(root, textvariable = varBateria)
labelBateria.place(x=685, y=110)
```

Figura 10. Configuración de la cámara USB y puerto serial.

Paso N.º 8:

Se vuelve a llamar a las imágenes de manera similar a la práctica anterior, con la diferencia, que en este caso se añaden imágenes al led y al buzzer, ya sea, cuando estén habilitados y deshabilitados. Se configura el tamaño de las imágenes para poder tener una buena visualización de cada dato en el HMI. De igual manera, se realiza con las imágenes que ya se habían planteado en la práctica anterior, tal como se observa en la Figura 11.

```
244
245 img_path_flecha_abajo_1 = ImageTk.PhotoImage(Image.open(path_flecha_abajo_1).resize((150,100)))
246 panel_flecha_abajo_1 = tk.Label(root, image = img_path_flecha_abajo_1)
247
248 img_path_flecha_abajo_2 = ImageTk.PhotoImage(Image.open(path_flecha_abajo_2).resize((150,100)))
249 panel_flecha_abajo_2 = tk.Label(root, image = img_path_flecha_abajo_2)
250
251 img_path_flecha_arriba_1 = ImageTk.PhotoImage(Image.open(path_flecha_arriba_1).resize((150,100)))
252 panel_flecha_arriba_1 = tk.Label(root, image = img_path_flecha_arriba_1)
253
254 img_path_flecha_arriba_2 = ImageTk.PhotoImage(Image.open(path_flecha_arriba_2).resize((150,100)))
255 panel_flecha_arriba_2 = tk.Label(root, image = img_path_flecha_arriba_2)
256
257 img_path_flecha_derecha_1 = ImageTk.PhotoImage(Image.open(path_flecha_derecha_1).resize((100,150)))
258 panel_flecha_derecha_1 = tk.Label(root, image = img_path_flecha_derecha_1)
259
260 img_path_flecha_derecha_2 = ImageTk.PhotoImage(Image.open(path_flecha_derecha_2).resize((100,150)))
261 panel_flecha_derecha_2 = tk.Label(root, image = img_path_flecha_derecha_2)
262
263 img_path_flecha_izq_2 = ImageTk.PhotoImage(Image.open(path_flecha_izq_2).resize((100,150)))
264 panel_flecha_izq_2 = tk.Label(root, image = img_path_flecha_izq_2)
265
266 img_path_flecha_izq_1 = ImageTk.PhotoImage(Image.open(path_flecha_izq_1).resize((100,150)))
267 panel_flecha_izq_1 = tk.Label(root, image = img_path_flecha_izq_1)
268
269 img_path_Boton_azul = ImageTk.PhotoImage(Image.open(path_boton_azul).resize((150,100)))
270 panel_Boton_azul = tk.Label(root, image = img_path_Boton_azul)
271
272 img_path_Boton_gris = ImageTk.PhotoImage(Image.open(path_boton_gris).resize((150,100)))
273 panel_Boton_gris = tk.Label(root, image = img_path_Boton_gris)
274
275 img_path_led_buzzer_on = ImageTk.PhotoImage(Image.open(path_led_buzzer_on).resize((180,60)))
276 panel_led_buzzer_on = tk.Label(root, image = img_path_led_buzzer_on)
277
278 img_path_led_buzzer_off = ImageTk.PhotoImage(Image.open(path_led_buzzer_off).resize((180,60)))
279 panel_led_buzzer_off = tk.Label(root, image = img_path_led_buzzer_off)
280
```

Figura 11. Adquisición de imágenes de las flechas, diodo led y buzzer.

Paso N.º 9:

Una vez ubicados los path de cada imagen en la pantalla HMI, se procede a colocar los botones para el led y buzzer, con la diferencia de que estas botones se activan a través de las teclas ya seleccionadas anteriormente. A su vez, escoger de manera correcta las coordenadas de los botones para evitar que estas se ubiquen una encima de otra y no permita visualizar de manera eficaz cada dato. Aquí también se importan los path de cada flecha en cada variable para que cuando sea pulsada active la variable y haga el cambio de imagen correspondiente. Además, se posiciona cada una de las imágenes en el plano X y Y, además, de poder ajustar el ancho y alto de los botones, como se visualiza en la Figura 12 en el HMI.

```
281
282  label=Label(root) #image = imagen camara opencv / relief = decoracion de borde
283  label.grid(row=1, column=1, padx=540,pady=250)
284
285
286  btn_flecha_abajo = Checkbutton(root, width=12, indicator=False,
287                     command=toggle_flecha_abajo, image=img_path_flecha_abajo_1)
288  btn_flecha_abajo.place(x=190, y=470, width=150, height=100)
289
290  btn_flecha_derecha = Checkbutton(root, width=12, indicator=False,
291                     command=toggle_flecha_derecha, image=img_path_flecha_derecha_1)
292  btn_flecha_derecha.place(x=340, y=345, width=100, height=150)
293
294  btn_flecha_arriba = Checkbutton(root, width=12, indicator=False,
295                     command=toggle_flecha_arriba, image=img_path_flecha_arriba_1)
296  btn_flecha_arriba.place(x=190, y=272, width=150, height=100)
297
298  btn_flecha_izq= Checkbutton(root, width=12,  indicator=False,
299                     command=toggle_flecha_izq, image=img_path_flecha_izq_1)
300  btn_flecha_izq.place(x=90, y=345, width=100, height=150)
301
302  btn_boton_parar= Checkbutton(root, width=12,  indicator=False,
303                     command=toggle_boton_parar, image=img_path_Boton_gris)
304  btn_boton_parar.place(x=190, y=372, width=150, height=100)
305
306  btn_led_buzzer= Checkbutton(root, width=12,  indicator=False,
307                     command=toggle_boton_parar, image=img_path_led_buzzer_off)
308  btn_led_buzzer.place(x=670, y=155, width=180, height=60)
309
```

Figura 12. Configuración de los botones para cada flecha y activación del diodo led con el buzzer.

Paso N.º 10:

Finalmente, se añaden los root faltantes con las cuales están configurado el diodo led y el buzzer. De la misma manera con las demás letras ya seleccionadas antes para el movimiento del robot. Como se muestra en la Figura 13, se coloca el comando “root.mainloop()” para que se habilite la pantalla HMI, además de una función “root.after(10,callback)” para devolver las variables de los widgets de Tkinter.

```
10
11  root.bind('<w>',adelante)
12  root.bind('<a>',izquierda)
13  root.bind('<d>',derecha)
14  root.bind('<s>',parar)
15  root.bind('<x>',atras)
16  root.bind('<f>',letraF)
17  root.bind('<b>',letraB)
18  root.after(10,callback) #Es un método definido para todos los widgets tkinter.
19  root.mainloop()
```

Figura 13. Devolución de variables direccionales y función callback.

3.9.3. Desarrollo de la práctica #3.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**GUIA DE PRÁCTICAS**	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 3**
NOMBRE PRÁCTICA:	Configuración de sistema de visión artificial (Determinar colores, ubicación del robot en el plano, punto de carga).	

1. Objetivo.

Configurar un sistema de visión artificial para detección de colores.

2. Objetivos Específicos.

- Desarrollar un programa en el IDE de Atom.
- Aprender a hacer detección de colores a través de librería OpenCV.
- Calibrar el sistema de visión artificial para detectar un color específico.

3. Instrucciones.

- Abrir el IDE de Atom.
- Importar las librerías necesarias para la visualización de la interfaz HMI.
- Calibrar el modelo HSV de cada color en el plano.
- Identificar detección de colores por visión artificial.

4. Desarrollo.

Paso N.º 1:

Se importan las librerías a utilizar en Python. En la Figura 1 se observa la importación de cada una de las librerías requeridas, incluyendo el módulo OpenCV.

Practica3.py

```
1  from tkinter import *
2  import cv2
3  import numpy as np
4  import time
5  import imutils
6  import tkinter as tk
7  import tkinter.messagebox
8  from PIL import Image, ImageTk
9  import serial
10 import serial.tools.list_ports
11 from math import *
12 import sys
```

Figura 1. Importación de librerías.

Paso N.º 2:

Se crea una carpeta en donde se llama a la imagen del logo de la universidad a utilizar. Se ajusta el tamaño en pixeles de la cámara USB y se crea una función llamada “cv2.VideoCapture(1)”, la cual permite utilizar la cámara mencionada. Por último, se configura un mensaje mediante una condición “if”, que indica si la cámara esta inicializada o desconectada, tal como se muestra en la Figura 2.

```
path = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/logoUPS_opt.jpg'
dataDict = {}

camHeight = 450
camWidth = 450
#url='http://192.168.137.209:8080/shot.jpg'
cam = cv2.VideoCapture(1)       # Selecciona cámara ( 0 es el default de la primera cámara)

if cam.isOpened():
    print("Camara USB inicializada")
else:
    sys.exit("Camara USB desconectada")
```

Figura 2. Configuración de cámara USB.

Paso N.º 3:

En esta sección, se empieza a configurar el diseño que tuvo nuestra pantalla para ello, se utiliza el comando “tk.Tk()” para poder crear y editar el diseño de la misma, de igual manera, poder añadir un título y configurar el tamaño de la ventana. Luego, se procede a crear varios labels para poder insertar mediante textos el título de la práctica, además de llamar la imagen del logo de la UPS mediante el comando “ImageTk.PhotoImage(Image.open(path))”. Se finaliza con la creación de una subventana para poder observar la imagen que brinda la cámara USB. Esto se logra mediante el comando “tk.Toplevel(controllerWindow)” y a su vez también poder incluir un título. En la Figura 3 se observa la sección del código explicado.

```
controllerWindow = tk.Tk()
controllerWindow.title("PRÁCTICA #3")
controllerWindow.geometry("400x630")
controllerWindow.resizable(0, 0)

lbltext=Label(text="Práctica #3: Configuración de sistema de visión artificial",font=("Bahnschrift SemiCondensed",13)).place(x=10,y=120)
lbltext1=Label(text="(Determinar colores, ubicación del robot en ",font=("Bahnschrift SemiCondensed",13)).place(x=45,y=145)
lbltext2=Label(text="el plano, punto de carga)",font=("Bahnschrift SemiCondensed",13)).place(x=100,y=170)

img = ImageTk.PhotoImage(Image.open(path))
panel = tk.Label(controllerWindow, image = img)
panel.pack(side = "top", fill = "both", expand = "yes")
panel.place(x=0, y=0)

videoWindow = tk.Toplevel(controllerWindow)
videoWindow.title("Vídeo de cámara")
videoWindow.resizable(0, 0)  #Si la pantalla es modificable
lmain = tk.Label(videoWindow)
lmain.pack()
videoWindow.withdraw()
```

Figura 3. Creación de ventanas con la librería Tkinter.

Paso N.º 4:

Se crea una función llamada “def getMouseClickPosition(mousePosition)” como se ve en la Figura 4, la cual permite usar el mouse en el plano X y Y de la imagen procesada. Luego de esto, se realiza una función con el comando “showCameraFrameWindow()” , con esta operación se logra mostrar ventana de video en tiempo real; incluyendo 1 botón para poder acceder a visualizar el video de la cámara, esto con el comando “videoWindow.deiconify” si es verdadero, caso contrario con el comando “videoWindow.withdraw” desaparece la ventana de video.

```
def getMouseClickPosition(mousePosition):
    global mouseX,mouseY
    global getPixelColor
    mouseX,mouseY = mousePosition.x,mousePosition.y
    getPixelColor = True

showVideoWindow = False

def showCameraFrameWindow():
    global showVideoWindow, showGraph
    global BRetourVideoTxt
    if showVideoWindow == False:
        if showGraph == True:
            graphWindow.withdraw()

        videoWindow.deiconify()
        showVideoWindow = True
        BRetourVideo["text"] = "Quitar vídeo "
    else:
        videoWindow.withdraw()
        showVideoWindow = False
        BRetourVideo["text"] = "Mostrar vídeo"

showVideoWindow = False

showCalqueCalibrationBool = False
def showCalqueCalibration():
    global showCalqueCalibrationBool
    showCalqueCalibrationBool = not showCalqueCalibrationBool
```

Figura 4. Creación funciones para movilización del mouse y video de cámara USB.

Paso N.º 5:

En esta sección, se empieza insertando una función llamada “def endProgam()” para destruir la ventana en caso de ya no utilizar la pantalla principal de la práctica; seguido de declarar los sliders de los factores HSV y colocar valores por defecto. Pero, para poder deslizar los sliders, se utiliza una función llamada “def resetSlider()”, se crean 3 objetos llamando a las variables de la función anterior. Concluye con un “main”, en donde se

agregan los objetos globales declarando algunas variables como los factores HSV, los colores en pixeles de cada círculo, el ancho y alto de la cámara, y la imagen procesada, tal como se puede visualizar en la Figura 5.

```
102  def endProgam():
103      controllerWindow.destroy()
104
105  sliderHDefault = 15
106  sliderSDefault = 70
107  sliderVDefault = 70
108
109  def resetSlider():
110      sliderH.set(sliderHDefault)
111      sliderS.set(sliderSDefault)
112      sliderV.set(sliderVDefault)
113
114  def donothing():
115      pass
116
117  start_time = 0
118  def main():
119      start_timeFPS = time.time()
120      global H,S,V
121      global getPixelColor
122      global x,y, alpha, beta
123      global camWidth,camHeight
124      global timeInterval, start_time
125      global showVideoWindow
126      global imgHSV
```

Figura 5. Funciones de cierre de ventana y creación de sliders (H-S-V).

Paso N.º 6:

Una vez teniendo declarada las funciones requeridas, se procede a leer el frame que envía la cámara con el comando “ret, frame = cam.read()” y a su vez convirtiendo la imagen de BGR a HSV, debido a que la imagen por defecto viene dada en BGR (Blue-Green-Red). Luego, se realiza una condición “if” para que el programa pueda ejecutar el mapeo y procesamiento de diferentes colores mediante la cámara y que el usuario pueda apreciar. Al final, se debe de tener en cuenta las máscaras de cada color, como también el

contorno de las mismas. En la Figura 6 se aprecia el procesamiento de colores utilizando el factor HSV.

```
    _, img=cam.read()

    ret, frame = cam.read() # Leer Frame
    img = img[0:int(camHeight),int((camWidth-camHeight)/2):int(camWidth-((camWidth-camHeight)/2))] #[Y1:Y2,X1:X2]

    imgHSV = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

    if getPixelColor == True and mouseY > 0 and mouseY < 480 and mouseX < 480:
        pixelColorOnClick = img[mouseY,mouseX]
        pixelColorOnClick = np.uint8([[pixelColorOnClick]])
        pixelColorOnClick = cv2.cvtColor(pixelColorOnClick,cv2.COLOR_BGR2HSV)
        H = pixelColorOnClick[0,0,0]
        S = pixelColorOnClick[0,0,1]
        V = pixelColorOnClick[0,0,2]
        T.delete("1.0","end")
        T.insert(tk.END, "H: " +str(H)+" S: " +str(S)+" V: " +str(V)+"\n")
        print(mouseX, mouseY)
        print(H, S, V )
        getPixelColor = False

    lowerBound=np.array([H-sliderH.get(),S-sliderS.get(),V-sliderV.get()])
    upperBound=np.array([H+sliderH.get(),S+sliderS.get(),V+sliderV.get()])

    mask=cv2.inRange(imgHSV,lowerBound,upperBound)
    mask = cv2.blur(mask,(6,6))                    # ajoute du flou a limage
    mask = cv2.erode(mask, None, iterations=2)     # retire les parasites

    cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours (cnts)
```

Figura 6. Procesamiento de colores por su máscara y contorno.

Paso N.º 7:

Se debe crear una condición para la visualización de la máscara del color y a su vez del contorno del mismo, esto se lo hace con el comando “showCalqueCalibrationBool”, y así también ayuda a retirar los residuos de cualquier ruido que exista en la imagen. Se diseña otra condición comparar cuantos contornos existen en las figuras procesadas, en este caso son mayores a cero debido a que se está utilizando múltiples círculos para el desarrollo de la práctica, tal como se visualiza en la Figura 7 de la programación en Python.

```
    if showCalqueCalibrationBool == True:
        mask = cv2.dilate(mask, None, iterations=2)        # retire les parasites
        res = cv2.bitwise_and(img,img, mask= mask)
        cv2.imshow('Mascara',mask)
        cv2.imshow('Solo Objeto',res)

    if len(cnts) > 0:
        c = max(cnts, key=cv2.contourArea)
        timeInterval = time.time() - start_time
        (x, y), radius = cv2.minEnclosingCircle(c)
        if radius > 10:
            cv2.putText(img,str(int(x)) + ";" + str(int(y)).format(0, 0),(int(x)-50, int(y)-50), cv2.FONT_HERSHEY_SIMPLEX,1, (255, 255, 255), 2)
            cv2.circle(img, (int(x), int(y)), int(radius),(0, 255, 255), 2)

            start_time = time.time()
    else:
        sommeErreurX, sommeErreurY = 0,0

    if showVideoWindow == True:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = Image.fromarray(img)
        imgtk = ImageTk.PhotoImage(image=img)
        lmain.imgtk = imgtk
        lmain.configure(image=imgtk)
    lmain.after(5, main)
```

Figura 7. Definición de número de contornos de la imagen procesada.

Paso N.º 8:

En esta parte de la programación se concentra en diseñar la pantalla principal que contempla esta práctica, se comienza insertando algunos textos como también se nombran los sliders que se utilizan para configurar los parámetros del factor HSV. En la Figura 8 se muestra el desarrollo del diseño de la interfaz GUI.

```
FrameVideoControl = tk.LabelFrame(controllerWindow, text="Controles de video")
FrameVideoControl.place(x=5,y=195,width=380)
BRetourVideo = tk.Button(FrameVideoControl, text="Mostrar video", command=showCameraFrameWindow)
BRetourVideo.pack()
BPositionCalibration = tk.Button(FrameVideoControl, text="Capa limites", command=showCalqueCalibration)
BPositionCalibration.place(x=280,y=0)

sliderH = tk.Scale(FrameVideoControl, from_=0, to=100, orient="horizontal", label="Hue", length=350, tickinterval = 10)
sliderH.set(sliderHDefault)
sliderH.pack()
sliderS = tk.Scale(FrameVideoControl, from_=0, to=100, orient="horizontal", label="Saturation", length=350, tickinterval = 10)
sliderS.set(sliderSDefault)
sliderS.pack()
sliderV = tk.Scale(FrameVideoControl, from_=0, to=100, orient="horizontal", label="Value", length=350, tickinterval = 10)
sliderV.set(sliderVDefault)
sliderV.pack()
```

Figura 8. Creación de textos en la pantalla principal de la práctica.

Paso N.º 9:

Casi finalizando, se crea un label para colocar los valores HSV, este mismo se lo ubica en el plano de manera que no interfiera con otras variables. También se realiza un botón “Reset” para poder resetear la programación y de igual manera otro botón “Salir” para cerrar la pantalla principal, tal como se observa en la Figura 9.

```
Frame_HSV = tk.LabelFrame(controllerWindow, text="Valores H, S, V ")
Frame_HSV.place(x=5,y=480,width=380, height=230)
T = tk.Text(Frame_HSV, height=5, width=30)
T.pack()
T.insert(tk.END, "Valores HSV escogidos\n")

BReset = tk.Button(controllerWindow, text = "Reset", command = resetSlider)
BReset.place(x=108, y=590)
BQuit = tk.Button(controllerWindow, text = "Salir", command = endProgam)
BQuit.place(x=200, y=590)
```

Figura 9. Creación de botones de Reset y Salir.

Paso N.º 10:

Para concluir, se crea un comando llamado “videoWindow.protocol("WM_DELETE_WINDOW",donothing)” el cual cuando se cierre la ventana, no quede ningún valor guardado, a su vez con el comando “videoWindow.bind("<Button-2>",getMouseClickPosition)” se desactiva la opción de selección del mouse para definición de cada color, y por útlimo se debe crear una función “tk.mainloop()” para la ejecución eficaz de la ventana. En la Figura 10 se muestra la sección del código detallado.

```
videoWindow.protocol("WM_DELETE_WINDOW",donothing)
videoWindow.bind("<Button-2>",getMouseClickPosition)

main()
tk.mainloop()
```

Figura 10. Comandos de deshabilitación de la pantalla principal.

3.9.4. Desarrollo de la práctica #4.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**GUIA DE PRÁCTICAS**	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 4**
NOMBRE PRÁCTICA:	Gestión multitarea autónoma. (Integración de tareas: determinación de colores y trayectoria entre varios puntos asignados).	

1. Objetivo.

Gestionar multitareas al prototipo de forma autónoma.

2. Objetivos Específicos.

- Integrar diversas tareas en el plano.
- Hacer detección de colores a través de la librería OpenCV.
- Calibrar el sistema de visión artificial para detectar un color específico.
- Lograr que el prototipo tenga una trayectoria entre varios puntos asignados.

3. Instrucciones.

- Abrir el IDE de Arduino.
- Compilar la programación “Motor_Control.ino” al Arduino NANO.
- Abrir el IDE de Atom.
- Importar las librerías necesarias para la visualización del prototipo.
- Desarrollar una programación en lenguaje Python.
- Verificar programación de distintas tareas en la interfaz gráfica.

4. Desarrollo.

Paso N.º 1:

Se empieza importando las librerías a utilizar en Python. En la Figura 1 se observa la importación de cada una de las librerías requeridas.

Practica4.py

```
1  from pyArduino import *
2  from tkinter import *
3  from PIL import Image, ImageTk
4  import cv2
5  import numpy as np
6  import sys
7  import tkinter as tk
8  import tkinter.messagebox
```

Figura 1. Importación de librerías.

Paso N.º 2:

Se define el cambio de estado y el cambio de variable, los cuales se las inicializa en cero, esto con la finalidad de que cada vez que se desee accionar el robot no quede ningún valor guardado. Además, con la sentencia "path" se inserta el directorio en donde se encuentra la imagen del logo de la universidad, tal como se muestra en la Figura 2.

```
10  g_estado =0
11  b=0
12
13  path = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/logoUPS_opt.jpg'
14
```

Figura 2. Definición de estados y directorio de imagen a utilizar.

Paso N.º 3:

En esta sección, se definen seis funciones con el comando "def" los cuales, luego se los va a incluir para la configuración de cada botón incluido en la práctica; se los llama mediante el comando "text=btnVar.get()". También, no obstante, se define la función "onClosing()"

para el cierre de las ventanas y para salir del bucle de eventos. En la Figura 3 se puede apreciar la sección del código explicado.

```
def toggle():
    btn.config(text=btnVar.get())

def toggle_aut():
    btntarea_aut.config(text=btnVar_aut.get())

def toggle1():
    btntarea1.config(text=btnVar1.get())

def toggle2():
    btntarea2.config(text=btnVar2.get())

def toggle3():
    btntarea3.config(text=btnVar3.get())

def toggle4():
    btntarea4.config(text=btnVar4.get())

def onClossing():
    arduino.sendData([0,0])
    arduino.close()
    root.quit()         #Salir del bucle de eventos.
    cap.release()       #Cerrar camara
    print("Camara USB desconectada")
    root.destroy()      #Destruye la ventana creada
```

Figura 3. Definición de funciones para las tareas.

Paso N.º 4:

Aquí se definen diferentes funciones de tipo entero para que, más después, poder utilizarlas en los diferentes sliders que representan los valores del factor HSV. Se definen 2 funciones en donde cada una tienen seis sliders, en la primera es para parametrizar el punto de referencia del robot y en la segunda para parametrizar los distintos colores que la cámara capte. La última función sirve para poder movilizar un punto dentro de la interfaz en el plano X y Y. En la Figura 4 se muestran las funciones creadas.

```
42  def hsvValue(int):
43      hMin.set(slider1.get())
44      hMax.set(slider2.get())
45      sMin.set(slider3.get())
46      sMax.set(slider4.get())
47      vMin.set(slider5.get())
48      vMax.set(slider6.get())
49
50  def hsvValuea(int):
51      hMina.set(slider1a.get())
52      hMaxa.set(slider2a.get())
53      sMina.set(slider3a.get())
54      sMaxa.set(slider4a.get())
55      vMina.set(slider5a.get())
56      vMaxa.set(slider6a.get())
57
58  def valor(int):
59      valorx.set(sliderx.get())
60      valory.set(slidery.get())
```

Figura 4. Creación de funciones para los sliders.

Paso N.º 5:

Como se observa en la Figura 5, se debe de realizar una función para la detección de los colores a través de la imagen que envía por frame la cámara, esto se lo hace con el comando “def objectDetection(rawImage)”; dentro de ella se declaran los valores globales de las máscaras, se inicializan los centroides del robot y de cada color, además de contar con una variable auxiliar en cero. Para la detección del contorno de cada color, se debe definir un área mínima para considerar que es un color.

Siguiendo con el procesamiento de la imagen, se debe convertir la imagen de BGR a espacio HSV con la función “cv2.cvtColor(rawImage, cv2.COLOR_BGR2HSV)”, además de definir los valores mínimos y máximos del factor HSV para cada sección. Por último, se establecen los parámetros para la detección de colores, aquí se debe de definir las máscaras de cada color independientemente, como también de los contornos de cada color en específico.

```
def objectDetection(rawImage):
    global a
    global mask, maska
    kernel = np.ones((10,10),np.uint8) # Filtro
    isObject = False     # Verdadero si encuentra un objeto
    cx,cy = 0,0          #centroide (x), centroide (y)
    cx1,cy1 = 0,0          #centroide (x), centroide (y)
    cxd,cyd = 0,0        #centroide (x) deseado, centroide (y) deseado
    cont = 0 # variable auxiliar
    minArea = 20  # Area minima para considerar que es un objeto

    ############### Procesamiento de la Imagen ###############

    hsv = cv2.cvtColor(rawImage, cv2.COLOR_BGR2HSV) #Convertirlo a espacio de color HSV

    lower=np.array([hMin.get(),sMin.get(),vMin.get()])
    upper=np.array([hMax.get(),sMax.get(),vMax.get()])

    lowera=np.array([hMina.get(),sMina.get(),vMina.get()])
    uppera=np.array([hMaxa.get(),sMaxa.get(),vMaxa.get()])

    azulBajo = np.array([100,100,20],np.uint8)
    azulAlto = np.array([125,255,255],np.uint8)

    #Deteccion de colores
    mask = cv2.inRange(hsv, lower, upper)
    maska = cv2.inRange(hsv, lowera, uppera)
    maskazul = cv2.inRange(hsv,azulBajo,azulAlto)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
    maska = cv2.morphologyEx(maska, cv2.MORPH_CLOSE, kernel)
    maskazul1 = cv2.morphologyEx(maskazul, cv2.MORPH_CLOSE, kernel)
    contours,_ = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contoursa,_ = cv2.findContours(maska.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours1,_ = cv2.findContours(maskazul.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Figura 5. Funciones para la detección y procesamiento de imagen.

Paso N.º 6:

En este fragmento se dedica a crear dos condiciones de tipo "for" para el centroide del robot y los círculos en el plano de la cámara. Dichas condiciones van a estar contando los contornos siempre y cuando el área de estos sea mayor al área mínima establecida anteriormente; en caso de cumplirse aquello, además el centroide del robot detecta si es mayor a 1 y rellena de color amarillo el punto de referencia con el comando "cv2.drawContours(rawImage,[cnt],0,(0,255,255),-1)" y hace el conteo de uno en uno. De la misma manera con los círculos. en cambio, pinta su contorno de color verde cada vez que sea detectado en el plano. En la Figura 6 se puede observar el fragmento con las condiciones brevemente explicadas.

```
97      ######################CENTROIDE ROBOT############################
98      for cnt in contours:
99          #Calcular el centro a partir de los momentos
100         momentos = cv2.moments(cnt)
101         area = momentos['m00']
102         if (area>minArea):
103             approx = cv2.approxPolyDP(cnt,0.01*cv2.arcLength(cnt,True),True)
104
105             if len(approx) > 1:
106                 cv2.drawContours(rawImage,[cnt],0,(0,255,255),-1)
107                 cx = int(momentos['m10']/momentos['m00'])
108                 cy = int(momentos['m01']/momentos['m00'])
109                 cont = cont+1
110     ############################CIRCULOS############################
111     for cnta in contoursa:
112         #Calcular el centro a partir de los momentos
113         momentos1 = cv2.moments(cnta)
114         area1 = momentos1['m00']
115         if (area1>minArea):
116             approx = cv2.approxPolyDP(cnta,0.01*cv2.arcLength(cnta,True),True)
117             if len(approx)>0:
118                 #Objeto
119                 cv2.drawContours(rawImage,[cnta],0,(255,0,255),-1)
120                 cxd = int(momentos1['m10']/momentos1['m00'])
121                 cyd = int(momentos1['m01']/momentos1['m00'])
122                 cont = cont+1
```

Figura 6. Creación condiciones para centroide del robot y círculos en el plano.

Paso N.º 7:

En la Figura 7 se muestra la función "callback()", la cual sirve para definir la adquisición de la imagen, como también leer el frame que esta brinda por medio de la cámara USB.

La condición "if ret" permite habilitar el frame que el color detecte en cada momento, además de definir la distancia mínima para la detección de cada círculo. Por último, se colocan también las coordenadas del robot y convertidas en pixeles para que no exista error de ejecución y el prototipo funcione de manera correcta.

```
def callback():

        ################ Adquisición de la Imagen ###########
        global maska,b
        global g_estado
        global distance
        global cxd, cyd
        global valors
        global var1
        #cap.open(url) # Antes de capturar el frame abrimos la url
        ret, frame = cap.read() # Leer frame

        if ret:

            isObject,mask,frame,cx,cy,cxd,cyd = objectDetection(frame)
            minDist = 37
            minDist1 = 20

            cv2.circle(frame,(cx,cy),10, (255,0,0), -1)
            cv2.circle(frame,(cxd,cyd),minDist, (0,255,0),3)


            if btnVar4.get() == 'Tarea 4 On':
                cxd = valorx.get()
                cyd = valory.get()
            if isObject:

                ################ Conversion coordenadas ###########
                hx = cx-frame.shape[1]/2
                hy = frame.shape[0]/2-cy

                ###################### Desired position in pixels #############
                hxd = cxd - frame.shape[1]/2
                hyd = frame.shape[0]/2 - cyd
```

Figura 7. Función callback para adquisición de imagen y coordenadas de la cámara.

Paso N.º 8

En este caso, como se muestra en la Figura 8, se realiza una comparación donde si la distancia resulta ser mayor a la distancia mínima, procede a recibir el dato del ángulo de orientación que envía Arduino a través del puerto de comunicación HC-05. Se tiene también, la determinación de errores de los puntos centroides del robot dentro de un "array"; además se incluyó la matriz del mismo robot rotado a 90° para que pueda desplazarse de manera eficiente en el plano.

```
            if distance>minDist:

                ################## Control cámara en mano ############
                a = 0.07 # punto de interes en metros (Donde se coloca el objeto)
                phi.set(arduino.rawData[0])

                # Errores
                hxe  =  hxd-hx
                hye  =  hyd-hy


                he = np.array([[hxe],[hye]]);

                K = np.diag([0.001,0.001])

                J = np.array([[-np.sin(phi.get()),-a*np.cos(phi.get())],
                              [ np.cos(phi.get()),-a*np.sin(phi.get())]])
                # Ley de control
                qp = np.linalg.inv(J)@(K@he);

                uRef.set(round(qp[0][0],3))
                wRef.set(round(qp[1][0],3))

            else:
                uRef.set(0)
                wRef.set(0)
                b=b+1

        else:

            uRef.set(0)
            wRef.set(0)
```

Figura 8. Declaración de fórmulas y matrices del modelo del robot.

Paso N.º 9:

Posteriormente, se comienza con la creación de botones para cada tarea mediante el uso de condiciones de tipo "if". Se debe de crear una condición de inicio "Start" para que comience con el funcionamiento de la práctica y empiece a recibir los datos de la velocidad lineal y angular. Como se ve en la Figura 9, la función de la tarea automática se ejecuta siempre y cuando la distancia sea menor a la distancia mínima del contorno del círculo; en caso de cumplirse aquello, pues, comienza a dirigirse a cada tarea haciendo cambios de estados mediante el factor HSV.

```
            if btnVar.get() == 'Start':
                arduino.sendData([uRef.get(),wRef.get()])
            if btnVar_aut.get() == 'Tarea Automatica On':
                if distance<minDist:
                    if g_estado == 0:
                        print("Punto A")
                    #agregar accion 1
                        slider1a.set(12)
                        slider3a.set(149)
                        slider5a.set(64)
                        slider2a.set(15)
                        slider4a.set(194)
                        slider6a.set(255)
                        g_estado = 1
                    elif g_estado == 1:
                        print("Punto B")
                        #agregar accion 2
                        slider1a.set(23)
                        slider3a.set(138)
                        slider5a.set(123)
                        slider2a.set(30)
                        slider4a.set(209)
                        slider6a.set(255)
                        g_estado = 2
                    elif g_estado == 2:
                        print("Punto C")
                        #agregar accion 2
                        slider1a.set(59)
                        slider3a.set(90)
                        slider5a.set(105)
                        slider2a.set(255)
                        slider4a.set(121)
                        slider6a.set(150)
                        g_estado = 0
```

Figura 9. Definición de funciones para la ejecución de la tarea automática.

Paso N.º 10:

De la misma manera que la sección anterior, en la Figura 10 se puede apreciar la creación de funciones para cada botón de las tareas individuales; en este caso cada función tiene distintos valores de factor HSV dependiendo del color que se haya escogido y parametrizado. Por útlimo, también se configura un botón para la tarea de seguimiento de punto en el plano, esto para que el robot siga la trayectoria de dicho punto acorde el usuario vaya moviendo el cursor del mouse

```
            if btnVar1.get() == 'Tarea 1 On':
                #Objeto Naranja
                slider1a.set(12)
                slider3a.set(149)
                slider5a.set(64)
                slider2a.set(15)
                slider4a.set(194)
                slider6a.set(255)
            if btnVar2.get() == 'Tarea 2 On':
                #Objeto Amarillo
                slider1a.set(23)
                slider3a.set(138)
                slider5a.set(123)
                slider2a.set(30)
                slider4a.set(209)
                slider6a.set(255)
            if btnVar3.get() == 'Tarea 3 On':
                #Color verde
                slider1a.set(59)
                slider3a.set(90)
                slider5a.set(105)
                slider2a.set(255)
                slider4a.set(121)
                slider6a.set(150)
            if btnVar4.get() == 'Tarea 4 On':
                cxd = valorx.get()
                cyd = valory.get()
                hxd = cxd - frame.shape[1]/2
                hyd = frame.shape[0]/2 - cyd
                isObject = True
                cv2.circle(frame,(cxd,cyd),minDist1, (0,255,255),3)
                print("Siguiendo el punto")
```

Figura 10. Definición de funciones para la ejecución de cada tarea individual.

Paso N.º 11:

Ahora, se define la cámara que se utilizó, en este caso como es una cámara USB se debe colocar el número 1, y se crea un "if" donde nos muestra mensaje indicando que la cámara ha sido inicializada. Así pues, no obstante, colocar el puerto serial que se está utilizando como lo es el "COM7" y se declara el tamaño de datos y la recepción de los mismos por Arduino. En la Figura 11 se muestra la configuración de la cámara y del puerto serial.

```
cap = cv2.VideoCapture(1)

if cap.isOpened():
    print("Camara USB inicializada")
else:
    sys.exit("Camara USB desconectada")

########################## Serial communication ##########

port = 'COM7'
sizeData = 2
arduino = serialArduino(port,sizeData=sizeData)
arduino.readSerialStart()
```

Figura 11. Configuración de la cámara USB y puerto serial.

Paso N.º 12:

Por consiguiente, se procede a diseñar la pantalla principal de la práctica establecida utilizando la librería Tkinter, se comienza con la declaración de la ventana por medio del comando "Tk()", seguido de la creación de objetos para colocar un título a la ventana y determinación de tamaño de la misma en pixeles.

Luego, se crean diferentes labels para introducir textos en la pantalla, en este caso se pone el título de la práctica; además de también poder editar el tipo de letra y tamaño del mismo. Después, se llama al path que se había declarado en un principio que contenía la imagen del logo de la universidad. También se ubica en la primera fila la imagen en tiempo real y en la segunda la imagen de las máscaras de los colores a detectar. Al final, se declara en la velocidad lineal y angular, variables de tipo "double" para poder introducir dichos datos en la pantalla y visualizar los valores que arroja del robot a recorrer. En la Figura 12 se puede observar la sección del diseño de la pantalla principal de esta práctica.

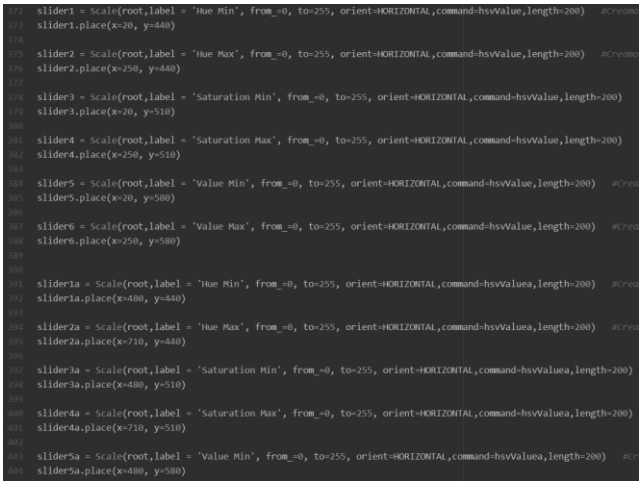
```
root = Tk()
root.protocol("WM_DELETE_WINDOW",onClossing)
root.title("PRÁCTICA #4") # título de la ventana
root.geometry("1366x768")    # Definir el tamaño de la ventana principal


lbltext=Label(text="Práctica #4: Gestión multitarea autónoma",font=("Bahnschrift SemiCondensed",15)).place(x=975,y=180)
lbltext1=Label(text="(Integración de tareas: determinación de colores y ",font=("Bahnschrift SemiCondensed",15)).place(x=945,y=210)
lbltext2=Label(text="trayectoria entre varios puntos asignados)",font=("Bahnschrift SemiCondensed",15)).place(x=975,y=240)


img = ImageTk.PhotoImage(Image.open(path))
panel = tk.Label(root, image = img)
panel.pack(side = "top", fill = "both", expand = "yes")
panel.place(x=935, y=35)

label=Label(root) #image = imagen camara opencv / relief = decoracion de borde
label.grid(padx=30,pady=20)

label1=Label(root)
label1.grid(row = 0,column=1,padx=0,pady=20)

uRef = DoubleVar(root,0)
varU = StringVar(root,"Linear velocity : 0.00")
labelU = Label(root, textvariable = varU)
labelU.grid(row = 1,padx=20,pady=10)

wRef = DoubleVar(root,0)
varW = StringVar(root,"Angular velocity : 0.00")
labelW = Label(root, textvariable = varW)
labelW.grid(row= 1,column = 1, padx=20,pady=10)
```

Figura 12. Diseño de la pantalla principal de la práctica.

Paso N.º 13:

Siguiendo con el desarrollo del diseño de la pantalla de esta práctica, como se puede visualizar en la Figura 13, ahora se tiene que diseñar los sliders previamente creados en las funciones anteriores; aquí se debe colocar el rango que tiene cada uno, como

también agregarle un texto y ubicarlos en la ventana de manera que no tope con las demás variables.

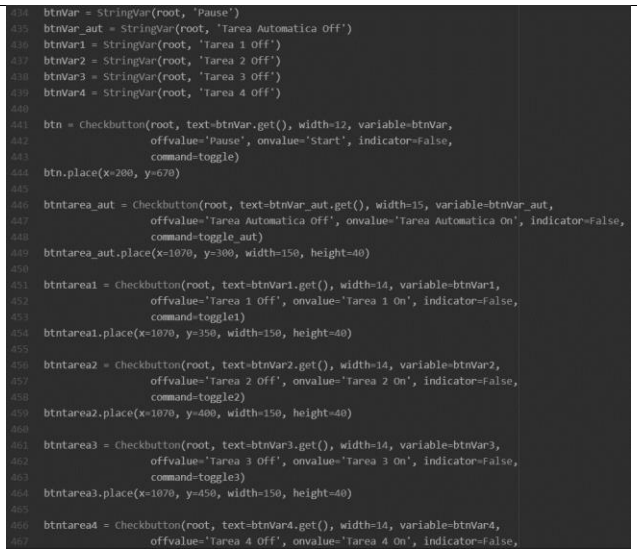


```
slider1 = Scale(root,label = 'Hue Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200)
slider1.place(x=20, y=440)

slider2 = Scale(root,label = 'Hue Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200)
slider2.place(x=250, y=440)

slider3 = Scale(root,label = 'Saturation Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200)
slider3.place(x=20, y=510)

slider4 = Scale(root,label = 'Saturation Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200)
slider4.place(x=250, y=510)

slider5 = Scale(root,label = 'Value Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200)
slider5.place(x=20, y=580)

slider6 = Scale(root,label = 'Value Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200)
slider6.place(x=250, y=580)


slider1a = Scale(root,label = 'Hue Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200)
slider1a.place(x=480, y=440)

slider2a = Scale(root,label = 'Hue Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200)
slider2a.place(x=710, y=440)

slider3a = Scale(root,label = 'Saturation Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200)
slider3a.place(x=480, y=510)

slider4a = Scale(root,label = 'Saturation Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200)
slider4a.place(x=710, y=510)

slider5a = Scale(root,label = 'Value Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200)
slider5a.place(x=480, y=580)
```

Figura 13. Creación de de rango de sliders en cada función.

Paso N.º 14:

Finalizando con el diseño de la pantalla, en la Figura 14 se acata el diseño de los botones para cada tarea, como también el de "Start". Aquí se define cada botón como variable de tipo "string" y obteniendo el dato declarado en inicio con el comando "btn = Checkbutton(root, text=btnVar.get())", se debe de indicar que comienza en falso para evitar error al momento de ejecutar cada uno y también ubicarlo en la ventana.



```
btnVar = StringVar(root, 'Pause')
btnVar_aut = StringVar(root, 'Tarea Automatica Off')
btnVar1 = StringVar(root, 'Tarea 1 Off')
btnVar2 = StringVar(root, 'Tarea 2 Off')
btnVar3 = StringVar(root, 'Tarea 3 Off')
btnVar4 = StringVar(root, 'Tarea 4 Off')

btn = Checkbutton(root, text=btnVar.get(), width=12, variable=btnVar,
                  offvalue='Pause', onvalue='Start', indicator=False,
                  command=toggle)
btn.place(x=200, y=670)

btntarea_aut = Checkbutton(root, text=btnVar_aut.get(), width=15, variable=btnVar_aut,
                  offvalue='Tarea Automatica Off', onvalue='Tarea Automatica On', indicator=False,
                  command=toggle_aut)
btntarea_aut.place(x=1070, y=300, width=150, height=40)

btntarea1 = Checkbutton(root, text=btnVar1.get(), width=14, variable=btnVar1,
                  offvalue='Tarea 1 Off', onvalue='Tarea 1 On', indicator=False,
                  command=toggle1)
btntarea1.place(x=1070, y=350, width=150, height=40)

btntarea2 = Checkbutton(root, text=btnVar2.get(), width=14, variable=btnVar2,
                  offvalue='Tarea 2 Off', onvalue='Tarea 2 On', indicator=False,
                  command=toggle2)
btntarea2.place(x=1070, y=400, width=150, height=40)

btntarea3 = Checkbutton(root, text=btnVar3.get(), width=14, variable=btnVar3,
                  offvalue='Tarea 3 Off', onvalue='Tarea 3 On', indicator=False,
                  command=toggle3)
btntarea3.place(x=1070, y=450, width=150, height=40)

btntarea4 = Checkbutton(root, text=btnVar4.get(), width=14, variable=btnVar4,
                  offvalue='Tarea 4 Off', onvalue='Tarea 4 On', indicator=False,
```

Figura 14. Creación de botones de cada tarea en específico.

Paso N.º 15:

Como se muestra en la Figura 15, colocamos el comando “root.mainloop()” para que habilite la pantalla HMI, además de una función “root.after(10,callback)” para devolver las variables de los widgets de Tkinter.

```
471  root.after(10,callback)
472  root.mainloop()
```

Figura 15. Comandos de deshabilitación de la pantalla principal.

3.9.5. Desarrollo de la práctica #5.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**GUIA DE PRÁCTICAS**	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 5**
NOMBRE PRÁCTICA:	Desarrollo de carga de batería del robot y movilización autónoma a punto de carga entre tareas.	

1. Objetivo.

Desarrollar un sistema de carga de batería del robot y movilizarlo autónomamente.

2. Objetivos Específicos.

- Determinar un rango de voltaje para identificar batería baja.
- Simular batería real de las baterías y batería simulada con potenciómetro.
- Movilizar el prototipo de forma autónoma al punto de carga entre tareas.

3. Instrucciones.

- Abrir el IDE de Arduino.
- Compilar la programación “Motor_Control.ino” al Arduino NANO.
- Abrir el IDE de Atom.
- Importar las librerías necesarias para la visualización del prototipo.
- Desarrollar una programación en lenguaje Python.
- Determinación de punto de carga en el plano.
- Verificar nivel de batería real y simulada en la interfaz gráfica.

4. Desarrollo.

Paso N.º 1:

Se comienza importando las librerías a utilizar en Python. En la Figura 1 se observa la importación de cada una de las librerías requeridas.

Practica5.py

```
1  from pyArduino import *
2  from tkinter import *
3  from PIL import Image, ImageTk
4  import cv2
5  import numpy as np
6  import sys
7  import tkinter as tk
8  import tkinter.messagebox
```

Figura 1. Importación de librerías.

Paso N.º 2:

Se define el cambio de estado y el cambio de variable, los cuales se los inicializa en cero, esto con la finalidad de que cada vez que se desee accionar el robot no quede ningún valor guardado. Además, se declara una variable "bateriav" la cual, también, se la inicializa en cero. Al final, con la sentencia “path” se inserta el directorio en donde se encuentra la imagen del logo de la universidad y algunas figuras para apreciar el nivel de la batería en la ventana, tal como se muestra en la Figura 2.

```
10  g_estado =0
11  b=0
12  valors = "A"
13  bateriav=0
14  path = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/logoUPS.png'
15  path_bvacia = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/b_vacia.jpeg'
16  path_binicial = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/b_inicial.jpeg'
17  path_bmedia = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/b_media.jpeg'
18  path_bfull = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/b_full.jpeg'
```

Figura 2. Definición de estados y directorio de imagen a utilizar.

Paso N.º 3:

En esta sección, se comienza definiendo dos funciones para el cambio de batería simulada a batería real mediante letras en el computador, con el comando "arduino.sendData()" se recibe el dato de Arduino y Python se encarga de hacer la gestión. También, no obstante, se definen seis funciones con el comando “def” los cuales, luego se los va a incluir para la configuración de cada botón incluido en la práctica; se los llama mediante el comando “text=btnVar.get()”. En la Figura 3 se puede apreciar la sección del código explicado.

```
def bsimulada(g):
    arduino.sendData("G")
    label_bsimu = Label(root,text="Batería Simulada")
    label_bsimu.place(x=710, y=592)

def breal(h):
    arduino.sendData("H")
    label_bre = Label(root,text="Batería Real        ")
    label_bre.place(x=710, y=592)

def letraF(f):
    arduino.sendData("F")

def letraB(b):
    arduino.sendData("B")

def toggle():
    btn.config(text=btnVar.get())

def toggle_aut():
    btntarea_aut.config(text=btnVar_aut.get())

def toggle1():
    btntarea1.config(text=btnVar1.get())
<
def toggle2():
    btntarea2.config(text=btnVar2.get())

def toggle3():
    btntarea3.config(text=btnVar3.get())

def toggle5():
    btntarea5.config(text=btnVar5.get())

def toggle_boton_parar():
    btn_boton_parar.config(image=img_path_Boton_azul)
    arduino.sendData([0.0,0.0])
```

Figura 3. Definición de funciones para las tareas.

Paso N.º 4:

Aquí se definen diferentes funciones de tipo entero para que, más después, se puedan utilizar en los diferentes sliders que representan los valores del factor HSV. Se definen 2 funciones en donde cada una tendrá seis sliders, en la primera es para parametrizar el punto de referencia del robot y en la segunda para parametrizar los distintos

colores que la cámara captará. La última función sirve para poder movilizar un punto dentro de la interfaz en el plano X y Y. En la Figura 4 se muestran las funciones creadas.

```
67  def hsvValue(int):
68      hMin.set(slider1.get())
69      hMax.set(slider2.get())
70      sMin.set(slider3.get())
71      sMax.set(slider4.get())
72      vMin.set(slider5.get())
73      vMax.set(slider6.get())
74
75  def hsvValuea(int):
76      hMina.set(slider1a.get())
77      hMaxa.set(slider2a.get())
78      sMina.set(slider3a.get())
79      sMaxa.set(slider4a.get())
80      vMina.set(slider5a.get())
81      vMaxa.set(slider6a.get())
82
83  def valor(int):
84      valorx.set(sliderx.get())
85      valory.set(slidery.get())
```

Figura 4. Creación de funciones para los sliders.

Paso N.º 5:

Como se observa en la Figura 5, se debe de realizar una función para la detección de los colores a través de la imagen que envía por frame la cámara, esto se lo hace con el comando “def objectDetection(rawImage)”; dentro de ella se declaran los valores globales de las máscaras, se los inicializa los centroides del robot y de cada color, además de contar con una variable auxiliar en cero. Para la detección del contorno de cada color, se debe definir un área mínima para considerar que es un color.

Siguiendo con el procesamiento de la imagen, se debe convertir la imagen de BGR a espacio HSV con la función “cv2.cvtColor(rawImage, cv2.COLOR_BGR2HSV)”, además de definir los valores mínimos y máximos del factor HSV para cada sección. Por último, se establecen los parámetros para la detección de colores, aquí se debe de definir las máscaras de cada color independientemente, como también de los contornos de cada color en específico.

```python
def objectDetection(rawImage):
    global a
    global mask, maska
    kernel = np.ones((10,10),np.uint8) # Filtro
    isObject = False     # Verdadero si encuentra un objeto
    cx,cy = 0,0          #centroide (x), centroide (y)
    cx1,cy1 = 0,0          #centroide (x), centroide (y)
    cxd,cyd = 0,0        #centroide (x) deseado, centroide (y) deseado
    cont = 0 # variable auxiliar
    minArea = 20  # Area minima para considerar que es un objeto

    ################ Procesamiento de la Imagen ###############

    hsv = cv2.cvtColor(rawImage, cv2.COLOR_BGR2HSV) #Convertirlo a espacio de color HSV

    #Se crea un array con las posiciones minimas y maximas capturadas de los sliders
    lower=np.array([hMin.get(),sMin.get(),vMin.get()])
    upper=np.array([hMax.get(),sMax.get(),vMax.get()])

    lowera=np.array([hMina.get(),sMina.get(),vMina.get()])
    uppera=np.array([hMaxa.get(),sMaxa.get(),vMaxa.get()])

    azulBajo = np.array([100,100,20],np.uint8)
    azulAlto = np.array([125,255,255],np.uint8)

    #Deteccion de colores
    mask = cv2.inRange(hsv, lower, upper)
    maska = cv2.inRange(hsv, lowera, uppera)
    maskazul = cv2.inRange(hsv,azulBajo,azulAlto)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
    maska = cv2.morphologyEx(maska, cv2.MORPH_CLOSE, kernel)
    maskazul1 = cv2.morphologyEx(maskazul, cv2.MORPH_CLOSE, kernel)
    contours,_ = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contoursa,_ = cv2.findContours(maska.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours1,_ = cv2.findContours(maskazul.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Figura 5. Funciones para la detección y procesamiento de imagen.

Paso N.º 6:

En este fragmento se concentra a crear dos condiciones de tipo "for" para el centroide del robot y los círculos en el plano de la cámara. Dichas condiciones van a estar contando los contornos siempre y cuando el área de estos sea mayor al área mínima establecida anteriormente; en caso de cumplirse aquello, además el centroide del robot detecta si es mayor a 1 y rellena de color amarillo el punto de referencia con el comando "cv2.drawContours(rawImage,[cnt],0,(0,255,255),-1)" y hace el conteo de uno en uno. De la misma manera con los círculos. En cambio, pinta su contorno de color verde cada vez que sea detectado en el plano. En la Figura 6 se puede observar el fragmento con las condiciones brevemente explicadas.

```
    for cnt in contours:
        #Calcular el centro a partir de los momentos
        momentos = cv2.moments(cnt)
        area = momentos['m00']
        if (area>minArea):
            approx = cv2.approxPolyDP(cnt,0.01*cv2.arcLength(cnt,True),True)

            if len(approx) > 1:
                cv2.drawContours(rawImage,[cnt],0,(0,255,255),-1)
                cx = int(momentos['m10']/momentos['m00'])
                cy = int(momentos['m01']/momentos['m00'])
                cont = cont+1

    for cnta in contoursa:
        #Calcular el centro a partir de los momentos
        momentos1 = cv2.moments(cnta)
        area1 = momentos1['m00']
        if (area1>minArea):
            approx = cv2.approxPolyDP(cnta,0.01*cv2.arcLength(cnta,True),True)
            if len(approx)>0:
                #Objeto
                cv2.drawContours(rawImage,[cnta],0,(255,0,255),-1)
                cxd = int(momentos1['m10']/momentos1['m00'])
                cyd = int(momentos1['m01']/momentos1['m00'])
                cont = cont+1

    if cont == 2:
        isObject = True
    else:
        isObject = False
```

Figura 6. Creación condiciones para centroide del robot y círculos en el plano.

Paso N.º 7:

En la Figura 7 se muestra la función "callback()", la cual sirve para definir la adquisición de la imagen, como también leer el frame que esta brinda por medio de la cámara USB.

La condición "if ret" permite habilitar el frame que el color detecte en cada momento, además de definir la distancia mínima para la detección de cada círculo. Por último, se colocan también las coordenadas del robot y convertidas en pixeles para que no exista error de ejecución y el prototipo funcione de manera correcta.

```
def callback():

        ################ Adquisición de la Imagen ###########
        global maska,b
        global g_estado
        global distance
        global cxd, cyd
        global valors
        global var1
        global bateriav
        #cap.open(url) # Antes de capturar el frame abrimos la url
        ret, frame = cap.read() # Leer Frame

        if ret:

            isObject,mask,frame,cx,cy,cxd,cyd = objectDetection(frame)
            minDist = 37

            cv2.circle(frame,(cx,cy),10, (255,0,0), -1)
            cv2.circle(frame,(cxd,cyd),minDist, (0,255,0),3)

            if isObject:

                ################ Conversion coordenadas ###########

                hx = cx-frame.shape[1]/2
                hy = frame.shape[0]/2-cy

                ###################### Desired position in pixels #############

                hxd = cxd - frame.shape[1]/2
                hyd = frame.shape[0]/2 - cyd

                # Distancia minima de error (Distancia Euclidiana)
                distance = np.sqrt((hxd-hx)**2+(hyd-hy)**2)
```

Figura 7. Función callback para adquisición de imagen y coordenadas de la cámara.

Paso N.º 8:

En este caso, como se muestra en la Figura 8, se realiza una comparación donde si la distancia resulta ser mayor a la distancia mínima, procede a recibir el dato del ángulo de orientación que envía Arduino a través del puerto de comunicación HC-05. Se tiene también, la determinación de errores de los puntos centroides del robot dentro de un "array"; además se incluyó la matriz del mismo robot rotado a 90° para que pueda desplazarse de manera eficiente en el plano.

```
                if distance>minDist:

                    ################## Control cámara en mano ############
                    a = 0.07 # punto de interes en metros (Donde se coloca el objeto)
                    phi.set(arduino.rawData[0])
                    bateria.set(arduino.rawData[1])
                    bateriav=arduino.rawData[1]

                    # Errores
                    hxe  =  hxd-hx
                    hye  =  hyd-hy

                    he = np.array([[hxe],[hye]]);

                    K = np.diag([0.001,0.001])

                    J = np.array([[-np.sin(phi.get()),-a*np.cos(phi.get())],
                                  [ np.cos(phi.get()),-a*np.sin(phi.get())]])
                    # Ley de control
                    qp = np.linalg.inv(J)@(K@he);

                    uRef.set(round(qp[0][0],3))
                    wRef.set(round(qp[1][0],3))
                    # if bateriav<2.0:
                    #     cxd = valorx.get()
                    #     cyd = valory.get()

                else:
                    uRef.set(0)
                    wRef.set(0)
                    b=b+1
            else:

                uRef.set(0)
                wRef.set(0)
```

Figura 8. Declaración de fórmulas y matrices del modelo del robot.

Paso N.º 9:

En esta sección de la programación, se procede a crear cuatro condiciones de tipo "if" para la visualización de nivel de batería por medio de cuatro figuras previamente importadas al inicio de la práctica. Para ello, en cada condición se debe llamar a la imagen con el comando "ImageTk.PhotoImage(Image.open(path).resize((90,20)))" para que de esta manera imprima la imagen según el nivel de batería declarado en cada condición. Como se ve en la Figura 9, se dividió cuando la batería sea menor a 0.1 V, mayor a 0.1 V y menor a 2.0 V, mayor o igual a 2.0 V y menor igual a 4.0, y finalizando cuando sea mayor a 4.0.

```
            if bateriav<0.1:
                imgv = ImageTk.PhotoImage(Image.open(path_bvacia).resize((90,20)))
                widget1 = Label(root, image=imgv)
                widget1.image = imgv
                widget1.place(x=750, y=530)

            if bateriav >0.1 and bateriav <2.0:

                imgi = ImageTk.PhotoImage(Image.open(path_binicial).resize((90,20)))
                widget2 = tkinter.Label(root, image=imgi)
                widget2.image = imgi
                widget2.place(x=750, y=530)

            if bateriav >=2.0 and bateriav <=4.0:
                imgm = ImageTk.PhotoImage(Image.open(path_bmedia).resize((90,20)))
                widget3 = Label(root, image=imgm)
                widget3.image = imgm
                widget3.place(x=750, y=530)

            if bateriav>4.0:
                #print("Cargado")
                imgf = ImageTk.PhotoImage(Image.open(path_bfull).resize((90,20)))
                widget4 = Label(root, image=imgf)
                widget4.image = imgf
                widget4.place(x=750, y=530)
```

Figura 9. Definición de número de contornos de la imagen procesada.

Paso N.º 10:

Posteriormente, se comienza con la creación de botones para cada tarea mediante el uso de condiciones de tipo "if". Se debe de crear una condición de inicio "Start" para que comience con el funcionamiento de la práctica y empiece a recibir los datos de la velocidad lineal y angular. Como se ve en la Figura 10, la función de la tarea automática se ejecuta siempre y cuando la distancia sea menor a la distancia mínima del contorno del círculo; en caso de cumplirse aquello, pues, comienza a dirigirse a cada tarea haciendo cambios de estados mediante el factor HSV.

```
260  ############################ Tareas ######################################
261              if btnVar.get() == 'Start':
262                  arduino.sendData([uRef.get(),wRef.get()])
263              if btnVar_aut.get() == 'Tarea Automatica On':
264                  if distance<minDist:
265                      if g_estado == 0:
266                      #agregar accion 1
267                          hMina.set(9)
268                          sMina.set(149)
269                          vMina.set(64)
270                          hMaxa.set(15)
271                          sMaxa.set(194)
272                          vMaxa.set(255)
273                          g_estado = 1
274                      elif g_estado == 1:
275                          #agregar acion 2
276                          hMina.set(18)
277                          sMina.set(138)
278                          vMina.set(123)
279                          hMaxa.set(30)
280                          sMaxa.set(209)
281                          vMaxa.set(255)
282                          g_estado = 2
283                      elif g_estado == 2:
284                          #agregar acion 2
285                          hMina.set(59)
286                          sMina.set(90)
287                          vMina.set(105)
288                          hMaxa.set(255)
289                          sMaxa.set(121)
290                          vMaxa.set(150)
291                          g_estado = 0
```

Figura 10. Definición de funciones para la ejecución de la tarea automática.

Paso N.º 11:

De la misma manera que la sección anterior, en la Figura 11 se puede apreciar la creación de funciones para cada botón de las tareas individuales; en este caso cada función tendrá distintos valores de factor HSV dependiendo del color que se haya escogido y parametrizado. Por útlimo, también se configura un botón para el punto de carga, esto para que el robot pueda dirigirse a dicho punto cada vez que su nivel de batería, sea real o simulada, sea menor a 2 V, caso contrario si es mayor a 4 V pues regresa a realizar la tarea o tareas que estaba haciendo.

```
            if btnVar1.get() == 'Tarea 1 On':
                #Objeto Naranja
                hMina.set(9)
                sMina.set(149)
                vMina.set(64)
                hMaxa.set(15)
                sMaxa.set(194)
                vMaxa.set(255)
            if btnVar2.get() == 'Tarea 2 On':
                #Objeto Amarillo
                hMina.set(18)
                sMina.set(138)
                vMina.set(123)
                hMaxa.set(30)
                sMaxa.set(209)
                vMaxa.set(255)
            if btnVar3.get() == 'Tarea 3 On':
                #Color verde
                hMina.set(59)
                sMina.set(90)
                vMina.set(105)
                hMaxa.set(255)
                sMaxa.set(121)
                vMaxa.set(150)
            if btnVar5.get() == 'Punto de Carga On':
                bateriav=arduino.rawData[1]
                #g_estado = 5
                if bateriav<2.0:
                    print("Punto de Carga")
                    hMina.set(0)
                    sMina.set(150)
                    vMina.set(30)
                    hMaxa.set(27)
                    sMaxa.set(205)
                    vMaxa.set(134)
                if bateriav>4.0:
                    print("Cargado")
                    g_estado == 0
```

Figura 11. Definición de funciones para la ejecución de cada tarea individual.

Paso N.º 12:

Ahora, se define la cámara que se utilizó, en este caso como es una cámara USB se debe de colocar el número 1, y se crea un "if" donde se muestra mensaje indicando que la cámara ha sido inicializada. Así pues, no obstante, colocar el puerto serial que se esté utilizando como lo es el "COM7" y se declara el tamaño de datos y la recepción de los mismos por Arduino. En la Figura 12 se muestra la configuración de la cámara y del puerto serial.

```
cap = cv2.VideoCapture(1)

if cap.isOpened():
    print("Camara USB inicializada")
else:
    sys.exit("Camara USB desconectada")

########################## Serial communication ##########

port = 'COM7'
sizeData = 2
arduino = serialArduino(port,sizeData=sizeData)
arduino.readSerialStart()
```

Figura 12. Configuración de la cámara USB y puerto serial

Paso N.º 13:

Por consiguiente, se procede a diseñar la pantalla principal de la práctica establecida utilizando la librería Tkinter, se comienza con la declaración de la ventana por medio del comando "Tk()", seguido de la creación de objetos para colocar un título a la ventana y determinación de tamaño de la misma en pixeles.

Luego, se crean diferentes labels para introducir textos en la pantalla, en este caso se pone el título de la práctica; además de también poder editar el tipo de letra y tamaño del mismo. Después, se llama al path que se había declarado en un principio que contenía la imagen del logo de la universidad. También se ubica en la primera fila la imagen en tiempo real y en la segunda la imagen de las máscaras de los colores a detectar. Al final, se declaran en la velocidad lineal, velocidad angular, ángulo de orientación y nivel de batería, variables de tipo "double" para poder introducirlos en la pantalla y visualizar los valores que arroja el robot mientras se desplaza por las tareas. En la Figura 13 se puede observar la sección del diseño de la pantalla principal de esta práctica.

```
root = Tk()
root.protocol("WM_DELETE_WINDOW",onClossing)
root.title("PRÁCTICA #5") # titulo de la ventana
root.geometry("950x715")    # Definir el tamaño de la ventana principal

lbltext=Label(text="Práctica #5: Desarrollo de carga de batería del robot ",font=("Bahnschrift SemiCondensed",15)).place(x=510,y=50)
lbltext1=Label(text="y movilización autónoma a punto de carga entre tareas.",font=("Bahnschrift SemiCondensed",15)).place(x=500,y=80)

img = ImageTk.PhotoImage(Image.open(path).resize((490,140)))
panel = tk.Label(root, image = img)
panel.pack(side = "top", fill = "both", expand = "yes")
panel.place(x=5, y=5)

label=Label(root) #image = imagen camara opencv / relief = decoracion de borde
label.place(x=30, y=170)

label1=Label(root)
label1.place(x=500, y=170)

uRef = DoubleVar(root,0)
varU = StringVar(root,"Linear velocity : 0.00")
labelU = Label(root, textvariable = varU)
labelU.place(x=150, y=500)

wRef = DoubleVar(root,0)
varW = StringVar(root,"Angular velocity : 0.00")
labelW = Label(root, textvariable = varW)
labelW.place(x=145, y=530)

phi = DoubleVar(root,0)
varPhi = StringVar(root,"Orientation : 0.00")
labelPhi = Label(root, textvariable = varPhi)
labelPhi.place(x=650, y=500)

bateria = DoubleVar(root,0)
varBateria = StringVar(root,"Nivel de Bateria : 0.00")
labelBateria = Label(root, textvariable = varBateria)
labelBateria.place(x=620, y=530)
```

Figura 13. Diseño de la pantalla principal de la práctica.

Paso N.º 14:

Finalizando con el diseño de la pantalla, en la Figura 14 se contempla el diseño de los botones para cada tarea, como también el de "Start". Aquí se define cada botón como variable de tipo "string" y obteniendo el dato declarado en inicio con el comando "btn = Checkbutton(root, text=btnVar.get())", se debe de indicar que comienza en falso para evitar error al momento de ejecutar cada uno y también ubicarlo en la ventana.

```
btnVar = StringVar(root, 'Pause')
btnVar_aut = StringVar(root, 'Tarea Automatica Off')
btnVar1 = StringVar(root, 'Tarea 1 Off')
btnVar2 = StringVar(root, 'Tarea 2 Off')
btnVar3 = StringVar(root, 'Tarea 3 Off')
btnVar5 = StringVar(root, 'Punto de Carga Off')
btn = Checkbutton(root, text=btnVar.get(), width=12, variable=btnVar,
                  offvalue='Pause', onvalue='Start', indicator=False,
                  command=toggle)
btn.place(x=415, y=670)

btntarea_aut = Checkbutton(root, text=btnVar_aut.get(), width=15, variable=btnVar_aut,
                  offvalue='Tarea Automatica Off', onvalue='Tarea Automatica On', indicator=False,
                  command=toggle_aut)
btntarea_aut.place(x=50, y=560, width=150, height=40)

btntarea1 = Checkbutton(root, text=btnVar1.get(), width=14, variable=btnVar1,
                  offvalue='Tarea 1 Off', onvalue='Tarea 1 On', indicator=False,
                  command=toggle1)
btntarea1.place(x=50, y=610, width=150, height=40)

btntarea2 = Checkbutton(root, text=btnVar2.get(), width=14, variable=btnVar2,
                  offvalue='Tarea 2 Off', onvalue='Tarea 2 On', indicator=False,
                  command=toggle2)
btntarea2.place(x=300, y=560, width=150, height=40)

btntarea3 = Checkbutton(root, text=btnVar3.get(), width=14, variable=btnVar3,
                  offvalue='Tarea 3 Off', onvalue='Tarea 3 On', indicator=False,
                  command=toggle3)
btntarea3.place(x=300, y=610, width=150, height=40)

btntarea5 = Checkbutton(root, text=btnVar5.get(), width=14, variable=btnVar5,
                  offvalue='Punto de Carga Off', onvalue='Punto de Carga On', indicator=False,
                  command=toggle5)
btntarea5.place(x=550, y=585, width=150, height=40)
```

Figura 14. Comandos de deshabilitación de la pantalla principal.

Paso N.º 15:

Como se muestra en la Figura 15, se coloca el comando “root.mainloop()” para que habilite la pantalla HMI, además de una función “root.after(10,callback)” para devolver las variables de los widgets de Tkinter.

```
root.bind('<g>',bsimulada)
root.bind('<h>',breal)
root.after(10,callback) #Es
root.mainloop()
```

Figura 15. Comandos de deshabilitación de la pantalla principal.

4. RESULTADOS

4.1. Práctica #1.

En la Figura 166 se observa ambas pantallas, a la izquierda se tiene la pantalla principal con todos los botones de las respectivas prácticas y a la derecha la ventana de la práctica #1, en la cual, se encuentran los controles para movilizar el robot.

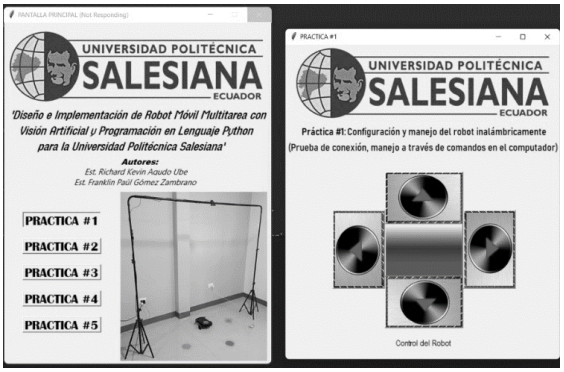


Figura 166. Pantalla principal y ventana de la práctica #1.

En la Figura 167 se visualiza el control del robot, en donde, a través de comandos del computador o con ayuda del cursor del mouse, se puede seleccionar el botón en la dirección a la que se desea que vaya el robot, en este caso, se encuentra pulsada la flecha que desplaza hacia adelante.

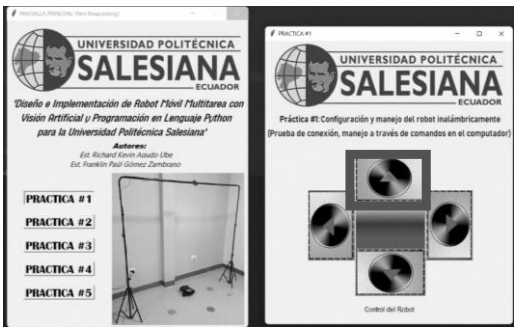


Figura 167. Control del robot con flecha pulsada hacia adelante.

A continuación, en la Figura 168 se muestra cómo se desplaza el robot hacia adelante.

Figura 168. Desplazamiento del robot hacia adelante.

En la Figura 169 se visualiza el control del robot, en donde, a través de comandos del computador o con ayuda del cursor del mouse, podemos seleccionar el botón en la dirección a la que se desea que vaya el robot, en este caso, se encuentra pulsada la flecha que desplaza hacia atrás.

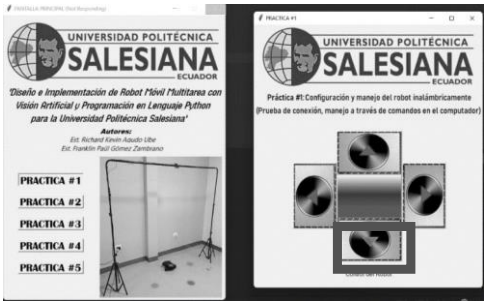


Figura 169. Control del robot con flecha pulsada hacia atrás.

A continuación, en la Figura 170 se muestra cómo se desplaza el robot hacia atrás.

Figura 170. Desplazamiento del robot hacia atrás.

En la Figura 171 se puede visualizar el control del robot, en donde, a través de comandos del computador o con ayuda del cursor del mouse, se selecciona el botón en la dirección a la que se desea que vaya el robot, en este caso, se encuentra pulsada la flecha que desplaza hacia la izquierda.

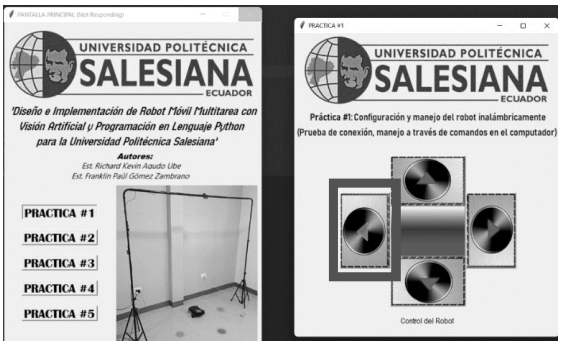


Figura 171. Control del robot con flecha pulsada hacia la izquierda.

A continuación, en la Figura 172 se muestra cómo se desplaza el robot hacia la izquierda.

Figura 172. Desplazamiento del robot hacia la izquierda.

En la Figura 173 se visualiza el control del robot, en donde, a través de comandos del computador o con ayuda del cursor del mouse, se puede seleccionar el botón en la dirección a la que se desea que vaya el robot, en este caso, se encuentra pulsada la flecha que desplaza hacia la derecha.

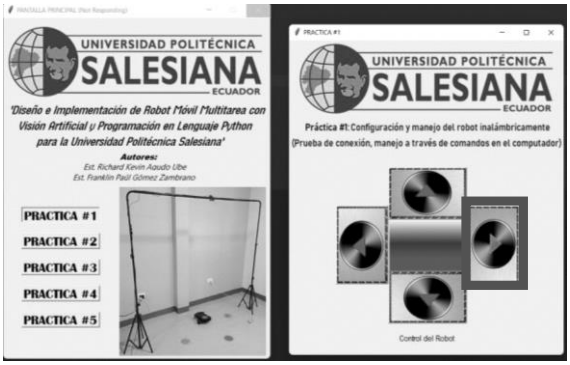


Figura 173. Control del robot con flecha pulsada hacia la derecha.

A continuación, en la Figura 174 se muestra cómo se desplaza el robot hacia la derecha.

Figura 174. Desplazamiento del robot hacia la derecha.

En la Figura 175 se visualiza el control del robot, en donde, a través de comandos del computador o con ayuda del cursor del mouse, se puede seleccionar el botón en la dirección a la que se desea que vaya el robot, en este caso, se encuentra pulsado el botón de paro.

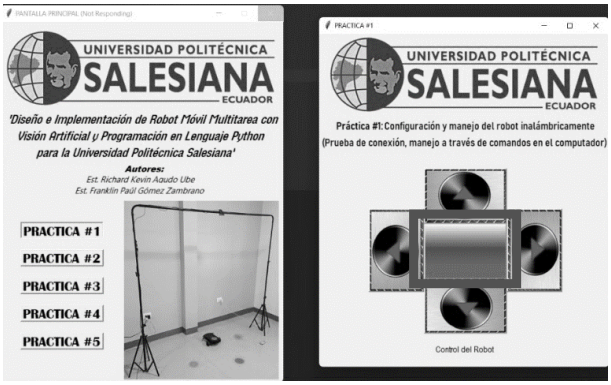


Figura 175. Control del robot con botón de paro pulsado.

A continuación, en la Figura 176 se muestra al robot en su estado inicial.

Figura 176. Robot detenido.

4.2. Práctica #2.

En la Figura 177 se observa el control del robot junto con unos labels que permiten visualizar los estados del robot a través de telemetría. Utilizando los comandos del computador o con ayuda del cursor del mouse, se puede seleccionar el botón en la dirección a la que se desea que vaya el robot, en este caso, se encuentra pulsada la flecha que desplaza hacia adelante.

Figura 177. Control del robot por telemetría con flecha pulsada hacia adelante.

En la Figura 178 se observa el control del robot junto con unos labels que permiten visualizar los estados del robot a través de telemetría. Utilizando los comandos del computador o con ayuda del cursor del mouse, se puede seleccionar el botón en la dirección a la que se desea que vaya el robot, en este caso, se encuentra pulsada la flecha que desplaza hacia la izquierda.

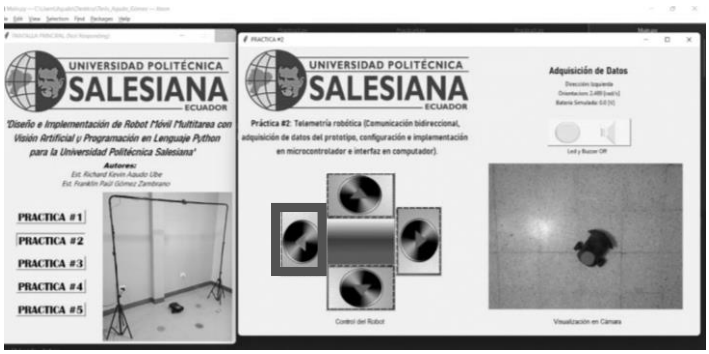


Figura 178. Control del robot por telemetría con flecha pulsada hacia la izquierda.

En la Figura 179 se observa el control del robot junto con unos labels que permiten visualizar los estados del robot a través de telemetría. Utilizando los comandos del computador o con ayuda del cursor del mouse, se puede seleccionar el botón en la dirección a la que se desea que vaya el robot, en este caso, se encuentra pulsada la flecha que desplaza hacia la derecha.

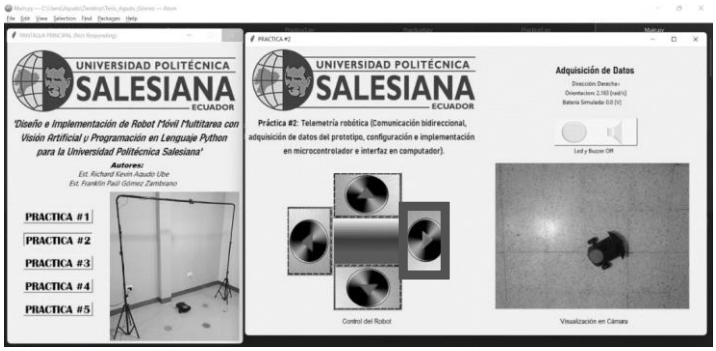


Figura 179. Control del robot por telemetría con flecha pulsada hacia la derecha.

En la Figura 180 se observa el control del robot junto con unos labels que permiten visualizar los estados del robot a través de telemetría. Utilizando los comandos del computador o con ayuda del cursor del mouse, se puede seleccionar el botón en la dirección a la que se desea que vaya el robot, en este caso, se encuentra pulsada la flecha que desplaza hacia atrás.

Figura 180. Control del robot por telemetría con flecha pulsada hacia atrás.

En la Figura 181 se observa el control del robot junto con unos labels que permiten visualizar los estados del robot a través de telemetría. Utilizando los comandos del computador o con ayuda del cursor del mouse, se puede seleccionar el botón en la dirección a la que se desea que vaya el robot, en este caso, el robot se encuentra detenido y tiene activado el diodo led y buzzer.

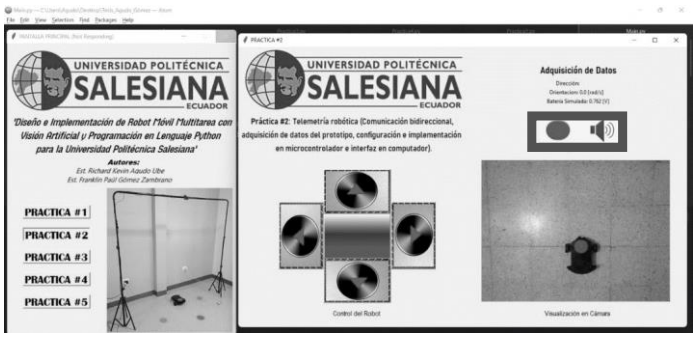


Figura 181. Control del robot por telemetría con activación de diodo led y buzzer.

4.3. Práctica #3.

En la Figura 182 se muestra la pantalla principal (Main) y la interfaz de la práctica #3, la cual, contiene dos botones, uno para habilitar las ventanas de las capas límites de cada color y el otro botón para cerrar dichas ventanas. Lleva incluido sliders que permiten reducir ruidos en cada color. Por último, se presenta una ventana en donde se observan los valores de HSV del color seleccionado.

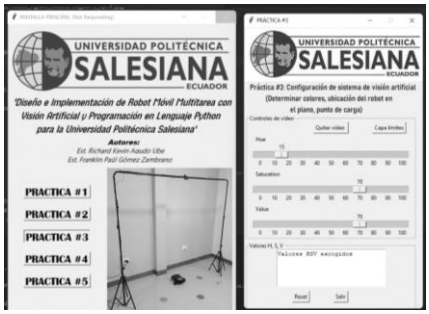


Figura 182. Pantalla principal y ventana de la práctica #3.

En la Figura 183 se observa la pantalla de video de cámara de la practica #3, junto con las ventanas de las máscaras de cada color en el plano, en este caso, se encuentra seleccionado el color azul.

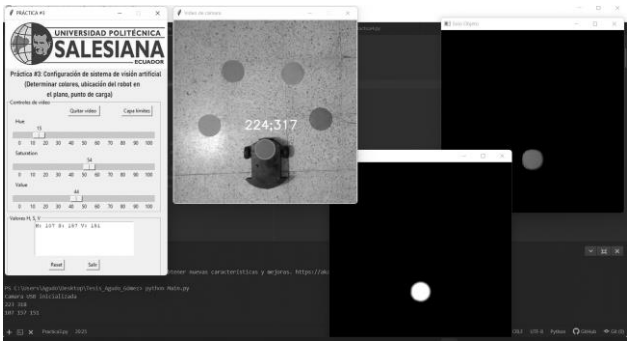


Figura 183. Valores HSV del color azul.

En la Figura 184 se observa la pantalla de video de cámara de la practica #3, junto con las ventanas de las máscaras de cada color en el plano, en este caso, se encuentra seleccionado el color amarillo.

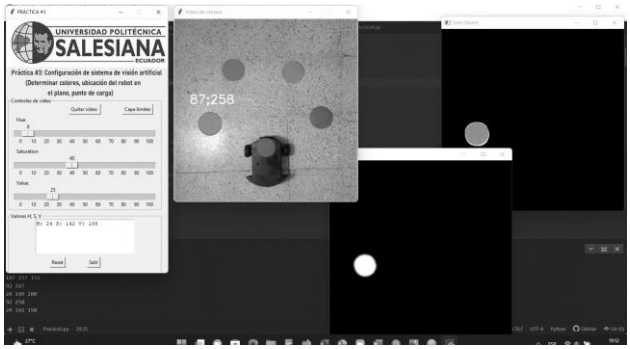


Figura 184. Valores HSV del color amarillo.

En la Figura 185 se observa la pantalla de video de cámara de la practica #3, junto con las ventanas de las máscaras de cada color en el plano, en este caso, se encuentra seleccionado el color verde.

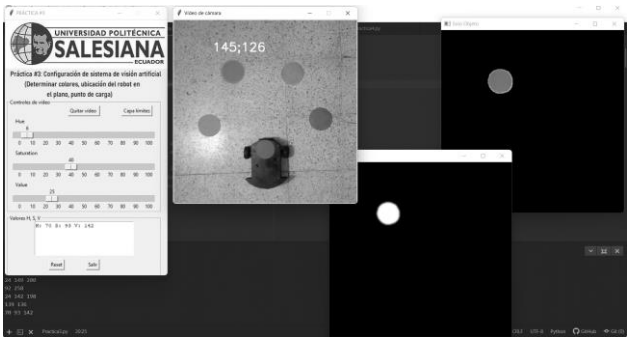


Figura 185. Valores HSV del color verde.

En la Figura 186 se observa la pantalla de video de cámara de la practica #3, junto con las ventanas de las máscaras de cada color en el plano, en este caso, se encuentra seleccionado el color naranja.

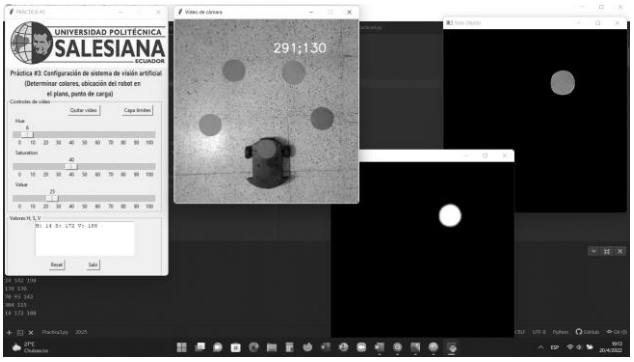


Figura 186. Valores HSV del color naranja.

En la Figura 187 se observa la pantalla de video de cámara de la practica #3, junto con las ventanas de las máscaras de cada color en el plano, en este caso, se encuentra seleccionado el color rojo.

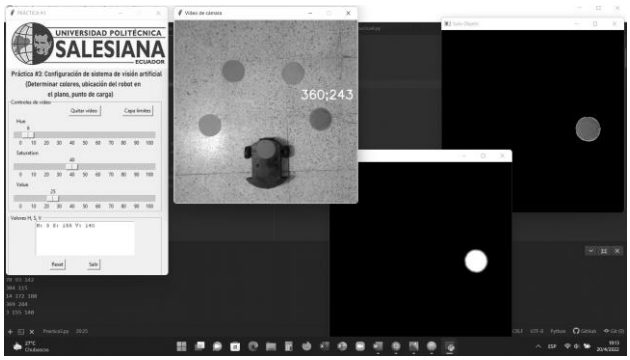


Figura 187. Valores HSV del color rojo.

4.4. Práctica #4.

En la interfaz de la práctica #4, se tiene parametrizado el color de la tarea 1, tal y como se aprecia en la Figura 188. Una vez realizado esto el robot se encuentra listo para dirigirse a este color. Hay que tomar en cuenta el factor luz a la hora de parametrizar cada tarea, ya que, de esta manera se evitan problemas en las trayectorias de estas mismas.

Figura 188. Color de la tarea 1 parametrizado.

En la interfaz de la práctica #4, se tiene parametrizado el color de la tarea 2, tal y como se aprecia en la Figura 189. Una vez realizado esto el robot se encuentra listo para dirigirse a este color. Hay que tomar en cuenta el factor luz a la hora de parametrizar cada tarea, ya que, de esta manera se evitan problemas en las trayectorias de estas mismas.

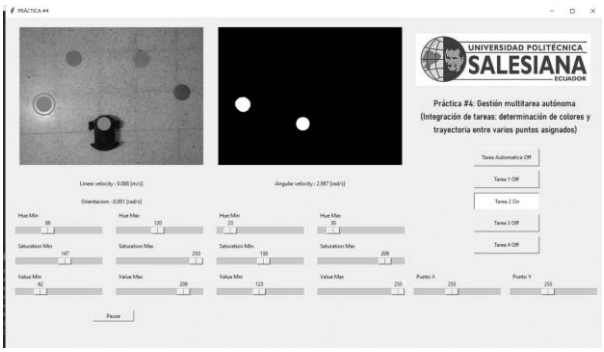


Figura 189. Color de la tarea 2 parametrizado.

En la interfaz de la práctica #4, se tiene parametrizado el color de la tarea 3, tal y como se aprecia en la Figura 190. Una vez realizado esto el robot se encuentra listo para dirigirse a este color. Hay que tomar en cuenta el factor luz a la hora de parametrizar cada tarea, ya que, de esta manera se evitan problemas en las trayectorias de estas mismas.

Figura 190. Color de la tarea 3 parametrizado.

En la tarea 4 se trabaja con un punto de referencia, es decir, teniendo un color seleccionado se puede proceder con la activación de la tarea 4, como se muestra a continuación en la Figura 191. La funcionalidad de esta tarea es que puede desplazarse hasta el punto que el usuario desee y puede ser tomada como base para diferentes aplicaciones.

Figura 191. Punto de seguimiento de la tarea 4.

En la Figura 192 se observa como el robot se desplaza hasta llegar al color que está configurado en la tarea 1.

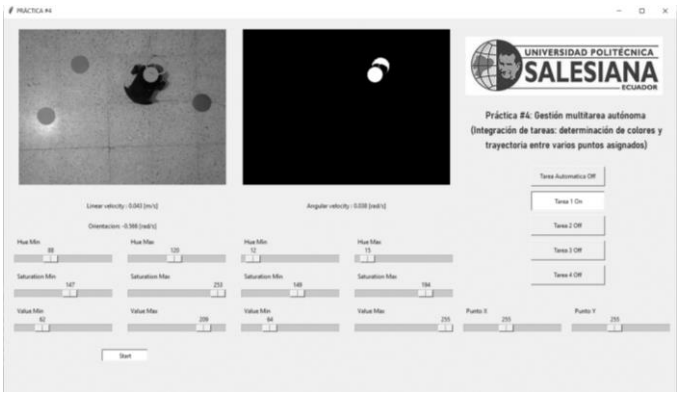


Figura 192. Desplazamiento del robot hacia el color de la tarea 1.

En la Figura 193 se muestra como el robot se desplaza hasta llegar al color que está configurado en la tarea 2.

Figura 193. Desplazamiento del robot hacia el color de la tarea 2.

En la Figura 194 se aprecia como el robot se desplaza hasta llegar al color que está configurado en la tarea 3.

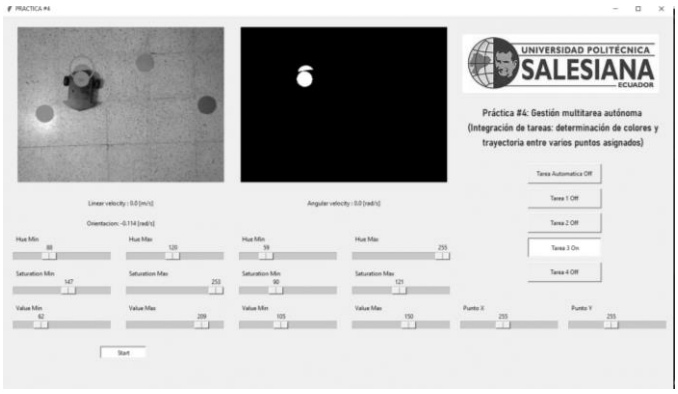


Figura 194. Desplazamiento del robot hacia el color de la tarea 3.

En la Figura 195 se puede ver como el robot se desplaza de color en color, ya que, una vez que llega a un color, pasa de manera automática al siguiente estado y así sucesivamente.

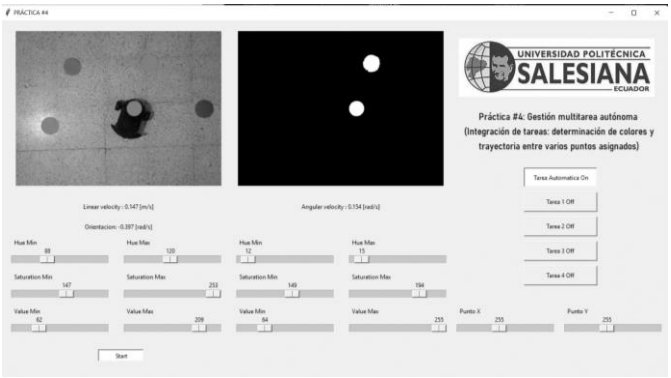


Figura 195. Desplazamiento del robot a los diferentes colores.

Se observa en la Figura 196, como el robot teniendo un color de referencia y activada la tarea 4, se dirige hacia la coordenada sleccionada.

Figura 196. Desplazamiento del robot hacia punto de referencia de la tarea 4.

4.5. Practica #5.

Se puede visualizar en la Figura 197 como se realiza el desplazamiento del robot hacia el color de la tarea 1 que ya se encuentra parametrizado. En esta práctica se eliminaron los sliders con la finalidad de tener más espacio para las ventanas de la cámara y para la ventana de la máscara. Incluye también un indicador de la batería real y simulada, el cambio entre estas se realiza mediante la tecla H, que muestra lo que está enviando el pin A0, en este caso la batería real y, la tecla G, que muestra lo que envía el pian A4, en este caso la batería simulada.

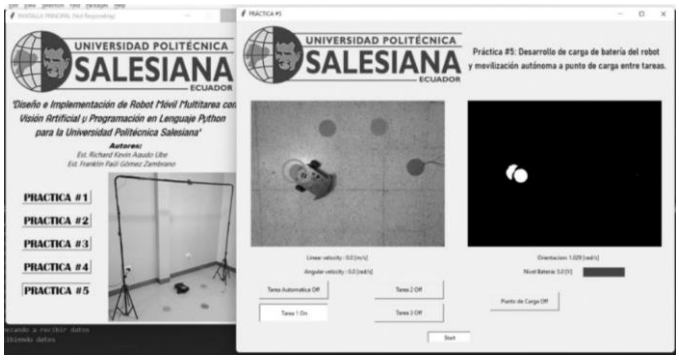


Figura 197. Interfaz de la práctica #5, desplazamiento de robot hacia la tarea 1.

Se observa en la Figura 198 como se realiza el desplazamiento del robot hacia el color de la tarea 2 que ya se encuentra parametrizado.

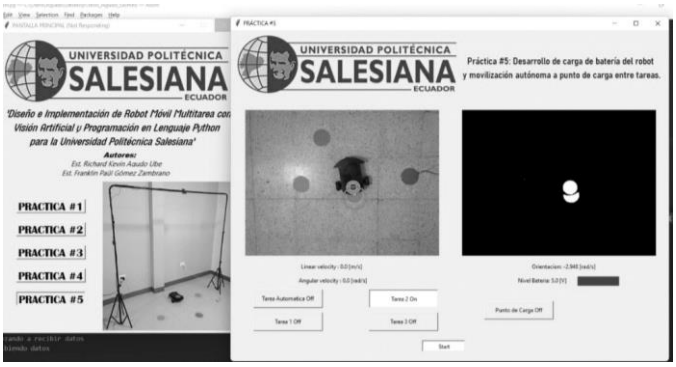


Figura 198. Interfaz de la práctica #5, desplazamiento de robot hacia la tarea 2.

Se observa en la Figura 199 como se realiza el desplazamiento del robot hacia el color de la tarea 3 que ya se encuentra parametrizado.

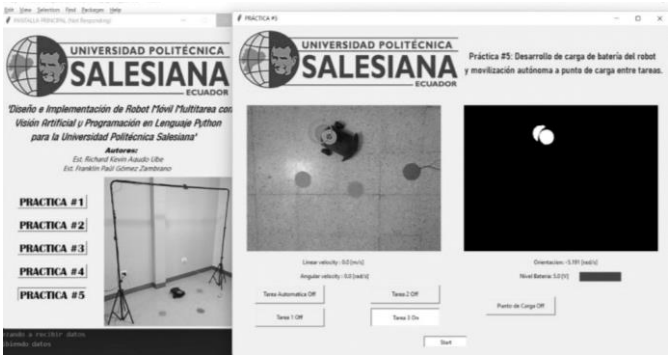


Figura 199. Interfaz de la práctica #5, desplazamiento de robot hacia la tarea 3.

En la Figura 200 se muestra como el robot al detectar que su nivel de batería es inferior a 2V, de manera autónoma interrumpe el ciclo de la tarea automática y se dirige hacia el punto de carga, en el cual, cuando ya se encuentre superior a 4V, vuelve a retomar la tarea que estaba realizando, como se puede apreciar en la Figura 201.

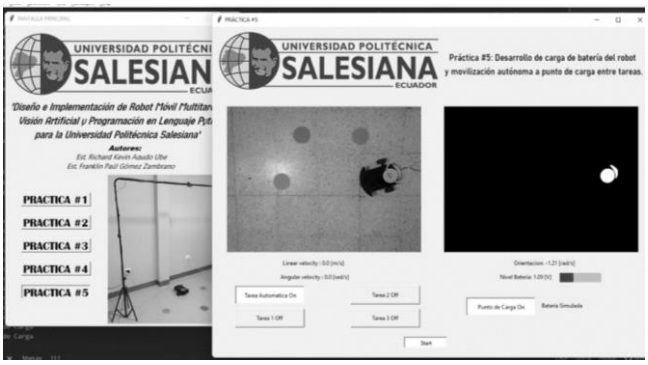


Figura 200. Robot situado en el punto de carga.

Figura 201. Robot cargado retomando la tarea automática.

5. ANALISIS DE RESULTADOS

5.1. Práctica #1.

En la Figura 202 se puede observar la interfaz de la práctica #1. Lo primordial en esta práctica fue poder realizar la configuración y manejo del robot inalámbricamente, es decir, verificar prueba de conexión y manejo a través de comandos en el computador. Se elaboró una interfaz con botones que permitieron dirigir el robot en la dirección que se deseaba, cada vez que una de las flechas era pulsada, ya sea mediante teclas o con el mouse, automáticamente se desactivaba el estado anterior y se dirigía a la trayectoria que se le establecía. Para poder controlar el robot a distancia se contó con un módulo bluetooth HC-05 y Python con sus respectivas librerías.

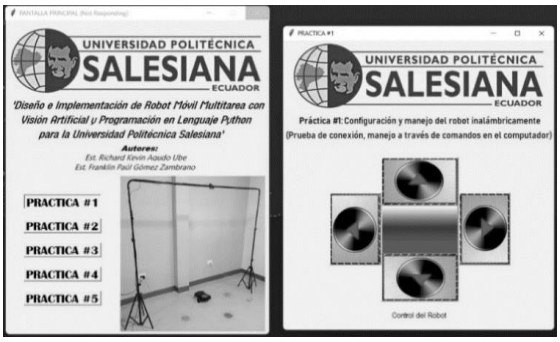


Figura 202. Controles direccionales del robot.

5.2. Práctica #2.

En la Figura 203 se observa la interfaz de la práctica #2, en la cual se realizó la verificación de comunicación bidireccional a través de un diodo led y un buzzer, mediante el pulso de teclas se comprobó si se activaban, esto indicó que existe comunicación entre ambas partes. Por otro lado, se tuvo la adquisición de datos a través de labels, en esta sección se logró observar la dirección en la que se encontraba nuestro robot, la orientación y el valor de una batería simulada colocada en un puerto analógico gracias a un potenciómetro, todo esto, con el fin de comprobar una correcta adquisición de datos. También se incluyó una ventana en la cual se pudo visualizar lo que la cámara estaba captando en tiempo real, tal y como se muestra en la Figura 204. A través de comandos del computador o con ayuda del cursor del mouse, se pudo seleccionar el botón en la dirección que se deseaba que vaya el robot, en este caso, se encontraba pulsada la flecha

que dirigía el robot hacia la izquierda y a su vez se mostraba el estado del mismo en cada uno de los labels según su movimiento.

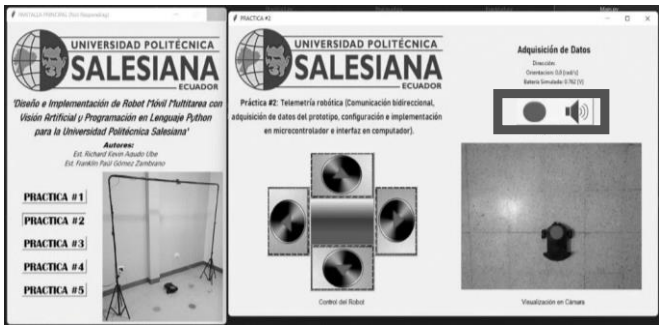


Figura 203. Activación de buzzer y diodo led.

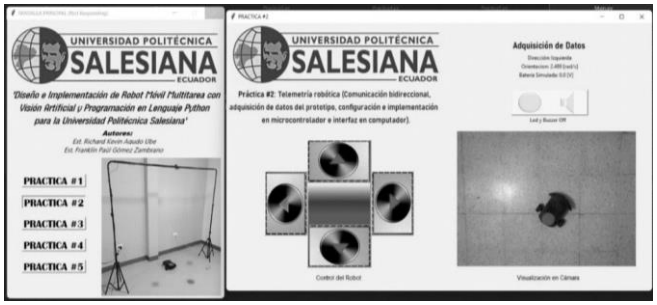


Figura 204. Estados del robot según su movimiento.

5.3. Práctica #3.

En la Figura 205 se muestra la pantalla principal con la práctica #3 seleccionada y a su derecha la interfaz de dicha práctica. Lo que se logró en esta práctica es poder configurar un sistema de visión artificial, lo que conllevaba la determinación de colores, ubicación del robot en el plano y también se incluyó un punto de carga. Una vez inicializada la ventana de la práctica, se procedió a mostrar las ventanas de las máscaras de cada color junto con la cámara, como se puede apreciar en la Figura 206. Entonces, cada vez que se seleccionaba un color, la interfaz mostraba sus valores HSV automáticamente, y a su vez, estaba diseñado para que permita ajustar a través de sliders la saturación, brillo y tono, eliminando de esta forma el ruido que se presentaba en cada color. Todo esto se

realizó, con la finalidad de poder identificar cada uno de los colores de manera correcta y poder usar estas bases para las siguientes prácticas.

Figura 205. Pantalla principal y ventana de la práctica #3.

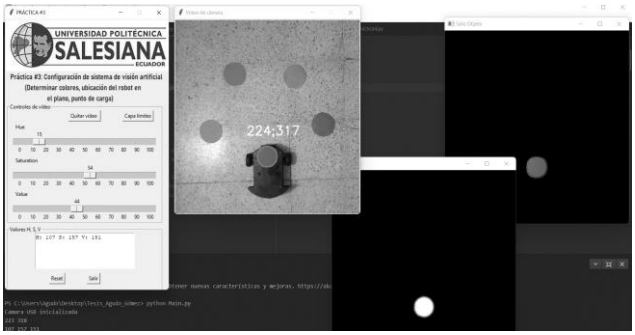


Figura 206. Ventana de la práctica #3 junto con las ventanas de máscara del color.

5.4. Práctica #4.

En la Figura 207 se observa la interfaz de la práctica #4, en la parte izquierda se tiene una ventana que mostraba lo que capta la cámara en tiempo real y en su parte inferior se encontraban unos sliders que servían para poder determinar el centroide del robot, es decir, la parte delantera que es de color azul. Una vez ya configurado se procedió a ir a la parte derecha, en donde se tuvo una ventana que permitió visualizar la máscara de cada uno de los colores que se muestran, para que, de esta manera en la parte inferior con ayuda de los sliders se puedan parametrizar los valores del factor HSV de manera correcta y poder realizar cada una de las tareas sin inconvenientes. Finalmente, en la parte inferior

derecha se ubicaron unos sliders, con los cuales se pudo desplazar nuestro robot en el plano X y Y.

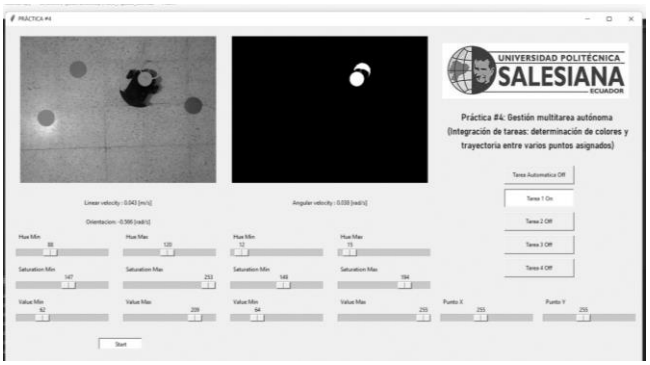


Figura 207. Interfaz de la práctica #4 con tarea 1 activada.

A continuación, en la Figura 208, se tiene colocados diferentes tipos de tareas y, para habilitar cada una de ellas colocamos un botón. Cada tarea realizó una trayectoria hasta llegar a un color específico, estos colores que se encontraron en cada tarea están parametrizados con sus respectivos valores de HSV mínimo y HSV máximo, cabe mencionar, que el factor luz fue muy importante a la hora de parametrizar los colores, ya que, podía presentar ruido, lo que afectaría al robot a la hora de llegar a cada tarea, por el motivo de que no se identificaría de manera correcta el color y no pasaría de un estado al siguiente.

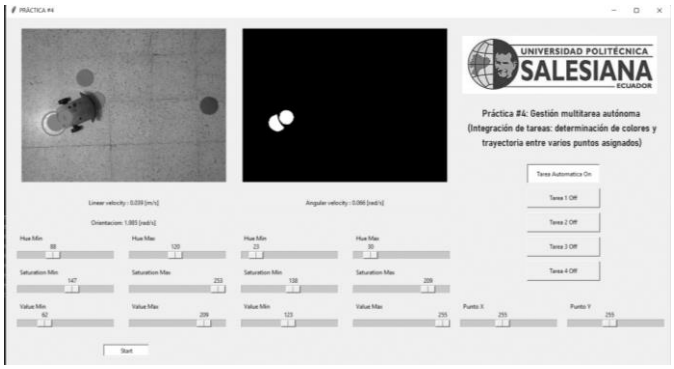


Figura 208. Interfaz de la práctica #4 con tarea automática activada.

En la Figura 209, se encuentra desarrollado un punto de seguimiento en la tarea 4, en donde, teniendo un color de referencia y activando dicha tarea, se mostraba como se establecía en la pantalla una circunferencia amarilla que representaba el punto a donde el robot debía dirigirse, todo esto se realizó, con el fin de poder crear una base para realizar una carga autónoma a través de identificación por color y recorrido a dicho punto de carga.

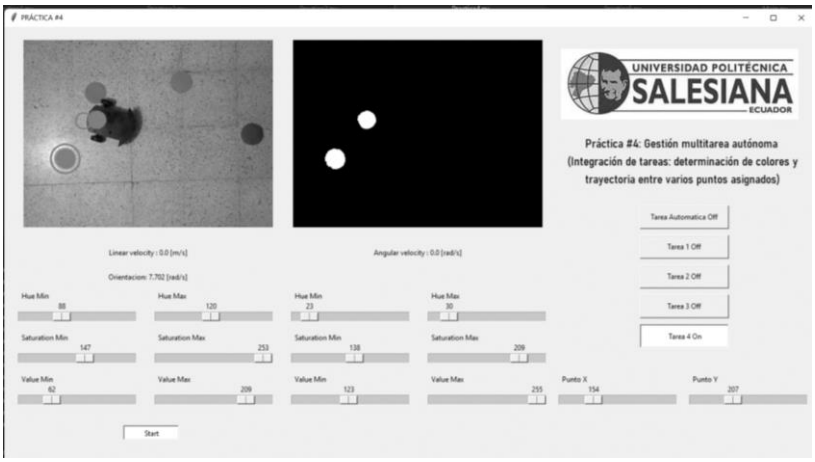


Figura 209. Interfaz de la práctica #4 con tarea 4 activada.

5.5. Practica #5.

En la Figura 210 se puede visualizar la práctica #5, en la cual, a diferencia de la interfaz anterior, ya no cuenta con los sliders para la parametrización. Esta práctica contó con 2 ventanas, una para visualizar la cámara en tiempo real y la otra ventana que mostró las máscaras de los colores que teníamos. La manera ideal para configurar los valores del factor HSV es ingresando estos mismos de forma directa gracias a las prácticas anteriores que facilitaron los valores específicos de cada color. Una vez, ya realizada esta configuración se obtuvo una interfaz menos cargada. Al momento de seleccionar cada tarea, el robot de manera automática se dirigió al color seleccionado.

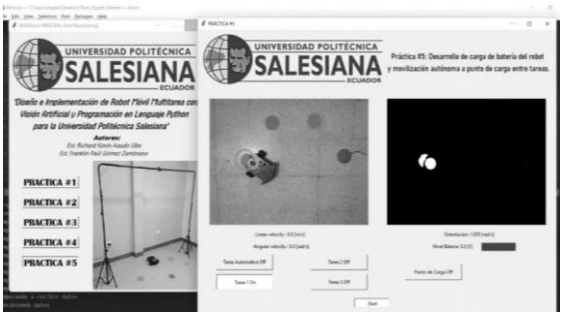


Figura 210. Práctica #5 con tarea 1 activada.

Luego, en la Figura 211, se observa el funcionamiento de la batería simulada, la cual, se configuró en el pin analógico A4 y con la ayuda de un divisor de voltaje se logró que el rango que se muestra en la entrada de ese pin, sea de 0 a 5V para no perjudicar dicho dato. Cuando el robot registraba un voltaje debajo de los 2V automáticamente interrumpía el ciclo de la tarea automática y se dirigía hacia el punto de carga hasta elevar su voltaje a más de 4V.

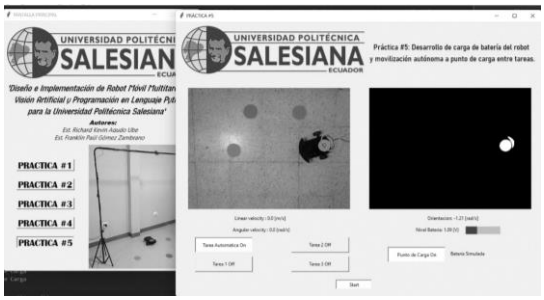


Figura 211. Práctica #5 con tarea automática y punto de carga activado con voltaje bajo.

Una vez que el voltaje fue mayor a 4V volvió retomar la tarea que tenía asignada. La carga del robot se dio por medio de una celda y a su vez, las ondas de carga fueron receptadas por medio de un módulo “Qi” Charging y junto con un módulo de carga alimentaba todo el circuito. Además, se utilizaron unos labels que mostraban la velocidad lineal que recorrió nuestro robot cuando se seleccionó una de las tareas, de la misma manera con la velocidad angular y ángulo de orientación, tal y como se muestra en la Figura 212.

Figura 212. Práctica #5 con tarea automática y punto de carga activado con voltaje alto.

6. CONCLUSIONES

- Para el control de los motores con caja reductora y encoder del robot móvil, se configuró en el IDE de Arduino las librerías necesarias para poder realizar el correcto control en lazo cerrado, haciendo que, de esta manera sea más fina la sintonía y poder enviar esos datos por medio de telemetría al IDE de Atom.

- En el IDE de Atom se deben importar las librerías correspondientes para proceder a configurar la detección de colores en el plano, para el desarrollo de la interfaz HMI de cada práctica, para la adquisición de datos bidireccionales entre el microcontrolador y Python.

- En la interfaz de prácticas en específico, se contó con labels, los cuales permiten visualizar la adquisición de datos de la orientación dado en radianes sobre segundos, la batería simulada a través de un pin análogo del microcontrolador dado en voltios.

- En las interfaces de gestión multitarea se puede visualizar los valores referentes a ángulos y velocidades que el robot brinda, además se visualiza el nivel de la batería, ya sea, simulada o a su vez su porcentaje real, todo esto lo realiza por medio de telemetría robótica.

- El estudio obtenido referente al motor se deduce a un control PID en lazo cerrado, se logró llegar a esto, realizando conexiones al motor con encoder, y de esta forma ir visualizando el comportamiento de cada una de las curvas que brinda la simulación.

- Se utilizó una aplicación para celular llamada IP Webcam, la cual brinda una dirección IP, con esto se puede colocar en la programación para poder enlazarla y obtener la imagen en tiempo real. Además, permitió configurar cambios en resoluciones y calidad de video de una manera sencilla.

- Mediante el uso de la cámara USB se logró obtener una mejor calidad de video y mejor apreciación de cada uno de los colores, para que de esta manera el robot no tenga perturbaciones al momento de ir a cada una de las tareas y al punto de carga.

- Se concluye que este prototipo es escalable, haciendo que de esta forma pueda utilizarse para desarrollo de nuevas tecnologías con visión artificial o IoT.

7. RECOMENDACIONES.

- Al momento de inicializar el robot, comprobar que las baterías estén con el voltaje requerido para su correcto funcionamiento.
- Verificar que los módulos step-up y step-down estén regulados correctamente para que se dispersen de manera correcta los voltajes requeridos a los motores con encoder y al microcontrolador.
- Revisar el estado de los motores, estos pueden generar movimientos erróneos si los encoders no funcionan de manera correcta.
- Verificar que las llantas estén adheridas con sus acoples de manera centrada con el fin de que el robot no presente desviaciones al momento de realizar las tareas.
- Ubicar en la parte delantera del robot el punto de referencia, ya que, si no se lo coloca en esa posición el robot no ejecuta las acciones planteadas de la manera correcta.
- Colocar la cámara USB de tal manera que quede centrada y pueda obtener una mayor visualización de cada uno de los colores.
- Al momento de compilar las prácticas, verificar que se encuentre colocado el puerto serial adecuado para que exista comunicación por medio del módulo bluetooth HC-05.
- Se debe considerar un lugar con muy buena iluminación, para que no existan perturbaciones al momento de configurar el modelo HSV de cada uno de los colores. Se debe de evitar brillos no favorables porque puede interferir con la adquisición de imagen que recibe la interfaz en Python.

8. REFERENCIAS BIBLIOGRÁFICAS.

Ortiz Ramirez, A. (2010). *Python como primer lenguaje de programación.* Monterrey.

Stack Overflow. (2018). *APRENDIZAJE atom-editor.* EBook Gratis.

3Dnatives. (2022). Obtenido de https://www.3dnatives.com/3D-compare/es/3d-printers/prusa-i3-mk2s-3/

AEROSEMI. (2022). *MT3608 Datasheet.*

Améstegui Moreno, M. (2001). *Control PID.* La Paz: Universidad Mayor de San Andres.

Arduino. (2022). *Arduino.cc*. Obtenido de https://docs.arduino.cc/hardware/nano

ARTIFICIAL, V. (s.f.). Obtenido de http://www.etitudela.com/celula/downloads/visionartificial.pdf

Atom. (s.f.). Obtenido de https://atom.io/

Bastero Huarte, N. (2018). *ChromeBook en el aula.* Cataluña: Universidad Pública de Navarra.

Biendicho Lletí, F. (2015). *"Comunicación Bluetooth entre Arduino UNO y Android aplicado a un detector de mentiras".* Gandia: UNIVERSIDAD POLITECNICA DE VALENCIA.

Brownlee, J. (2020). *Deep Learning for Computer Vision: Image Classification, Object Detection and face Recognition in Python.*

COGNEX. (2018). *INTRODUCCIÓN A LA VISIÓN ARTIFICIAL.*

Components info. (s.f.). Obtenido de https://www.componentsinfo.com/mt3608-module-pinout-datasheet/

Cornejo Ortega, A. D., & Tintin Suquilanda, J. L. (2010). *Diseño, construcción e implementación de un sistema de telemetría utilizando tecnología GSM;para el monitoreo de los parámetros de temperatura, presión de aceite,velocidad de giro del motor y velocidad de desplazamiento de un vehículo chevrolet optra 2008.* Cuenca: Universidad Politécnica Salesiana Sede Cuenca.

Déleg, M. (2010). *TECNOLOGÍA LED.* Cuenca: Universidad Politécnica Salesiana.

EasyEDA. (2021). Obtenido de https://easyeda.com/

Ebrahim, M. (22 de enero de 2018). Obtenido de https://likegeeks.com/es/ejemplos-de-la-gui-de-python/

Ferrer, V. (Enero de 2020). *Pilas Recargables*. Obtenido de https://vicentferrer.com/pilas-recargables/

Gil, I. (2015). *La impresión 3D y sus alcances en la arquitectura.* Madrid. Obtenido de http://oa.upm.es/38442/7/PFC_IRENE_GIL_GIL.pdf

González Marcos, A., Martínez de Pisón Ascacíbar, F. J., Pernía Espinoza, A. V., Alba Elías, F., Castejón Limas, M., Ordieres Meré, J., & Vergara González, E. (2006). *Técnicas y Algoritmos Básicos de Visión Artifical.* Logroño: Universidad de la Rioja.

Grassvalley. (2012). *http://grassvalley.com.* Obtenido de CMOS: Listos para el broadcast de hoy: https://www.grassvalley.com/docs/WhitePapers/broadcast/cameras/ldk3000plus/CAM-4073M-ES_CMOS_Whitepaper.pdf

J. Charras, D. H. (13 de julio de 2020). *KiCAD*. Obtenido de https://kicad-pcb.org/: https://kicad-pcb.org/about/kicad/

Kurniawan, A. (2019). *Arduino Nano A Hands-On Guide for Beginner.* Tasikmalaya.

La enciclopedia de la ingeniería. (15 de Octubre de 2018). *Ingeniería Mecafenix*. Obtenido de https://www.ingmecafenix.com/electronica/el-buzzer/

Lambán, P., & Domínguez, A. (2013). *https://impresoras3dblog.wordpress.com/*. Obtenido de Impresoras 3D Blog: https://impresoras3dblog.wordpress.com/

Liechti, C. (2018). *pySerial Documentation Release 3.4.*

Mosquera De la Cruz, J., Ferrin Bolaños, C., Santander Ariza, D., Rosero Ramos, K., Libreros Segura, M., & Loaiza Correa, H. (2020). Sistema de reconocimiento de voz para controlar la aplicación WhatsApp orientado a personas con limitaciones motrices. *Lumen Gentium*, 8-9.

Naylamp. (2021). *Naylamp Mechatronics*. Obtenido de https://naylampmechatronics.com/motores-dc/616-motor-dc-con-caja-reductora-y-encoder-25ga-12v-350rpm.html

Naylamp Mechatronics . (2021). Obtenido de https://naylampmechatronics.com/baterias/194-cargador-de-bateria-litio-tp4056-micro-usb.html

Naylamp Mechatronics. (2021). Obtenido de https://naylampmechatronics.com/conversores-dc-dc/196-convertidor-voltaje-dc-dc-step-down-3a-lm2596.html#:~:text=El%20convertidor%20DC%2DDC%20LM2596,dise%C3%B1o%20de%20fuentes%20de%20alimentaci%C3%B3n.

Nomada. (2016). *Especificaciones Técnicas del Driver de baja Potencia para Motores [L293D].* Guadalupe.

OpenCV team. (s.f.). *OpenCV*. Recuperado el 25 de 08 de 2020, de https://opencv.org/

OpenCV team. (s.f.). *OpenCV*. Obtenido de https://opencv.org/

Pinto Salamanca, M. L., Barrera Lombana, N., & Pérez Holguín, W. J. (2010). *USO DE LA ROBÓTICA EDUCATIVA.* Universidad Pedagógica y Tecnológica de Colombia.

Pruna, E., Edison, S., & Mullo , S. (2017). *PI and PID Controller Tuning Tool Based on the Lambda Method.* Sangolqui: IEEE.

Python Software Foundation. (s.f.). Obtenido de https://www.python.org/community/logos/

Python Software Foundation. (2021). *Biblioteca Estándar de Python.* Obtenido de https://docs.python.org/3/library/tkinter.html

Python Software Foundation. (15 de Enero de 2021). *Python Package Index.* Obtenido de https://pypi.org/project/imutils/

S. Carballas, J. C. (2018). *Creación de un protocolo de comunicación entre un PC y un puerto USB de un microcontrolador y su integración en MATLAB.* Coruña, España. Obtenido de https://ruc.udc.es/dspace/bitstream/handle/2183/21183/CarballasChas_Sergio_TFG_2018.pdf?sequence=2&isAllowed=y

Salvador, J. (11 de Diciembre de 2019). *Stanser*. Obtenido de https://www.stanser.com/materiales-que-puede-cortar-cada-tipo-de-cnc/

Sanson, & Dalila. (2022). *Autodesk Inventor*. Obtenido de Autodesk Inventor: https://www.asidek.es/industria-y-fabricacion-2/autodesk-inventor/#:~:text=Autodesk%20Inventor%3A%20el%20software%20profesional,documentaci%C3%B3n%20de%20productos%20en%203D.

Sculpteo. (2020). *https://www.sculpteo.com/es/*. Obtenido de https://www.sculpteo.com/es/glosario/corte-por-laser-definicion/

Solís Cascante, F. R. (2018). *DISEÑO Y CONTRUCCION DE UN PACK DE BATERIAS DE LITIO.* Universidad Internacional SEK.

STMicroelectronics. (1996). *PUSH-PULL FOUR CHANNEL DRIVER WITH DIODES.* Italia.

Texas instruments. (1999). *LM2596 SIMPLE SWITCHER Power Converter 150-kHz.* Texas.

TROTEC. (2020). *https://www.troteclaser.com/es*. Obtenido de https://www.troteclaser.com/es-ec/faqs/como-cortar-con-laser/

V. M. Arévalo, J. G. (2004). *LA LIBRERÍA DE VISIÓN ARTIFICIAL OPENCV.* Málaga.

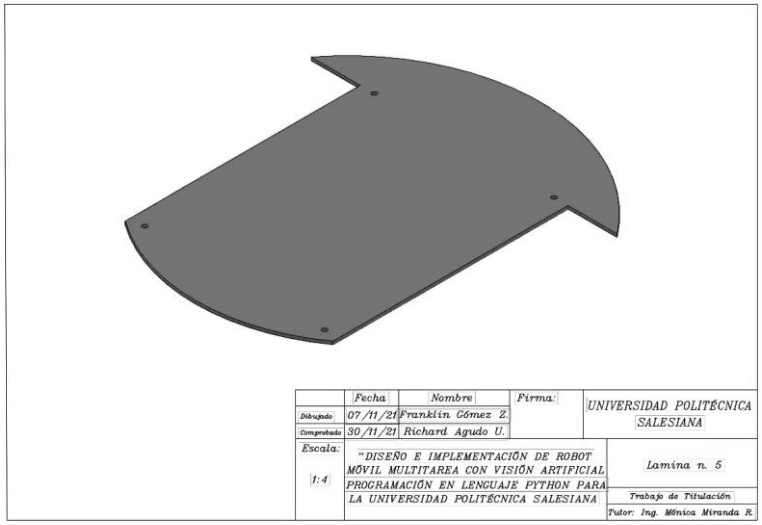
van der Walt, S., Colbert, C., & Varoquaux, G. (2011). *The NumPy Array: A Structure for Efficient Numerical Computation.*

9. ANEXOS

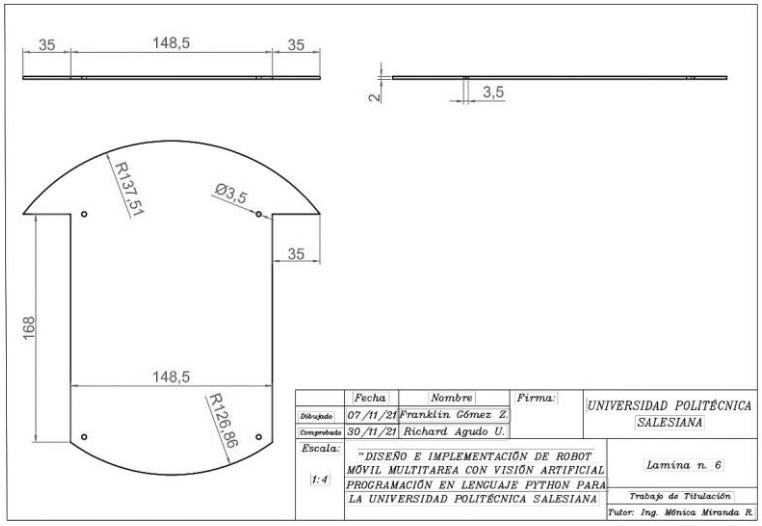
ANEXO A. CRONOGRAMA DE DURACIÓN DEL PROYECTO**.**

ACTIVIDADES		MESES																	
		1		2		3		4		5		6		7		8		9		10	
1	INVESTIGACIÓN Y COMPRA DE MATERIALES																				
2	ARMADO DE ESTRUCTURA Y DEL PROTOTIPO																				
3	PROGRAMACIÓN, ALGORITMOS DE VISIÓN ARTIFICIAL Y PROTOTIPO																				
4	PRUEBAS Y CORRECIÓN DE FALLAS																				
5	DESARROLLO DE PRÁCTICAS																				
6	REALIZACIÓN DE LA DOCUMENTACIÓN																				
7	PRUEBA Y ERROR DEL FUNCIONAMIENTO DEL PROYECTO																				
8	REVISIÓN DE LA DOCUMENTACIÓN																				
9	PRESENTACIÓN DE PROYECTO																				

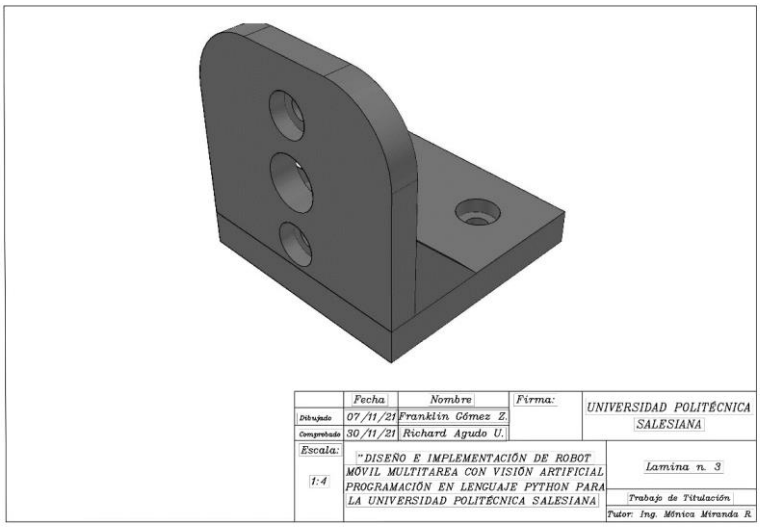
ANEXO B. DISEÑO DE BASE DEL PROTOTIPO – VISTA 3D.



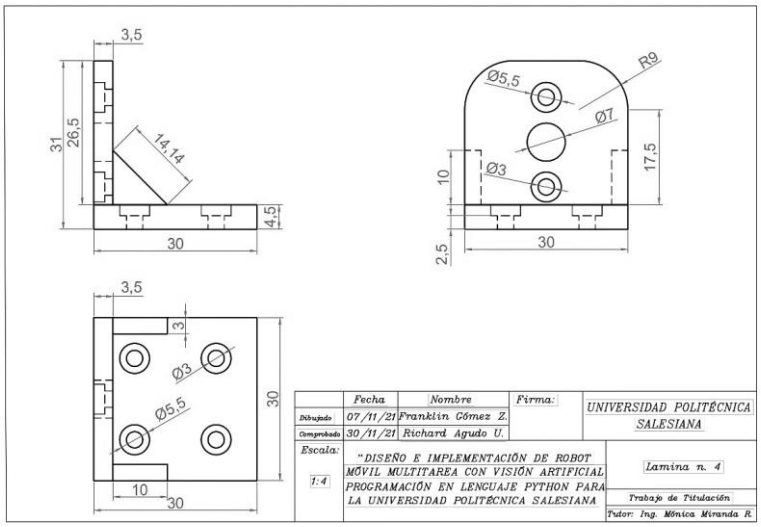
ANEXO C. DISEÑO DE BASE DEL PROTOTIPO – VISTAS.



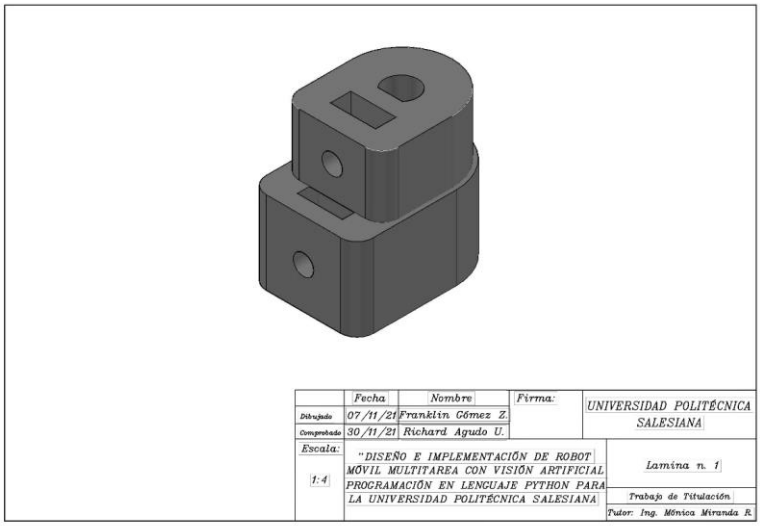
ANEXO D. DISEÑO DE SOPORTE DEL MOTOR – VISTA 3D.



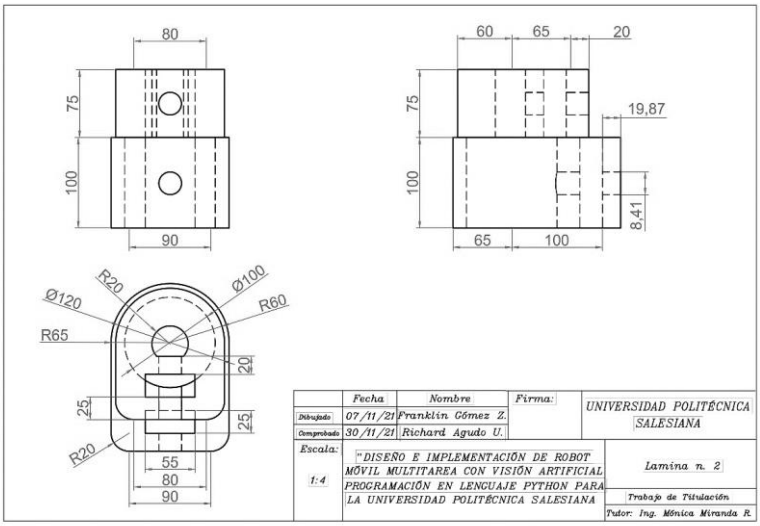
ANEXO E. DISEÑO DE SOPORTE DEL MOTOR – VISTAS.



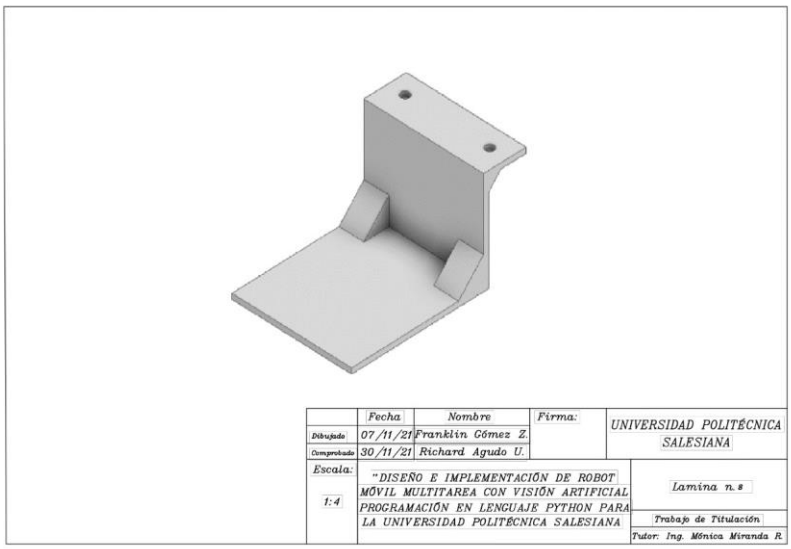
ANEXO F. DISEÑO DE ACOPLE DEL MOTOR – VISTA 3D.



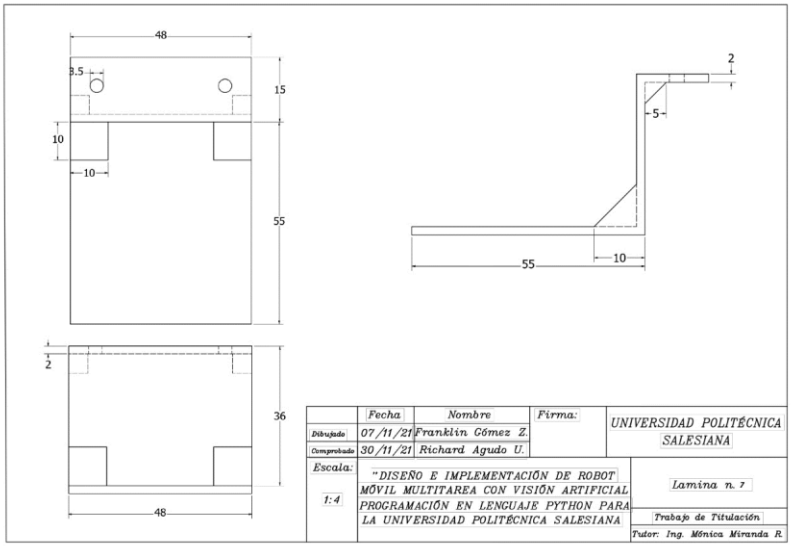
ANEXO G. DISEÑO DE ACOPLE DEL MOTOR – VISTAS.



ANEXO H. DISEÑO DE SOPORTE DE CELDA – VISTA 3D.



ANEXO I. DISEÑO DE SOPORTE DE CELDA – VISTAS.



ANEXO J. PROGRAMACIÓN EN ARDUINO.

```
#include "PinChangeInterrupt.h" //Librería para poder utilizar todos los pines de arduino como interrupciones
#include "motorControl.h" //Librería para controlar en lazo cerrado los motores
#include "LowPower.h"
////////////////////////// CONTROLADOR PID ////////////////////

unsigned long lastTime, sampleTime = 100; //Variable sin signo para utilizar en tiempo de muestreo y declaro variable de tiempo de muestreo

motorControl motor1(sampleTime);  //Creamos un objeto y enviamos el tiempo de muestreo al motor 1
motorControl motor2(sampleTime);  //Creamos un objeto y enviamos el tiempo de muestreo al motor 2

////////////////////// COMUNICACION SERIAL /////////////////

String inputString = "";  //Variable donde almacena la cadena que empieza con valor vacío
bool stringComplete = false;  //Bandera para verificar si los datos llegaron completos
const char separator = ',';
const int dataLength = 2; //Recibe 2 datos: la velocidad lineal de referencia y la velocidad angular de referencia
double data[dataLength];  //Recibe el valor que regula el ciclo de trabajo (PWM) con esto regulamos la velocidad del motor o sea el Sp

//////////////////////MOTOR DERECHO////////////////////////////////
//// Ojo se ha invertido canales////////

const int    C1R = 2;    // Entrada de la señal A del encoder.
const int    C2R = 3;    // Entrada de la señal B del encoder.
int outValueR = 0;       //Variable de control, por defecto empieza en cero

//// Puente H L293D ////
const int    in1 = 7;    //Pin para el sentido de giro del motor (derecho)
const int    in2 = 8;    //Pin para el sentido de giro del motor (derecho)
const int    enA = 6;    //Pin de activación del enable de la señal A en el puente H

//////////////Variables para lectura de encoder////////////

volatile int countR = 0; //Numero de cuentas que va a tener el encoder
volatile int antR = 0;  //Estado anterior
volatile int actR = 0;  //Estado actual
double wR = 0;  //Variable proceso (Velocidad angular en rad/s)
double w1Ref = 0;  //Setpoint (Velocidad angular de referencia en rad/s)

//////////////////////MOTOR IZQUIERDO////////////////////////////////

const int    C1L = 5;    // Entrada de la señal A del encoder.
const int    C2L = 4;    // Entrada de la señal B del encoder.
int outValueL = 0;      //Variable de control, por defecto empieza en cero
```

```
//// Puente H L293D ////
const int    in3 = 9;    //Pin para el sentido de giro del motor (izquierdo)
const int    in4 = 10;   //Pin para el sentido de giro del motor (izquierdo)
const int    enB = 11;   //Pin de activación del enable de la señal A en el puente H

//////////////Variables para lectura de encoder////////////

volatile int countL = 0;  //Numero de cuentas que va a tener el encoder
volatile int antL = 0;  //Estado anterior
volatile int actL = 0;  //Estado actual
double wL = 0;  //Variable proceso (Velocidad angular en rad/s)
double w2Ref = 0;  //Setpoint (Velocidad angular de referencia en rad/s)


//////// VARIABLES PARA CALCULAR VELOCIDADES ANGULARES /////////

double constValue = 3.1733; // (1000*2*pi)/R ---> R = 1980 Resolution encoder quadruple

////////////////////////// ROBOT //////////////////////////
double uRobot  = 0;  //Velocidad lineal mt/s
double wRobot  = 0;  //Velocidad angular mt/s
double phi = 0;  //Ángulo de orientación
const double R = 0.0335; // Radio de la llanta
const double d = 0.205; // Distancia entre llantas

////////////////////////// BATTERY //////////////////////////
int batteryPin = A4;   //Pin análogo de señal de la batería simulada
const int buzzer = 12;       //Pin digital de señal del buzzer (zumbador)
const int LED = 13;       //LED
void setup()
{
  Serial.begin(9600);  //Iniciar el puerto serial a 9600 baudios
  double voltage = 0.0;  //Declaro la batería como dato double
  voltage  = analogRead(batteryPin)*(5.0/1023.0);  //Obtener el voltaje simulado en pin A4

  /*////////////////// SINTONIA LAMBDA PID ///////////////////

    motor.lambdaTunning(1.9281,0.1831,0.1337);  //(K,tau,delay)
    Serial.print()motor.getKc());
    Serial.print(", ");
    Serial.print()motor.getTi());
    Serial.print(", ");
    Serial.print()motor.getTd());
    Serial.print(", "); */

  ////////////////// SINTONIA FINA MOTOR 1 //////////////////
  motor1.setGains(0.21, 0.07, 0.05); // (Kc,Ti,Td)

  ////////////////// Limites de señales ///////////////////
  motor1.setCvLimits(255, 20); //Limitamos señal de control de 20 a 255 (Considerar zona
muerta)
  motor1.setPvLimits(11, 0); //Velocidad angular max 11 rad/s y min 0 rad/s
```

```
  ////////////////// SINTONIA FINA MOTOR 2 //////////////////
  motor2.setGains(0.21, 0.07, 0.05); // (Kc,Ti,Td)

  ////////////////// Limites de señales //////////////////
  motor2.setCvLimits(255, 20); //Limitamos señal de control de 20 a 255 (Considerar zona muerta)
  motor2.setPvLimits(11, 0); //Velocidad angular max 11 rad/s y min 0 rad/s

  ////////////////// DECLARACIÓN DE PINES E/S //////////////////

  pinMode(C1R, INPUT);  //Declaro como entrada la señal A del encoder derecho
  pinMode(C2R, INPUT);  //Declaro como entrada la señal B del encoder derecho
  pinMode(C1L, INPUT);  //Declaro como entrada la señal A del encoder izquierdo
  pinMode(C2L, INPUT);  //Declaro como entrada la señal B del encoder izquierdo

  pinMode(in1, OUTPUT);  //Declaro como salida el pin del sentido de giro del motor derecho
  pinMode(in2, OUTPUT);  //Declaro como salida el pin del sentido de giro del motor derecho
  pinMode(in3, OUTPUT);  //Declaro como salida el pin del sentido de giro del motor izquierdo
  pinMode(in4, OUTPUT);  //Declaro como salida el pin del sentido de giro del motor izquierdo

  pinMode(enA, OUTPUT);  //Declaro como salida el pin del sentido de giro del motor izquierdo
  pinMode(enB, OUTPUT);  //Declaro como salida el pin del sentido de giro del motor izquierdo


  digitalWrite(in1, false);  //Inicializo IN1 en falso
  digitalWrite(in2, false);  //Inicializo IN2 en falso
  digitalWrite(in3, false);  //Inicializo IN3 en falso
  digitalWrite(in4, false);  //Inicializo IN4 en falso

  ////////////////// PINES DE INTERRUPCION //////////////////

  attachInterrupt(digitalPinToInterrupt(C1R), encoderR, CHANGE); //Declaro C1R como interrupción (encoder derecho)
  attachInterrupt(digitalPinToInterrupt(C2R), encoderR, CHANGE); //Declaro C2R como interrupción (encoder derecho)

  attachPinChangeInterrupt(digitalPinToPinChangeInterrupt(C1L), encoderL, CHANGE); //Declaro C1L como interrupción (encoder izquierdo)
  attachPinChangeInterrupt(digitalPinToPinChangeInterrupt(C2L), encoderL, CHANGE); //Declaro C2L como interrupción (encoder izquierdo)

  ////////////////////////// BUZZER //////////////////////////

  pinMode(buzzer, OUTPUT); //Declaro como salida el pin donde se conecta el buzzer
  digitalWrite(buzzer, LOW); //Incializo el buzzer en bajo

  lastTime = millis();  //Actualiza el tiempo anterior
}

void loop() {

  ////////// SI RECIBE DATOS //////////
  if (stringComplete)
```

```
 {
   for (int i = 0; i < dataLength ; i++)
   {
     int index = inputString.indexOf(separator);
     data[i] = inputString.substring(0, index).toFloat();
     inputString = inputString.substring(index + 1);
   }

   velocityMotor(data[0], data[1]);


   inputString = "";
   stringComplete = false;
 }

 ////////////////// CONTROLADOR PID ////////////////

 if (millis() - lastTime >= sampleTime) //Compara el tiempo actual con el tiempo anterior
 { //Y si es >= al tiempo de muestreo hacemos lo sgte
   wR = constValue * countR / (millis() - lastTime); //Calculamos velocidad angular rad/s motor
der.
   wL = constValue * countL / (millis() - lastTime); //Calculamos velocidad angular rad/d motor
izq.

   lastTime = millis();  //Almacenamos el tiempo actual
   countR = 0;  //Reiniciamos los pulsos motor der.
   countL = 0; //Reiniciamos los pulsos motor izq.

   velocityRobot(wR, wL); //Velocidades angulares del robot

   phi = phi + wRobot * 0.1; //Aquí obtenemos la orientación del robot en radianes
   Serial.println(phi, 3);


   battery();

   outValueR = motor1.compute(w1Ref, wR); //Devuelve la variable de control a motor der.
   outValueL = motor2.compute(w2Ref, wL); //Devuelve la variable de control a motor izq.

   if (outValueR > 0) clockwise(in2, in1, enA, outValueR); else anticlockwise(in2, in1, enA,
abs(outValueR)); //Si es >0 motor der. gira sentido H. sino sentido A.H.
   if (outValueL > 0) anticlockwise(in3, in4, enB, outValueL); else clockwise(in3, in4, enB,
abs(outValueL)); //Si es >0 motor izq. gira sentido A.H. sino sentido H.
 }                                                                    //"abs" debido a que arduino
admite de 0 a 255


}

/////////////// RECEPCION DE DATOS ////////////////////
void serialEvent()
{
 while (Serial.available()) //Mientras exista datos
```

```
  {
    char inChar = (char)Serial.read(); //Leer cada uno de los caracteres
    inputString += inChar;  //Concatenar cada uno de los valores en el string
    if (inChar == '\n')  //Comparala bandera, si este es igual a un salto de linea
    {
      stringComplete = true;  //Levantar la bandera, recibe datos
    }
    //////////////LED Y BUZZER///////////////
    if (inChar == 'F')
    {
      digitalWrite(13, HIGH);
      digitalWrite(12, HIGH);
      velocityMotor(2, 0);
    }
    if (inChar == 'B')
    {

      digitalWrite(13, LOW);
      digitalWrite(12, LOW);
      velocityMotor(0, 0);
    }
    //////////////BATERÍA REAL Y BATERÍA SIMULADA///////////////
    if (inChar == 'G')
    {
      batteryPin = A4;
    }
    if (inChar == 'H')
    {
      batteryPin = A0;

    }
  }
}

void encoderR(void) //Función del encoder der. para pecisión cuádruple
{
  antR = actR;  //Actualizamos los estados
  actR = PIND & 12; //Filtramos los pines del puerto D (2 y 3)

  if (antR == 0  && actR == 4)  countR++;
  if (antR == 4  && actR == 12)  countR++;
  if (antR == 8  && actR == 0)  countR++;
  if (antR == 12 && actR == 8)  countR++;

  if (antR == 0 && actR == 8)  countR--;
  if (antR == 4 && actR == 0)  countR--;
  if (antR == 8 && actR == 12) countR--;
  if (antR == 12 && actR == 4) countR--;

}
void encoderL(void)  //Función del encoder izq. para pecisión cuádruple
{
  antL = actL;  //Actualizamos los estados
```

```
  actL = PIND & 48; //Filtramos los pines del puerto D (4 y 5)

  if (antL == 0  && actL == 16)  countL++;
  if (antL == 16 && actL == 48)  countL++;
  if (antL == 32 && actL == 0)  countL++;
  if (antL == 48 && actL == 32)  countL++;

  if (antL == 0  && actL == 32)  countL--;
  if (antL == 16 && actL == 0)  countL--;
  if (antL == 32 && actL == 48)  countL--;
  if (antL == 48 && actL == 16)  countL--;

}

void clockwise(int pin1, int pin2, int analogPin, int pwm)
{
  digitalWrite(pin1, LOW);
  digitalWrite(pin2, HIGH);
  analogWrite(analogPin, pwm);
}

void anticlockwise(int pin1, int pin2, int analogPin, int pwm)
{
  digitalWrite(pin1, HIGH);
  digitalWrite(pin2, LOW);
  analogWrite(analogPin, pwm);
}

void velocityRobot(double w1, double w2)
{
  uRobot = (R * (w1 + w2)) / 2;
  wRobot = (R * (w1 - w2)) / d;
}
void velocityMotor(double u, double w)
{
  w1Ref = (u + (d * w / 2)) / R; //Velocidad llanta derecha
  w2Ref = (u - (d * w / 2)) / R; //Velocidad llanta izquierda
}
void battery()  //Función batería
{
  double voltage = 0.0;  //Declaro la batería como dato double
  voltage  = analogRead(batteryPin)*(5.0/1023.0);  //Obtener el voltaje en pin A0
  Serial.println(voltage, 3);
}
```

ANEXO K. PROGRAMACIÓN PRINCIPAL EN PYTHON – MAIN.

```
from tkinter import *
import cv2
from PIL import Image, ImageTk
import tkinter as tk
import tkinter.messagebox
```

```
from tkinter import filedialog
import os

path = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/logoUPS.png'
path1 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/robot4.jpeg'

def onClossing():
    root.quit()          #Salir del bucle de eventos
    root.destroy()       #Destruye la ventana creada
root = Tk()
root.protocol("WM_DELETE_WINDOW",onClossing)
root.title("PANTALLA PRINCIPAL")
root.geometry("510x650")

lbltext=Label(text="'Diseño e Implementación de Robot Móvil Multitarea con",font=("Siemens
AD Sans",17,"italic")).place(x=8,y=160)
lbltext1=Label(text="Visión Artificial y Programación en Lenguaje Python",font=("Siemens AD
Sans",17,"italic")).place(x=25,y=190)
lbltext1=Label(text="para la Universidad Politécnica Salesiana'",font=("Siemens AD
Sans",17,"italic")).place(x=60,y=220)

lblautores=Label(text="Autores:",font=("Arial black",11, "italic")).place(x=220,y=250)
lblagudo=Label(text="Est. Richard Kevin Agudo
Ube",font=("Ebrima",11,"italic")).place(x=150,y=270)
lblgomez=Label(text="Est. Franklin Paúl Gómez
Zambrano",font=("Ebrima",11,"italic")).place(x=130,y=290)

img = ImageTk.PhotoImage(Image.open(path))
panel = tk.Label(root, image = img)
panel.pack(side = "top", fill = "both", expand = "yes")
panel.place(x=0, y=10)

img1 = ImageTk.PhotoImage(Image.open(path1))
panel1 = tk.Label(root, image = img1)
panel1.pack(side = "top", fill = "both", expand = "yes")
panel1.place(x=220, y=320)

############################Funciones para llamar
practicas#######################
def p1():
    pr1 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/Practica1.py'
    os.system('"%s"' % pr1)

def p2():
    pr2 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/Practica2.py'
    os.system('"%s"' % pr2)

def p3():
    pr3 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/Practica3.py'
    os.system('"%s"' % pr3)

def p4():
    pr4 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/Practica4.py'
```

```
    os.system('"%s"' % pr4)

def p5():
    pr5 = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/Practica5.py'
    os.system('"%s"' % pr5)

###################Botones de practicas#########################################
boton1=Button(root,text='PRACTICA                #1',font=("Britannic                Bold",18),
command=p1).place(x=35, y=360, width=150, height=30)
boton2=Button(root,text='PRACTICA                #2',font=("Britannic                Bold",18),
command=p2).place(x=35, y=410, width=150, height=30)
boton3=Button(root,text='PRACTICA                #3',font=("Britannic                Bold",18),
command=p3).place(x=35, y=460, width=150, height=30)
boton4=Button(root,text='PRACTICA                #4',font=("Britannic                Bold",18),
command=p4).place(x=35, y=510, width=150, height=30)
boton5=Button(root,text='PRACTICA                #5',font=("Britannic                Bold",18),
command=p5).place(x=35, y=560, width=150, height=30)

root.mainloop()
```

ANEXO L. PROGRAMACIÓN EN PYTHON – PRÁCTICA #1.

```
from pyArduino import *
from tkinter import *
from PIL import Image, ImageTk
import sys
import cv2
import numpy as np
import imutils
import tkinter as tk
import tkinter.messagebox

def onClossing():
    arduino.sendData([0,0])
    arduino.close()
    root.quit()        #Salir del bucle de eventos.
    root.destroy()     #Destruye la ventana creada

path = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/logoUPS.png'
path_flecha_abajo_1                                                    =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/abajo_off.jpeg'
path_flecha_abajo_2                                                    =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/abajo_on.jpeg'
path_flecha_arriba_1                                                   =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/arriba_off.jpeg'
path_flecha_arriba_2                                                   =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/arriba_on.jpeg'
path_flecha_derecha_1                                                  =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/derecha_off.jpeg'
path_flecha_derecha_2                                                  =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/derecha_on.jpeg'
```

```
path_flecha_izq_1 =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/izquierda_off.jpeg'
path_flecha_izq_2 =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/izquierda_on.jpeg'
path_boton_gris = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/paro_off.png'
path_boton_azul = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/paro_on.png'


def adelante(w):
    arduino.sendData([0.3,0.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_2)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    print("adelante")

def derecha(d):
    arduino.sendData([0.0,-2.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_2)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    print("derecha")

def izquierda(a):
    arduino.sendData([0.0,2.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_2)
    btn_boton_parar.config( image=img_path_Boton_gris)
    print("izquierda")

def parar(s):
    arduino.sendData([0.0,0.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_azul)
    print("parar")

def atras(x):
    arduino.sendData([-0.3,0.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_2)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    print("atras")
```

```
def toggle_flecha_abajo():
    btn_flecha_abajo.config(image=img_path_flecha_abajo_2)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    arduino.sendData([-0.3,0.0])

def toggle_flecha_derecha():
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_2)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    arduino.sendData([0.0,-2.0])

def toggle_flecha_arriba():
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_2)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    arduino.sendData([0.3,0.0])

def toggle_flecha_izq():
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_2)
    btn_boton_parar.config( image=img_path_Boton_gris)
    arduino.sendData([0.0,2.0])

def toggle_boton_parar():
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_azul)
    arduino.sendData([0.0,0.0])

def onClossing():
    arduino.sendData([0,0])
    arduino.close()
    root.quit()        #Salir del bucle de eventos.
    root.destroy()     #Destruye la ventana creada


############################## Serial communication ############
port = 'COM7'
arduino = serialArduino(port)
arduino.readSerialStart()

################################ HMI design ##################
```

```
root = Tk()
root.protocol("WM_DELETE_WINDOW",onClossing)
root.title("PRACTICA #1") # titulo de la ventana
root.geometry("525x600")    # Definir el tamaño de la ventana principal

lblcomandos=Label(text="Control del Robot",font=("Arial Narrow",12)).place(x=208,y=555)
lbltitle=Label(text="Práctica #1:",font=("Bahnschrift SemiBold SemiConden",14)).place(x=28,y=150)
lbltext=Label(text="Configuración y manejo del robot inalámbricamente",font=("Bahnschrift SemiCondensed",14)).place(x=115,y=150)
lbltext1=Label(text="(Prueba de conexión, manejo a través de comandos en el computador)",font=("Bahnschrift SemiCondensed",14)).place(x=2,y=180)

img = ImageTk.PhotoImage(Image.open(path))
panel = tk.Label(root, image = img)
panel.pack(side = "top", fill = "both", expand = "yes")
panel.place(x=10, y=0)

img_path_flecha_abajo_1 =
ImageTk.PhotoImage(Image.open(path_flecha_abajo_1).resize((150,100)))
panel_flecha_abajo_1 = tk.Label(root, image = img_path_flecha_abajo_1)

img_path_flecha_abajo_2 =
ImageTk.PhotoImage(Image.open(path_flecha_abajo_2).resize((150,100)))
panel_flecha_abajo_2 = tk.Label(root, image = img_path_flecha_abajo_2)

img_path_flecha_arriba_1 =
ImageTk.PhotoImage(Image.open(path_flecha_arriba_1).resize((150,100)))
panel_flecha_arriba_1 = tk.Label(root, image = img_path_flecha_arriba_1)

img_path_flecha_arriba_2 =
ImageTk.PhotoImage(Image.open(path_flecha_arriba_2).resize((150,100)))
panel_flecha_arriba_2 = tk.Label(root, image = img_path_flecha_arriba_2)

img_path_flecha_derecha_1 =
ImageTk.PhotoImage(Image.open(path_flecha_derecha_1).resize((100,150)))
panel_flecha_derecha_1 = tk.Label(root, image = img_path_flecha_derecha_1)

img_path_flecha_derecha_2 =
ImageTk.PhotoImage(Image.open(path_flecha_derecha_2).resize((100,150)))
panel_flecha_derecha_2 = tk.Label(root, image = img_path_flecha_derecha_2)

img_path_flecha_izq_2 =
ImageTk.PhotoImage(Image.open(path_flecha_izq_2).resize((100,150)))
panel_flecha_izq_2 = tk.Label(root, image = img_path_flecha_izq_2)

img_path_flecha_izq_1 =
ImageTk.PhotoImage(Image.open(path_flecha_izq_1).resize((100,150)))
panel_flecha_izq_1 = tk.Label(root, image = img_path_flecha_izq_1)

img_path_Boton_azul =
ImageTk.PhotoImage(Image.open(path_boton_azul).resize((150,100)))
panel_Boton_azul = tk.Label(root, image = img_path_Boton_azul)
```

```
img_path_Boton_gris                                                    =
ImageTk.PhotoImage(Image.open(path_boton_gris).resize((150,100)))
panel_Boton_gris = tk.Label(root, image = img_path_Boton_gris)


btn_flecha_abajo = Checkbutton(root, width=12, indicator=False,
            command=toggle_flecha_abajo, image=img_path_flecha_abajo_1)
btn_flecha_abajo.place(x=190, y=440, width=150, height=100)

btn_flecha_derecha = Checkbutton(root, width=12, indicator=False,
            command=toggle_flecha_derecha, image=img_path_flecha_derecha_1)
btn_flecha_derecha.place(x=340, y=315, width=100, height=150)

btn_flecha_arriba = Checkbutton(root, width=12, indicator=False,
            command=toggle_flecha_arriba, image=img_path_flecha_arriba_1)
btn_flecha_arriba.place(x=190, y=242, width=150, height=100)

btn_flecha_izq= Checkbutton(root, width=12,  indicator=False,
            command=toggle_flecha_izq, image=img_path_flecha_izq_1)
btn_flecha_izq.place(x=90, y=315, width=100, height=150)

btn_boton_parar= Checkbutton(root, width=12,  indicator=False,
            command=toggle_boton_parar, image=img_path_Boton_gris)
btn_boton_parar.place(x=190, y=342, width=150, height=100)


root.bind('<w>',adelante)
root.bind('<a>',izquierda)
root.bind('<d>',derecha)
root.bind('<s>',parar)
root.bind('<x>',atras)
root.mainloop()
```

ANEXO M. PROGRAMACIÓN EN PYTHON – PRÁCTICA #2.

```
from pyArduino import *
from tkinter import *
from PIL import Image, ImageTk
import sys
import cv2
import numpy as np
import imutils
import tkinter as tk
import tkinter.messagebox

def onClossing():
    arduino.sendData([0,0])
    arduino.close()
    root.quit()         #Salir del bucle de eventos.
    cap.release()       #Cerrar camara
    print("Camara USB desconectada")
```

```
    root.destroy()      #Destruye la ventana creada

path = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/logoUPS.png'
path_flecha_abajo_1                                                    =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/abajo_off.jpeg'
path_flecha_abajo_2                                                    =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/abajo_on.jpeg'
path_flecha_arriba_1                                                   =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/arriba_off.jpeg'
path_flecha_arriba_2                                                   =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/arriba_on.jpeg'
path_flecha_derecha_1                                                  =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/derecha_off.jpeg'
path_flecha_derecha_2                                                  =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/derecha_on.jpeg'
path_flecha_izq_1                                                      =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/izquierda_off.jpeg'
path_flecha_izq_2                                                      =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/izquierda_on.jpeg'
path_boton_gris = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/paro_off.png'
path_boton_azul = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/paro_on.png'
path_led_buzzer_on                                                     =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/led_buzzer_on.jpeg'
path_led_buzzer_off                                                    =
'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/flechas/led_buzzer_off.jpeg'


def adelante(w):
    arduino.sendData([0.3,0.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_2)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    btn_led_buzzer.config(image=img_path_led_buzzer_off)
    label_adelante = Label(root,text="Adelante")
    label_adelante.place(x=760, y=70)

def derecha(d):
    arduino.sendData([0.0,-2.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_2)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    btn_led_buzzer.config(image=img_path_led_buzzer_off)
    label_der = Label(root,text="Derecha")
    label_der.place(x=760, y=70)

def izquierda(a):
    arduino.sendData([0.0,2.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
```

```
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_2)
    btn_boton_parar.config( image=img_path_Boton_gris)
    btn_led_buzzer.config(image=img_path_led_buzzer_off)
    label_izq = Label(root,text="Izquierda")
    label_izq.place(x=760, y=70)

def parar(s):
    arduino.sendData([0.0,0.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_azul)
    btn_led_buzzer.config(image=img_path_led_buzzer_off)
    label_parar = Label(root,text="Paro        ")
    label_parar.place(x=760, y=70)

def atras(x):
    arduino.sendData([-0.3,0.0])
    btn_flecha_abajo.config( image=img_path_flecha_abajo_2)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    btn_led_buzzer.config(image=img_path_led_buzzer_off)
    label_atras = Label(root,text="Atrás      ")
    label_atras.place(x=760, y=70)

def letraF(f):
    arduino.sendData("F")
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    btn_led_buzzer.config(image=img_path_led_buzzer_on)
    label_led_buzzer = Label(root,text="Led y Buzzer On")
    label_led_buzzer.place(x=710, y=215)

def letraB(b):
    arduino.sendData("B")
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    btn_led_buzzer.config(image=img_path_led_buzzer_off)
    label_led_buzzer = Label(root,text="Led y Buzzer Off")
    label_led_buzzer.place(x=710, y=215)

def toggle_flecha_abajo():
    btn_flecha_abajo.config(image=img_path_flecha_abajo_2)
```

```
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    arduino.sendData([-0.3,0.0])
    label_atras = Label(root,text="Atrás        ")
    label_atras.place(x=760, y=70)

def toggle_flecha_derecha():
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_2)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    arduino.sendData([0.0,-2.0])
    label_der = Label(root,text="Derecha")
    label_der.place(x=760, y=70)

def toggle_flecha_arriba():
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_2)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_gris)
    arduino.sendData([0.3,0.0])
    label_adelante = Label(root,text="Adelante")
    label_adelante.place(x=760, y=70)

def toggle_flecha_izq():
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_2)
    btn_boton_parar.config( image=img_path_Boton_gris)
    arduino.sendData([0.0,2.0])
    label_izq = Label(root,text="Izquierda")
    label_izq.place(x=760, y=70)

def toggle_boton_parar():
    btn_flecha_abajo.config( image=img_path_flecha_abajo_1)
    btn_flecha_arriba.config( image=img_path_flecha_arriba_1)
    btn_flecha_derecha.config( image=img_path_flecha_derecha_1)
    btn_flecha_izq.config( image=img_path_flecha_izq_1)
    btn_boton_parar.config( image=img_path_Boton_azul)
    arduino.sendData([0.0,0.0])
    label_parar = Label(root,text="Paro        ")
    label_parar.place(x=760, y=70)


    ################## Procesamiento de la Imagen #################

def callback():
```

```
        ################### Adquisición de la Imagen #############
        global var1, bateria, phi

        ret, frame = cap.read() # Leer Frame
        phi.set(arduino.rawData[0])
        bateria.set(arduino.rawData[1])


        varPhi.set("Orientacion: "+str(phi.get())+" [rad/s]")
        varBateria.set("Batería Simulada: "+str(bateria.get())+" [V]")

        if ret:
            img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            img = Image.fromarray(img)
            img.thumbnail((420,420))
            tkimage = ImageTk.PhotoImage(img)
            label.configure(image = tkimage)
            label.image = tkimage
            root.after(10,callback)

        else:
            onClossing()


############################## Ip Cam ############################

url='http://192.168.137.45:8080/shot.jpg'

cap = cv2.VideoCapture(1)
if cap.isOpened():
    print("Camara USB inicializada")
else:
    sys.exit("Camara USB desconectada")

#cap.open(url)
ret, frame = cap.read()

############################## Serial communication ############

port = 'COM7'
sizeData = 2
arduino = serialArduino(port,sizeData=sizeData)
arduino.readSerialStart()

################################ HMI design #################

root = Tk()
root.protocol("WM_DELETE_WINDOW",onClossing)
root.title("PRACTICA #2") # titulo de la ventana
root.geometry("1000x620")    # Definir el tamaño de la ventana principal


lblvideo=Label(text="Visualización en Cámara",font=("Arial Narrow",12)).place(x=680,y=580)
```

```
lblcomandos=Label(text="Control del Robot",font=("Arial Narrow",12)).place(x=208,y=580)
lbldir=Label(text="Dirección:").place(x=705,y=70)

lbltitle=Label(text="Práctica #2:",font=("Bahnschrift SemiBold SemiConden",14)).place(x=25,y=150)
lbltext=Label(text=" Telemetría robótica (Comunicación bidireccional, ",font=("Bahnschrift SemiCondensed",14)).place(x=115,y=150)
lbltext1=Label(text="adquisición de datos del prototipo, configuración e implementación",font=("Bahnschrift SemiCondensed",14)).place(x=4,y=180)
lbltext2=Label(text="en microcontrolador e interfaz en computador).",font=("Bahnschrift SemiCondensed",14)).place(x=80,y=210)
lbltext3=Label(text="Adquisición de Datos",font=("Bahnschrift SemiBold SemiConden",16)).place(x=670,y=35)

img = ImageTk.PhotoImage(Image.open(path))
panel = tk.Label(root, image = img)
panel.pack(side = "top", fill = "both", expand = "yes")
panel.place(x=5, y=0)

#Labels
phi = DoubleVar(root,0)
varPhi = StringVar(root,"Orientación : 0.00")
labelPhi = Label(root, textvariable = varPhi)
labelPhi.place(x=690, y=90)

bateria = DoubleVar(root,0)
varBateria = StringVar(root,"Batería Simulada : 0.00")
labelBateria = Label(root, textvariable = varBateria)
labelBateria.place(x=685, y=110)


img_path_flecha_abajo_1 =
ImageTk.PhotoImage(Image.open(path_flecha_abajo_1).resize((150,100)))
panel_flecha_abajo_1 = tk.Label(root, image = img_path_flecha_abajo_1)

img_path_flecha_abajo_2 =
ImageTk.PhotoImage(Image.open(path_flecha_abajo_2).resize((150,100)))
panel_flecha_abajo_2 = tk.Label(root, image = img_path_flecha_abajo_2)

img_path_flecha_arriba_1 =
ImageTk.PhotoImage(Image.open(path_flecha_arriba_1).resize((150,100)))
panel_flecha_arriba_1 = tk.Label(root, image = img_path_flecha_arriba_1)

img_path_flecha_arriba_2 =
ImageTk.PhotoImage(Image.open(path_flecha_arriba_2).resize((150,100)))
panel_flecha_arriba_2 = tk.Label(root, image = img_path_flecha_arriba_2)

img_path_flecha_derecha_1 =
ImageTk.PhotoImage(Image.open(path_flecha_derecha_1).resize((100,150)))
panel_flecha_derecha_1 = tk.Label(root, image = img_path_flecha_derecha_1)

img_path_flecha_derecha_2 =
ImageTk.PhotoImage(Image.open(path_flecha_derecha_2).resize((100,150)))
```

```
panel_flecha_derecha_2 = tk.Label(root, image = img_path_flecha_derecha_2)

img_path_flecha_izq_2                                                    =
ImageTk.PhotoImage(Image.open(path_flecha_izq_2).resize((100,150)))
panel_flecha_izq_2 = tk.Label(root, image = img_path_flecha_izq_2)

img_path_flecha_izq_1                                                    =
ImageTk.PhotoImage(Image.open(path_flecha_izq_1).resize((100,150)))
panel_flecha_izq_1 = tk.Label(root, image = img_path_flecha_izq_1)

img_path_Boton_azul                                                      =
ImageTk.PhotoImage(Image.open(path_boton_azul).resize((150,100)))
panel_Boton_azul = tk.Label(root, image = img_path_Boton_azul)

img_path_Boton_gris                                                      =
ImageTk.PhotoImage(Image.open(path_boton_gris).resize((150,100)))
panel_Boton_gris = tk.Label(root, image = img_path_Boton_gris)

img_path_led_buzzer_on                                                   =
ImageTk.PhotoImage(Image.open(path_led_buzzer_on).resize((180,60)))
panel_led_buzzer_on = tk.Label(root, image = img_path_led_buzzer_on)

img_path_led_buzzer_off                                                  =
ImageTk.PhotoImage(Image.open(path_led_buzzer_off).resize((180,60)))
panel_led_buzzer_off = tk.Label(root, image = img_path_led_buzzer_off)


label=Label(root) #image = imagen camara opencv / relief = decoracion de borde
label.grid(row=1, column=1, padx=540,pady=250)


btn_flecha_abajo = Checkbutton(root, width=12, indicator=False,
        command=toggle_flecha_abajo, image=img_path_flecha_abajo_1)
btn_flecha_abajo.place(x=190, y=470, width=150, height=100)

btn_flecha_derecha = Checkbutton(root, width=12, indicator=False,
        command=toggle_flecha_derecha, image=img_path_flecha_derecha_1)
btn_flecha_derecha.place(x=340, y=345, width=100, height=150)

btn_flecha_arriba = Checkbutton(root, width=12, indicator=False,
        command=toggle_flecha_arriba, image=img_path_flecha_arriba_1)
btn_flecha_arriba.place(x=190, y=272, width=150, height=100)

btn_flecha_izq= Checkbutton(root, width=12,  indicator=False,
        command=toggle_flecha_izq, image=img_path_flecha_izq_1)
btn_flecha_izq.place(x=90, y=345, width=100, height=150)

btn_boton_parar= Checkbutton(root, width=12,  indicator=False,
        command=toggle_boton_parar, image=img_path_Boton_gris)
btn_boton_parar.place(x=190, y=372, width=150, height=100)

btn_led_buzzer= Checkbutton(root, width=12,  indicator=False,
        command=toggle_boton_parar, image=img_path_led_buzzer_off)
```

```
btn_led_buzzer.place(x=670, y=155, width=180, height=60)

root.bind('<w>',adelante)
root.bind('<a>',izquierda)
root.bind('<d>',derecha)
root.bind('<s>',parar)
root.bind('<x>',atras)
root.bind('<f>',letraF)
root.bind('<b>',letraB)
root.after(10,callback) #Es un método definido para todos los widgets tkinter.
root.mainloop()
```

ANEXO N. PROGRAMACIÓN EN PYTHON – PRÁCTICA #3.

```
from tkinter import *
import cv2
import numpy as np
import time
import imutils
import tkinter as tk
import tkinter.messagebox
from PIL import Image, ImageTk
import serial
import serial.tools.list_ports
from math import *
import sys

path = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/logoUPS_opt.jpg'
dataDict = {}

camHeight = 450
camWidth = 450
#url='http://192.168.137.209:8080/shot.jpg'
cam = cv2.VideoCapture(1)       # Selecciona cámara ( 0 es el default de la primera cámara)

if cam.isOpened():
    print("Camara USB inicializada")
else:
    sys.exit("Camara USB desconectada")

cam.set(3, camWidth)            # Configura el ancho del vídeo
cam.set(4, camHeight)           # Configura el alto del vídeo

getPixelColor = False           # Flag que nos indica si tenemos escogido color
H , S , V = 0,0,0               # Propiedades iniciales de pixel buscado
mouseX , mouseY = 0, 0          # Variables iniciales del mouse donde se ubicará para escoger
color

controllerWindow = tk.Tk()
controllerWindow.title("PRÁCTICA #3")
controllerWindow.geometry("400x630")
```

```
controllerWindow.resizable(0, 0)

lbltext=Label(text="Práctica #3: Configuración de sistema de visión artificial",font=("Bahnschrift SemiCondensed",13)).place(x=10,y=120)
lbltext1=Label(text="(Determinar colores, ubicación del robot en ",font=("Bahnschrift SemiCondensed",13)).place(x=45,y=145)
lbltext2=Label(text="el plano, punto de carga)",font=("Bahnschrift SemiCondensed",13)).place(x=100,y=170)

img = ImageTk.PhotoImage(Image.open(path))
panel = tk.Label(controllerWindow, image = img)
panel.pack(side = "top", fill = "both", expand = "yes")
panel.place(x=0, y=0)

videoWindow = tk.Toplevel(controllerWindow)
videoWindow.title("Vídeo de cámara")
videoWindow.resizable(0, 0)  #Si la pantalla es modificable
lmain = tk.Label(videoWindow)
lmain.pack()
videoWindow.withdraw()


def getMouseClickPosition(mousePosition):
    global mouseX,mouseY
    global getPixelColor
    mouseX,mouseY = mousePosition.x,mousePosition.y
    getPixelColor = True

showVideoWindow = False

def showCameraFrameWindow():
    global showVideoWindow, showGraph
    global BRetourVideoTxt
    if showVideoWindow == False:
        if showGraph == True:
            graphWindow.withdraw()

        videoWindow.deiconify()
        showVideoWindow = True
        BRetourVideo["text"] = "Quitar vídeo "
    else:
        videoWindow.withdraw()
        showVideoWindow = False
        BRetourVideo["text"] = "Mostrar vídeo"

showVideoWindow = False

showCalqueCalibrationBool = False
def showCalqueCalibration():
    global showCalqueCalibrationBool
    showCalqueCalibrationBool = not showCalqueCalibrationBool

showGraph = False
```

```
def showGraphWindow():
    global showGraph, showVideoWindow

    if showGraph == False:
        if showVideoWindow == True:
            videoWindow.withdraw()
            showVideoWindow = False
            BRetourVideo["text"] = "Mostrar vídeo"
        showGraph = True

    else:
        showGraph = False

t = 480

def endProgam():
    controllerWindow.destroy()

sliderHDefault = 15
sliderSDefault = 70
sliderVDefault = 70

def resetSlider():
    sliderH.set(sliderHDefault)
    sliderS.set(sliderSDefault)
    sliderV.set(sliderVDefault)

def donothing():
    pass

start_time = 0
def main():
    start_timeFPS = time.time()
    global H,S,V
    global getPixelColor
    global x,y, alpha, beta
    global camWidth,camHeight
    global timeInterval, start_time
    global showVideoWindow
    global imgHSV

    #cam.open(url) # Antes de capturar el frame abrimos la url
    _, img=cam.read()

    ret, frame = cam.read() # Leer Frame
    img      =      img[0:int(camHeight),int((camWidth-camHeight)/2):int(camWidth-((camWidth-
camHeight)/2))] #[Y1:Y2,X1:X2]

    imgHSV = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

    if getPixelColor == True and mouseY > 0 and mouseY < 480 and mouseX < 480:
        pixelColorOnClick = img[mouseY,mouseX]
        pixelColorOnClick = np.uint8([[pixelColorOnClick]])
```

```
        pixelColorOnClick = cv2.cvtColor(pixelColorOnClick,cv2.COLOR_BGR2HSV)
        H = pixelColorOnClick[0,0,0]
        S = pixelColorOnClick[0,0,1]
        V = pixelColorOnClick[0,0,2]
        T.delete("1.0","end")
        T.insert(tk.END, "H: " +str(H)+" S: " +str(S)+" V: " +str(V)+"\n")
        print(mouseX, mouseY)
        print(H, S, V )
        getPixelColor = False

    lowerBound=np.array([H-sliderH.get(),S-sliderS.get(),V-sliderV.get()])
    upperBound=np.array([H+sliderH.get(),S+sliderS.get(),V+sliderV.get()])

    mask=cv2.inRange(imgHSV,lowerBound,upperBound)
    mask = cv2.blur(mask,(6,6))                    # ajoute du flou a limage
    mask = cv2.erode(mask, None, iterations=2)        # retire les parasites

    cnts                                =                                cv2.findContours(mask.copy(),
cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours (cnts)

    center = None

    if showCalqueCalibrationBool == True:
        mask = cv2.dilate(mask, None, iterations=2)       # retire les parasites
        res = cv2.bitwise_and(img,img, mask= mask)
        cv2.imshow('Mascara',mask)
        cv2.imshow('Solo Objeto',res)

    if len(cnts) > 0:
        c = max(cnts, key=cv2.contourArea)
        timeInterval = time.time() - start_time
        (x, y), radius = cv2.minEnclosingCircle(c)
        if radius > 10:
            cv2.putText(img,str(int(x))    +    ";"    +    str(int(y)).format(0,    0),(int(x)-50,    int(y)-50),
cv2.FONT_HERSHEY_SIMPLEX,1, (255, 255, 255), 2)
            cv2.circle(img, (int(x), int(y)), int(radius),(0, 255, 255), 2)

            start_time = time.time()
    else:
        sommeErreurX, sommeErreurY = 0,0

    if showVideoWindow == True:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = Image.fromarray(img)
        imgtk = ImageTk.PhotoImage(image=img)
        lmain.imgtk = imgtk
        lmain.configure(image=imgtk)
    lmain.after(5, main)

FrameVideoControl = tk.LabelFrame(controllerWindow, text="Controles de vídeo")
FrameVideoControl.place(x=5,y=195,width=380)
```

```
BRetourVideo = tk.Button(FrameVideoControl, text="Mostrar vídeo", command=showCameraFrameWindow)
BRetourVideo.pack()
BPositionCalibration = tk.Button(FrameVideoControl, text="Capa límites", command=showCalqueCalibration)
BPositionCalibration.place(x=280,y=0)

sliderH = tk.Scale(FrameVideoControl, from_=0, to=100, orient="horizontal", label="Hue", length=350, tickinterval = 10)
sliderH.set(sliderHDefault)
sliderH.pack()
sliderS = tk.Scale(FrameVideoControl, from_=0, to=100, orient="horizontal", label="Saturation", length=350, tickinterval = 10)
sliderS.set(sliderSDefault)
sliderS.pack()
sliderV = tk.Scale(FrameVideoControl, from_=0, to=100, orient="horizontal", label="Value", length=350, tickinterval = 10)
sliderV.set(sliderVDefault)
sliderV.pack()

Frame_HSV = tk.LabelFrame(controllerWindow, text="Valores H, S, V ")
Frame_HSV.place(x=5,y=480,width=380, height=230)
T = tk.Text(Frame_HSV, height=5, width=30)
T.pack()
T.insert(tk.END, "Valores HSV escogidos\n")

BReset = tk.Button(controllerWindow, text = "Reset", command = resetSlider)
BReset.place(x=108, y=590)
BQuit = tk.Button(controllerWindow, text = "Salir", command = endProgam)
BQuit.place(x=200, y=590)

videoWindow.protocol("WM_DELETE_WINDOW",donothing)
videoWindow.bind("<Button-2>",getMouseClickPosition)

main()
tk.mainloop()
```

ANEXO O. PROGRAMACIÓN EN PYTHON – PRÁCTICA #4.

```
from pyArduino import *
from tkinter import *
from PIL import Image, ImageTk
import cv2
import numpy as np
import sys
import tkinter as tk
import tkinter.messagebox

g_estado =0
b=0
valors = "A"
```

```
path = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/logoUPS_opt.jpg'

def toggle():
    btn.config(text=btnVar.get())

def toggle_aut():
    btntarea_aut.config(text=btnVar_aut.get())

def toggle1():
    btntarea1.config(text=btnVar1.get())

def toggle2():
    btntarea2.config(text=btnVar2.get())

def toggle3():
    btntarea3.config(text=btnVar3.get())

def toggle4():
    btntarea4.config(text=btnVar4.get())

def onClossing():
    arduino.sendData([0,0])
    arduino.close()
    root.quit()         #Salir del bucle de eventos.
    cap.release()       #Cerrar camara
    print("Camara USB desconectada")
    root.destroy()      #Destruye la ventana creada

def hsvValue(int):
    hMin.set(slider1.get())
    hMax.set(slider2.get())
    sMin.set(slider3.get())
    sMax.set(slider4.get())
    vMin.set(slider5.get())
    vMax.set(slider6.get())

def hsvValuea(int):
    hMina.set(slider1a.get())
    hMaxa.set(slider2a.get())
    sMina.set(slider3a.get())
    sMaxa.set(slider4a.get())
    vMina.set(slider5a.get())
    vMaxa.set(slider6a.get())

def valor(int):
    valorx.set(sliderx.get())
    valory.set(slidery.get())

def objectDetection(rawImage):
    global a
    global mask, maska
    kernel = np.ones((10,10),np.uint8) # Filtro
    isObject = False     # Verdadero si encuentra un objeto
```

```
    cx,cy = 0,0          #centroide (x), centroide (y)
    cx1,cy1 = 0,0          #centroide (x), centroide (y)
    cxd,cyd = 0,0        #centroide (x) deseado, centroide (y) deseado
    cont = 0 # variable auxiliar
    minArea = 20  # Area mínima para considerar que es un objeto

    ################## Procesamiento de la Imagen ################

    hsv = cv2.cvtColor(rawImage, cv2.COLOR_BGR2HSV) #Convertirlo a espacio de color
HSV

    lower=np.array([hMin.get(),sMin.get(),vMin.get()])
    upper=np.array([hMax.get(),sMax.get(),vMax.get()])

    lowera=np.array([hMina.get(),sMina.get(),vMina.get()])
    uppera=np.array([hMaxa.get(),sMaxa.get(),vMaxa.get()])

    azulBajo = np.array([100,100,20],np.uint8)
    azulAlto = np.array([125,255,255],np.uint8)

    #Deteccion de colores
    mask = cv2.inRange(hsv, lower, upper)
    maska = cv2.inRange(hsv, lowera, uppera)
    maskazul = cv2.inRange(hsv,azulBajo,azulAlto)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
    maska = cv2.morphologyEx(maska, cv2.MORPH_CLOSE, kernel)
    maskazul1 = cv2.morphologyEx(maskazul, cv2.MORPH_CLOSE, kernel)
    contours,_          =          cv2.findContours(mask.copy(),          cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    contoursa,_          =          cv2.findContours(maska.copy(),          cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    contours1,_          =          cv2.findContours(maskazul.copy(),          cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    ######################CENTROIDE ROBOT#################################
    for cnt in contours:
        #Calcular el centro a partir de los momentos
        momentos = cv2.moments(cnt)
        area = momentos['m00']
        if (area>minArea):
            approx = cv2.approxPolyDP(cnt,0.01*cv2.arcLength(cnt,True),True)

            if len(approx) > 1:
                cv2.drawContours(rawImage,[cnt],0,(0,255,255),-1)
                cx = int(momentos['m10']/momentos['m00'])
                cy = int(momentos['m01']/momentos['m00'])
                cont = cont+1
    ################################CIRCULOS#####################################
    for cnta in contoursa:
        #Calcular el centro a partir de los momentos
        momentos1 = cv2.moments(cnta)
        area1 = momentos1['m00']
        if (area1>minArea):
```

```
            approx = cv2.approxPolyDP(cnta,0.01*cv2.arcLength(cnta,True),True)
            if len(approx)>0:
                #Objeto
                cv2.drawContours(rawImage,[cnta],0,(255,0,255),-1)
                cxd = int(momentos1['m10']/momentos1['m00'])
                cyd = int(momentos1['m01']/momentos1['m00'])
                cont = cont+1

    if cont == 2:
        isObject = True
    else:
        isObject = False

    return isObject,mask,rawImage,cx,cy,cxd,cyd

def callback():

        ################### Adquisición de la Imagen #############
        global maska,b
        global g_estado
        global distance
        global cxd, cyd
        global valors
        global var1
        #cap.open(url) # Antes de capturar el frame abrimos la url
        ret, frame = cap.read() # Leer Frame

        if ret:

            isObject,mask,frame,cx,cy,cxd,cyd = objectDetection(frame)
            minDist = 37
            minDist1 = 20

            cv2.circle(frame,(cx,cy),10, (255,0,0), -1)
            cv2.circle(frame,(cxd,cyd),minDist, (0,255,0),3)


            if btnVar4.get() == 'Tarea 4 On':
                cxd = valorx.get()
                cyd = valory.get()
            if isObject:

                ################### Conversion coordenadas #############
                hx = cx-frame.shape[1]/2
                hy = frame.shape[0]/2-cy

                ######################### Desired position in pixels ###############
                hxd = cxd - frame.shape[1]/2
                hyd = frame.shape[0]/2 - cyd

                # Distancia minima de error (Distancia Euclidiana)
                distance = np.sqrt((hxd-hx)**2+(hyd-hy)**2)
```

```
    if distance>minDist:

        ################### Control cámara en mano #############
        a = 0.07 # punto de interes en metros (Donde se coloca el objeto)
        phi.set(arduino.rawData[0])

        # Errores
        hxe  =  hxd-hx
        hye  =  hyd-hy


        he = np.array([[hxe],[hye]]);

        K = np.diag([0.001,0.001])

        J = np.array([[-np.sin(phi.get()),-a*np.cos(phi.get())],
                      [ np.cos(phi.get()),-a*np.sin(phi.get())]])
        # Ley de control
        qp = np.linalg.inv(J)@(K@he);

        uRef.set(round(qp[0][0],3))
        wRef.set(round(qp[1][0],3))

    else:
        uRef.set(0)
        wRef.set(0)
        b=b+1

else:

    uRef.set(0)
    wRef.set(0)

varU.set("Linear velocity : "+str(uRef.get())+" [m/s]")
varW.set("Angular velocity : "+str(wRef.get())+" [rad/s]")
varPhi.set("Orientacion: "+str(phi.get())+" [rad/s]")

# Boton de inicio
if btnVar.get() == 'Start':
    arduino.sendData([uRef.get(),wRef.get()])
if btnVar_aut.get() == 'Tarea Automatica On':
    if distance<minDist:
        if g_estado == 0:
            print("Punto A")
                            #agregar accion 1
            slider1a.set(12)
            slider3a.set(149)
            slider5a.set(64)
            slider2a.set(15)
            slider4a.set(194)
            slider6a.set(255)
            g_estado = 1
        elif g_estado == 1:
```

```
            print("Punto B")
                                #agregar acion 2
            slider1a.set(23)
            slider3a.set(138)
            slider5a.set(123)
            slider2a.set(30)
            slider4a.set(209)
            slider6a.set(255)
            g_estado = 2
        elif g_estado == 2:
            print("Punto C")
                                #agregar acion 2
            slider1a.set(59)
            slider3a.set(90)
            slider5a.set(105)
            slider2a.set(255)
            slider4a.set(121)
            slider6a.set(150)
            g_estado = 0
if btnVar1.get() == 'Tarea 1 On':
    #Objeto Naranja
    slider1a.set(12)
    slider3a.set(149)
    slider5a.set(64)
    slider2a.set(15)
    slider4a.set(194)
    slider6a.set(255)
if btnVar2.get() == 'Tarea 2 On':
    #Objeto Amarillo
    slider1a.set(23)
    slider3a.set(138)
    slider5a.set(123)
    slider2a.set(30)
    slider4a.set(209)
    slider6a.set(255)
if btnVar3.get() == 'Tarea 3 On':
    #Color verde
    slider1a.set(59)
    slider3a.set(90)
    slider5a.set(105)
    slider2a.set(255)
    slider4a.set(121)
    slider6a.set(150)
if btnVar4.get() == 'Tarea 4 On':
    cxd = valorx.get()
    cyd = valory.get()
    hxd = cxd - frame.shape[1]/2
    hyd = frame.shape[0]/2 - cyd
    isObject = True
    cv2.circle(frame,(cxd,cyd),minDist1, (0,255,255),3)
    print("Siguiendo el punto")

# Mostrar imagen en el HMI
```

```
            img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            img = Image.fromarray(img)
            img.thumbnail((420,420))
            tkimage = ImageTk.PhotoImage(img)
            label.configure(image = tkimage)
            label.image = tkimage

            mask = mask.astype("float32")
            maska = maska.astype("float32")
            result = 255*(mask + maska)
            result = result.clip(0, 255).astype("uint8")
            img1 = Image.fromarray(result)
            img1.thumbnail((420,420))
            tkimage1 = ImageTk.PhotoImage(img1)
            label1.configure(image = tkimage1)
            label1.image = tkimage1

            root.after(10,callback)

        else:
            onClossing()

############################## Ip Cam ############################
#url='http://192.168.100.125:8080/shot.jpg'
#url='http://192.168.137.33:8080/shot.jpg'

cap = cv2.VideoCapture(1)

if cap.isOpened():
    print("Camara USB inicializada")
else:
    sys.exit("Camara USB desconectada")

############################## Serial communication ############

port = 'COM7'
sizeData = 2
arduino = serialArduino(port,sizeData=sizeData)
arduino.readSerialStart()

################################ HMI design #################

root = Tk()
root.protocol("WM_DELETE_WINDOW",onClossing)
root.title("PRÁCTICA #4") # titulo de la ventana
root.geometry("1366x768")    # Definir el tamaño de la ventana principal


lbltext=Label(text="Práctica #4: Gestión multitarea autónoma",font=("Bahnschrift
SemiCondensed",15)).place(x=975,y=180)
lbltext1=Label(text="(Integración de tareas: determinación de colores y ",font=("Bahnschrift
SemiCondensed",15)).place(x=945,y=210)
```

```
lbltext2=Label(text="trayectoria entre varios puntos asignados)",font=("Bahnschrift SemiCondensed",15)).place(x=975,y=240)


img = ImageTk.PhotoImage(Image.open(path))
panel = tk.Label(root, image = img)
panel.pack(side = "top", fill = "both", expand = "yes")
panel.place(x=935, y=35)

label=Label(root) #image = imagen camara opencv / relief = decoracion de borde
label.grid(padx=30,pady=20)

label1=Label(root)
label1.grid(row = 0,column=1,padx=0,pady=20)

uRef = DoubleVar(root,0)
varU = StringVar(root,"Linear velocity : 0.00")
labelU = Label(root, textvariable = varU)
labelU.grid(row = 1,padx=20,pady=10)

wRef = DoubleVar(root,0)
varW = StringVar(root,"Angular velocity : 0.00")
labelW = Label(root, textvariable = varW)
labelW.grid(row= 1,column = 1, padx=20,pady=10)

phi = DoubleVar(root,0)
varPhi = StringVar(root,"Orientation : 0.00")
labelPhi = Label(root, textvariable = varPhi)
labelPhi.grid(row= 2, padx=20,pady=10)


hMin = IntVar()
hMax = IntVar()
sMin = IntVar()
sMax = IntVar()
vMin = IntVar()
vMax = IntVar()

hMina = IntVar()
hMaxa = IntVar()
sMina = IntVar()
sMaxa = IntVar()
vMina = IntVar()
vMaxa = IntVar()
valorx = IntVar()
valory = IntVar()


slider1 = Scale(root,label = 'Hue Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200) #Creamos un dial para recoger datos numericos
slider1.place(x=20, y=440)
```

```
slider2 = Scale(root,label = 'Hue Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200) #Creamos un dial para recoger datos numericos
slider2.place(x=250, y=440)

slider3 = Scale(root,label = 'Saturation Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200) #Creamos un dial para recoger datos numericos
slider3.place(x=20, y=510)

slider4 = Scale(root,label = 'Saturation Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200) #Creamos un dial para recoger datos numericos
slider4.place(x=250, y=510)

slider5 = Scale(root,label = 'Value Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200) #Creamos un dial para recoger datos numericos
slider5.place(x=20, y=580)

slider6 = Scale(root,label = 'Value Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200) #Creamos un dial para recoger datos numericos
slider6.place(x=250, y=580)


slider1a = Scale(root,label = 'Hue Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200) #Creamos un dial para recoger datos numericos
slider1a.place(x=480, y=440)

slider2a = Scale(root,label = 'Hue Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200) #Creamos un dial para recoger datos numericos
slider2a.place(x=710, y=440)

slider3a = Scale(root,label = 'Saturation Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200) #Creamos un dial para recoger datos numericos
slider3a.place(x=480, y=510)

slider4a = Scale(root,label = 'Saturation Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200) #Creamos un dial para recoger datos numericos
slider4a.place(x=710, y=510)

slider5a = Scale(root,label = 'Value Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200) #Creamos un dial para recoger datos numericos
slider5a.place(x=480, y=580)
```

```
slider6a = Scale(root,label = 'Value Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200) #Creamos un dial para recoger datos numericos
slider6a.place(x=710, y=580)

sliderx = Scale(root,label = 'Punto X', from_=0, to=600, orient=HORIZONTAL,command=valor,length=200) #Creamos un dial para recoger datos numericos
sliderx.place(x=930, y=580)
slidery = Scale(root,label = 'Punto Y', from_=0, to=600, orient=HORIZONTAL,command=valor,length=200) #Creamos un dial para recoger datos numericos
slidery.place(x=1150, y=580)


slider1.set(88)
slider3.set(147)
slider5.set(62)

slider2.set(120)
slider4.set(253)
slider6.set(209)

slider1a.set(21)
slider3a.set(126)
slider5a.set(147)

slider2a.set(88)
slider4a.set(208)
slider6a.set(182)

sliderx.set(255)
slidery.set(255)

btnVar = StringVar(root, 'Pause')
btnVar_aut = StringVar(root, 'Tarea Automatica Off')
btnVar1 = StringVar(root, 'Tarea 1 Off')
btnVar2 = StringVar(root, 'Tarea 2 Off')
btnVar3 = StringVar(root, 'Tarea 3 Off')
btnVar4 = StringVar(root, 'Tarea 4 Off')

btn = Checkbutton(root, text=btnVar.get(), width=12, variable=btnVar,
            offvalue='Pause', onvalue='Start', indicator=False,
            command=toggle)
btn.place(x=200, y=670)

btntarea_aut = Checkbutton(root, text=btnVar_aut.get(), width=15, variable=btnVar_aut,
            offvalue='Tarea Automatica Off', onvalue='Tarea Automatica On', indicator=False,
            command=toggle_aut)
btntarea_aut.place(x=1070, y=300, width=150, height=40)

btntarea1 = Checkbutton(root, text=btnVar1.get(), width=14, variable=btnVar1,
            offvalue='Tarea 1 Off', onvalue='Tarea 1 On', indicator=False,
```

```
            command=toggle1)
btntarea1.place(x=1070, y=350, width=150, height=40)

btntarea2 = Checkbutton(root, text=btnVar2.get(), width=14, variable=btnVar2,
            offvalue='Tarea 2 Off', onvalue='Tarea 2 On', indicator=False,
            command=toggle2)
btntarea2.place(x=1070, y=400, width=150, height=40)

btntarea3 = Checkbutton(root, text=btnVar3.get(), width=14, variable=btnVar3,
            offvalue='Tarea 3 Off', onvalue='Tarea 3 On', indicator=False,
            command=toggle3)
btntarea3.place(x=1070, y=450, width=150, height=40)

btntarea4 = Checkbutton(root, text=btnVar4.get(), width=14, variable=btnVar4,
            offvalue='Tarea 4 Off', onvalue='Tarea 4 On', indicator=False,
            command=toggle4)
btntarea4.place(x=1070, y=500, width=150, height=40)

root.after(10,callback) #Es un método definido para todos los widgets tkinter.
root.mainloop()
```

ANEXO P. PROGRAMACIÓN EN PYTHON – PRÁCTICA #5.

```
from pyArduino import *
from tkinter import *
from PIL import Image, ImageTk
import cv2
import numpy as np
import sys
import tkinter as tk
import tkinter.messagebox

g_estado =0
b=0
valors = "A"
bateriav=0
path = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/logoUPS.png'
path_bvacia = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/b_vacia.jpeg'
path_binicial = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/b_inicial.jpeg'
path_bmedia = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/b_media.jpeg'
path_bfull = 'C:/Users/Agudo/Desktop/Tesis_Agudo_Gómez/b_full.jpeg'

def bsimulada(g):
    arduino.sendData("G")
    label_bsimu = Label(root,text="Batería Simulada")
    label_bsimu.place(x=710, y=592)


def breal(h):
    arduino.sendData("H")
    label_bre = Label(root,text="Batería Real       ")
    label_bre.place(x=710, y=592)
```

```
def letraF(f):
    arduino.sendData("F")

def letraB(b):
    arduino.sendData("B")

def toggle():
    btn.config(text=btnVar.get())

def toggle_aut():
    btntarea_aut.config(text=btnVar_aut.get())

def toggle1():
    btntarea1.config(text=btnVar1.get())
    #reload(pyArduino)
def toggle2():
    btntarea2.config(text=btnVar2.get())

def toggle3():
    btntarea3.config(text=btnVar3.get())

def toggle5():
    btntarea5.config(text=btnVar5.get())

def toggle_boton_parar():
    btn_boton_parar.config(image=img_path_Boton_azul)
    arduino.sendData([0.0,0.0])

def onClossing():
    arduino.sendData([0,0])
    arduino.close()
    root.quit()         #Salir del bucle de eventos.
    cap.release()       #Cerrar camara
    print("Ip Cam Disconected")
    root.destroy()      #Destruye la ventana creada


def hsvValue(int):
    hMin.set(slider1.get())
    hMax.set(slider2.get())
    sMin.set(slider3.get())
    sMax.set(slider4.get())
    vMin.set(slider5.get())
    vMax.set(slider6.get())

def hsvValuea(int):
    hMina.set(slider1a.get())
    hMaxa.set(slider2a.get())
    sMina.set(slider3a.get())
    sMaxa.set(slider4a.get())
    vMina.set(slider5a.get())
    vMaxa.set(slider6a.get())
```

```
def valor(int):
    valorx.set(sliderx.get())
    valory.set(slidery.get())

def objectDetection(rawImage):
    global a
    global mask, maska
    kernel = np.ones((10,10),np.uint8) # Filtro
    isObject = False     # Verdadero si encuentra un objeto
    cx,cy = 0,0          #centroide (x), centroide (y)
    cx1,cy1 = 0,0          #centroide (x), centroide (y)
    cxd,cyd = 0,0        #centroide (x) deseado, centroide (y) deseado
    cont = 0 # variable auxiliar
    minArea = 20  # Area minima para considerar que es un objeto

    ################## Procesamiento de la Imagen ################

    hsv = cv2.cvtColor(rawImage, cv2.COLOR_BGR2HSV) #Convertirlo a espacio de color
HSV

    #Se crea un array con las posiciones minimas y maximas capturadas de los sliders
    lower=np.array([hMin.get(),sMin.get(),vMin.get()])
    upper=np.array([hMax.get(),sMax.get(),vMax.get()])

    lowera=np.array([hMina.get(),sMina.get(),vMina.get()])
    uppera=np.array([hMaxa.get(),sMaxa.get(),vMaxa.get()])

    azulBajo = np.array([100,100,20],np.uint8)
    azulAlto = np.array([125,255,255],np.uint8)

    #Deteccion de colores
    mask = cv2.inRange(hsv, lower, upper)
    maska = cv2.inRange(hsv, lowera, uppera)
    maskazul = cv2.inRange(hsv,azulBajo,azulAlto)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
    maska = cv2.morphologyEx(maska, cv2.MORPH_CLOSE, kernel)
    maskazul1 = cv2.morphologyEx(maskazul, cv2.MORPH_CLOSE, kernel)
    contours,_          =          cv2.findContours(mask.copy(),          cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    contoursa,_          =          cv2.findContours(maska.copy(),          cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    contours1,_         =         cv2.findContours(maskazul.copy(),         cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    for cnt in contours:
        #Calcular el centro a partir de los momentos
        momentos = cv2.moments(cnt)
        area = momentos['m00']
        if (area>minArea):
            approx = cv2.approxPolyDP(cnt,0.01*cv2.arcLength(cnt,True),True)

            if len(approx) > 1:
```

```
            cv2.drawContours(rawImage,[cnt],0,(0,255,255),-1)
            cx = int(momentos['m10']/momentos['m00'])
            cy = int(momentos['m01']/momentos['m00'])
            cont = cont+1

    for cnta in contoursa:
        #Calcular el centro a partir de los momentos
        momentos1 = cv2.moments(cnta)
        area1 = momentos1['m00']
        if (area1>minArea):
            approx = cv2.approxPolyDP(cnta,0.01*cv2.arcLength(cnta,True),True)
            if len(approx)>0:
                #Objeto
                cv2.drawContours(rawImage,[cnta],0,(255,0,255),-1)
                cxd = int(momentos1['m10']/momentos1['m00'])
                cyd = int(momentos1['m01']/momentos1['m00'])
                cont = cont+1

    if cont == 2:
        isObject = True
    else:
        isObject = False

    return isObject,mask,rawImage,cx,cy,cxd,cyd

def callback():

        ################### Adquisición de la Imagen #############
        global maska,b
        global g_estado
        global distance
        global cxd, cyd
        global valors
        global var1
        global bateriav
        #cap.open(url) # Antes de capturar el frame abrimos la url
        ret, frame = cap.read() # Leer Frame

        if ret:

            isObject,mask,frame,cx,cy,cxd,cyd = objectDetection(frame)
            minDist = 37

            cv2.circle(frame,(cx,cy),10, (255,0,0), -1)
            cv2.circle(frame,(cxd,cyd),minDist, (0,255,0),3)

            if isObject:

                ################### Conversion coordenadas #############

                hx = cx-frame.shape[1]/2
                hy = frame.shape[0]/2-cy
```

```
    ######################### Desired position in pixels ###############

    hxd = cxd - frame.shape[1]/2
    hyd = frame.shape[0]/2 - cyd

    # Distancia minima de error (Distancia Euclidiana)
    distance = np.sqrt((hxd-hx)**2+(hyd-hy)**2)

    if distance>minDist:

        ################### Control cámara en mano ############
        a = 0.07 # punto de interes en metros (Donde se coloca el objeto)
        phi.set(arduino.rawData[0])
        bateria.set(arduino.rawData[1])
        bateriav=arduino.rawData[1]

        # Errores
        hxe  =  hxd-hx
        hye  =  hyd-hy

        he = np.array([[hxe],[hye]]);

        K = np.diag([0.001,0.001])

        J = np.array([[-np.sin(phi.get()),-a*np.cos(phi.get())],
                      [ np.cos(phi.get()),-a*np.sin(phi.get())]])
        # Ley de control
        qp = np.linalg.inv(J)@(K@he);

        uRef.set(round(qp[0][0],3))
        wRef.set(round(qp[1][0],3))
        # if bateriav<2.0:
        #     cxd = valorx.get()
        #     cyd = valory.get()

    else:
        uRef.set(0)
        wRef.set(0)
        b=b+1
else:

    uRef.set(0)
    wRef.set(0)

varU.set("Linear velocity : "+str(uRef.get())+" [m/s]")
varW.set("Angular velocity : "+str(wRef.get())+" [rad/s]")
varPhi.set("Orientacion: "+str(phi.get())+" [rad/s]")
varBateria.set("Nivel Bateria: "+str(bateria.get())+" [V]")

# Boton de inicio
if bateriav<0.1:
    imgv = ImageTk.PhotoImage(Image.open(path_bvacia).resize((90,20)))
    widget1 = Label(root, image=imgv)
```

```
            widget1.image = imgv
            widget1.place(x=750, y=530)

        if bateriav >0.1 and bateriav <2.0:

            imgi = ImageTk.PhotoImage(Image.open(path_binicial).resize((90,20)))
            widget2 = tkinter.Label(root, image=imgi)
            widget2.image = imgi
            widget2.place(x=750, y=530)

        if bateriav >=2.0 and bateriav <=4.0:
            imgm = ImageTk.PhotoImage(Image.open(path_bmedia).resize((90,20)))
            widget3 = Label(root, image=imgm)
            widget3.image = imgm
            widget3.place(x=750, y=530)

        if bateriav>4.0:
            #print("Cargado")
            imgf = ImageTk.PhotoImage(Image.open(path_bfull).resize((90,20)))
            widget4 = Label(root, image=imgf)
            widget4.image = imgf
            widget4.place(x=750, y=530)

##############################                                    Tareas
#########################################
        if btnVar.get() == 'Start':
            arduino.sendData([uRef.get(),wRef.get()])
        if btnVar_aut.get() == 'Tarea Automatica On':
            if distance<minDist:
                if g_estado == 0:
                                    #agregar accion 1
                    hMina.set(9)
                    sMina.set(149)
                    vMina.set(64)
                    hMaxa.set(15)
                    sMaxa.set(194)
                    vMaxa.set(255)
                    g_estado = 1
                elif g_estado == 1:
                                    #agregar acion 2
                    hMina.set(18)
                    sMina.set(138)
                    vMina.set(123)
                    hMaxa.set(30)
                    sMaxa.set(209)
                    vMaxa.set(255)
                    g_estado = 2
                elif g_estado == 2:
                                    #agregar acion 2
                    hMina.set(59)
                    sMina.set(90)
                    vMina.set(105)
                    hMaxa.set(255)
```

```
            sMaxa.set(121)
            vMaxa.set(150)
            g_estado = 0
if btnVar1.get() == 'Tarea 1 On':
    #Objeto Naranja
    hMina.set(9)
    sMina.set(149)
    vMina.set(64)
    hMaxa.set(15)
    sMaxa.set(194)
    vMaxa.set(255)
if btnVar2.get() == 'Tarea 2 On':
    #Objeto Amarillo
    hMina.set(18)
    sMina.set(138)
    vMina.set(123)
    hMaxa.set(30)
    sMaxa.set(209)
    vMaxa.set(255)
if btnVar3.get() == 'Tarea 3 On':
    #Color verde
    hMina.set(59)
    sMina.set(90)
    vMina.set(105)
    hMaxa.set(255)
    sMaxa.set(121)
    vMaxa.set(150)
if btnVar5.get() == 'Punto de Carga On':
    bateriav=arduino.rawData[1]
    #g_estado = 5
    if bateriav<2.0:
        print("Punto de Carga")
        hMina.set(0)
        sMina.set(150)
        vMina.set(30)
        hMaxa.set(27)
        sMaxa.set(205)
        vMaxa.set(134)
    if bateriav>4.0:
        print("Cargado")
        g_estado == 0

# Mostrar imagen en el HMI
img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
img = Image.fromarray(img)
img.thumbnail((420,420))
tkimage = ImageTk.PhotoImage(img)
label.configure(image = tkimage)
label.image = tkimage

mask = mask.astype("float32")
maska = maska.astype("float32")
result = 255*(mask + maska)
```

```
            result = result.clip(0, 255).astype("uint8")
            img1 = Image.fromarray(result)
            img1.thumbnail((420,420))
            tkimage1 = ImageTk.PhotoImage(img1)
            label1.configure(image = tkimage1)
            label1.image = tkimage1

            root.after(10,callback)

        else:
            onClossing()

############################## Ip Cam ############################
#url='http://192.168.100.125:8080/shot.jpg'
#url='http://192.168.137.33:8080/shot.jpg'

cap = cv2.VideoCapture(1)

if cap.isOpened():
    print("Camara USB inicializada")
else:
    sys.exit("Camara USB desconectada")

############################## Serial communication ############

port = 'COM7'
sizeData = 2
arduino = serialArduino(port,sizeData=sizeData)
arduino.readSerialStart()

################################ HMI design #################

root = Tk()
root.protocol("WM_DELETE_WINDOW",onClossing)
root.title("PRÁCTICA #5") # titulo de la ventana
root.geometry("950x715")    # Definir el tamaño de la ventana principal

lbltext=Label(text="Práctica #5: Desarrollo de carga de batería del robot ",font=("Bahnschrift
SemiCondensed",15)).place(x=510,y=50)
lbltext1=Label(text="y movilización autónoma a punto de carga entre
tareas.",font=("Bahnschrift SemiCondensed",15)).place(x=500,y=80)

img = ImageTk.PhotoImage(Image.open(path).resize((490,140)))
panel = tk.Label(root, image = img)
panel.pack(side = "top", fill = "both", expand = "yes")
panel.place(x=5, y=5)

label=Label(root) #image = imagen camara opencv / relief = decoracion de borde
label.place(x=30, y=170)

label1=Label(root)
label1.place(x=500, y=170)
```

```
uRef = DoubleVar(root,0)
varU = StringVar(root,"Linear velocity : 0.00")
labelU = Label(root, textvariable = varU)
labelU.place(x=150, y=500)

wRef = DoubleVar(root,0)
varW = StringVar(root,"Angular velocity : 0.00")
labelW = Label(root, textvariable = varW)
labelW.place(x=145, y=530)

phi = DoubleVar(root,0)
varPhi = StringVar(root,"Orientation : 0.00")
labelPhi = Label(root, textvariable = varPhi)
labelPhi.place(x=650, y=500)

bateria = DoubleVar(root,0)
varBateria = StringVar(root,"Nivel de Bateria : 0.00")
labelBateria = Label(root, textvariable = varBateria)
labelBateria.place(x=620, y=530)


hMin = IntVar()
hMax = IntVar()
sMin = IntVar()
sMax = IntVar()
vMin = IntVar()
vMax = IntVar()

hMina = IntVar()
hMaxa = IntVar()
sMina = IntVar()
sMaxa = IntVar()
vMina = IntVar()
vMaxa = IntVar()
valorx = IntVar()
valory = IntVar()

#####SLIDERS COMENTADOS#############
# slider1 = Scale(root,label = 'Hue Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200) #Creamos un dial para recoger datos numericos
# #slider1.place(x=20, y=440)
# slider1.place(x=800, y=800)
# #
# slider2 = Scale(root,label = 'Hue Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200) #Creamos un dial para recoger datos numericos
# #slider2.place(x=250, y=440)
# slider2.place(x=800, y=800)
# #
# slider3 = Scale(root,label = 'Saturation Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200) #Creamos un dial para recoger datos numericos
```

```
# #slider3.place(x=20, y=510)
# slider3.place(x=800, y=800)
# #
# slider4 = Scale(root,label = 'Saturation Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200) #Creamos un dial para recoger datos numericos
# #slider4.place(x=250, y=510)
# slider4.place(x=800, y=800)
# #
# slider5 = Scale(root,label = 'Value Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200) #Creamos un dial para recoger datos numericos
# #slider5.place(x=20, y=580)
# slider5.place(x=800, y=800)
# #
# slider6 = Scale(root,label = 'Value Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValue,length=200) #Creamos un dial para recoger datos numericos
# #slider6.place(x=250, y=580)
# slider6.place(x=800, y=800)
# #
# slider1a = Scale(root,label = 'Hue Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200) #Creamos un dial para recoger datos numericos
# #slider1a.place(x=480, y=440)
# slider1a.place(x=800, y=800)
# #
# slider2a = Scale(root,label = 'Hue Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200) #Creamos un dial para recoger datos numericos
# #slider2a.place(x=710, y=440)
# slider2a.place(x=800, y=800)
# #
# slider3a = Scale(root,label = 'Saturation Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200) #Creamos un dial para recoger datos numericos
# #slider3a.place(x=480, y=510)
# slider3a.place(x=800, y=800)
# #
# slider4a = Scale(root,label = 'Saturation Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200) #Creamos un dial para recoger datos numericos
# #slider4a.place(x=710, y=510)
# slider4a.place(x=800, y=800)
# #
# slider5a = Scale(root,label = 'Value Min', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200) #Creamos un dial para recoger datos numericos
# #slider5a.place(x=480, y=580)
# slider5a.place(x=800, y=800)
# #
```

```
# slider6a = Scale(root,label = 'Value Max', from_=0, to=255, orient=HORIZONTAL,command=hsvValuea,length=200) #Creamos un dial para recoger datos numericos
# #slider6a.place(x=710, y=580)
# slider6a.place(x=800, y=800)
# #
# sliderx = Scale(root,label = 'Punto Carga X', from_=0, to=600, orient=HORIZONTAL,command=valor,length=200) #Creamos un dial para recoger datos numericos
# sliderx.place(x=800, y=800)
#
# slidery = Scale(root,label = 'Punto Carga y', from_=0, to=600, orient=HORIZONTAL,command=valor,length=200) #Creamos un dial para recoger datos numericos
# slidery.place(x=800, y=800)
#
################################

hMin.set(88)
sMin.set(147)
vMin.set(62)

hMax.set(120)
sMax.set(253)
vMax.set(209)

hMina.set(21)
sMina.set(126)
vMina.set(147)

hMaxa.set(88)
sMaxa.set(208)
vMaxa.set(182)

valorx.set(39)
valory.set(329)

btnVar = StringVar(root, 'Pause')
btnVar_aut = StringVar(root, 'Tarea Automatica Off')
btnVar1 = StringVar(root, 'Tarea 1 Off')
btnVar2 = StringVar(root, 'Tarea 2 Off')
btnVar3 = StringVar(root, 'Tarea 3 Off')
btnVar5 = StringVar(root, 'Punto de Carga Off')
btn = Checkbutton(root, text=btnVar.get(), width=12, variable=btnVar,
            offvalue='Pause', onvalue='Start', indicator=False,
            command=toggle)
btn.place(x=415, y=670)

btntarea_aut = Checkbutton(root, text=btnVar_aut.get(), width=15, variable=btnVar_aut,
            offvalue='Tarea Automatica Off', onvalue='Tarea Automatica On', indicator=False,
            command=toggle_aut)
btntarea_aut.place(x=50, y=560, width=150, height=40)
```

```
btntarea1 = Checkbutton(root, text=btnVar1.get(), width=14, variable=btnVar1,
                offvalue='Tarea 1 Off', onvalue='Tarea 1 On', indicator=False,
                command=toggle1)
btntarea1.place(x=50, y=610, width=150, height=40)

btntarea2 = Checkbutton(root, text=btnVar2.get(), width=14, variable=btnVar2,
                offvalue='Tarea 2 Off', onvalue='Tarea 2 On', indicator=False,
                command=toggle2)
btntarea2.place(x=300, y=560, width=150, height=40)

btntarea3 = Checkbutton(root, text=btnVar3.get(), width=14, variable=btnVar3,
                offvalue='Tarea 3 Off', onvalue='Tarea 3 On', indicator=False,
                command=toggle3)
btntarea3.place(x=300, y=610, width=150, height=40)

btntarea5 = Checkbutton(root, text=btnVar5.get(), width=14, variable=btnVar5,
                offvalue='Punto de Carga Off', onvalue='Punto de Carga On', indicator=False,
                command=toggle5)
btntarea5.place(x=550, y=585, width=150, height=40)

root.bind('<g>',bsimulada)
root.bind('<h>',breal)
root.after(10,callback) #Es un método definido para todos los widgets tkinter.
root.mainloop()
```