UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE GUAYAQUIL

CARRERA DE INGENIERÍA EN ELECTRÓNICA

PROYECTO TÉCNICO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO ELECTRÓNICO

TEMA:

"IMPLEMENTACIÓN DE MÓDULO DE ENTRENAMIENTO PARA CONTROL PD DE UN SISTEMA BALLAND PLATE PARA LA UNIVERSIDAD POLITÉCNICA SALESIANA"

AUTORES:

VALAREZO CHÁVEZ ARTURO FERNANDO
SOLÓRZANO HINOSTROZA JORGE GABRIEL

TUTOR:

MSC. MÓNICA MIRANDA

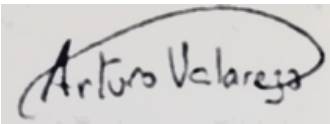
GUAYAQUIL - ECUADOR
2021

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, **ARTURO FERNANDO VALAREZO CHÁVEZ** y **JORGE GABRIEL SOLORZANO HINOSTROZA** autorizamos a la **UNIVERSIDAD POLITÉCNICA SALESIANA** la publicación total o parcial de este trabajo de titulación y su reproducción sin fines de lucro.

Además, se declara que los conceptos y análisis desarrollados y conclusiones del presente trabajo son de exclusiva responsabilidad del autor.

Guayaquil, 19 de Julio del 2021



Arturo Valarezo Chávez
CI: 0916997869

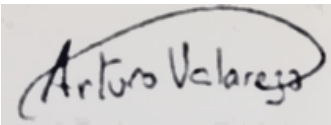
Jorge Solórzano Hinostroza
CI: 2450341868

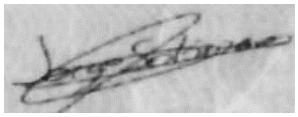
CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UPS

Nosotros, **ARTURO FERNANDO VALAREZO CHÁVEZ**, con documento de identificación Nº 0916997869 y **JORGE GABRIEL SOLORZANO HINOSTROZA**, con documento de identificación Nº 2450341868, manifestamos nuestra voluntad y cedemos a la **UNIVERSIDAD POLITÉCNICA SALESIANA** la titularidad sobre los derechos patrimoniales en virtud de que somos autores del trabajo de grado titulado: **"IMPLEMENTACIÓN DE MÓDULO DE ENTRENAMIENTO PARA CONTROL PD DE UN SISTEMA BALLAND PLATE PARA LA UNIVERSIDAD POLITÉCNICA SALESIANA"** mismo que 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 antes cedidos.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en nuestra condición de autor me reservo los derechos morales de la obra antes citada. En concordancia, suscrito de este documento en el momento que hacemos entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Politécnica Salesiana.

Guayaquil, 19 de Julio del 2021





Arturo Valarezo Chávez	Jorge Solórzano Hinostroza
CI: 0916997869	CI: 2450341868

CERTIFICADO DE DIRECCIÓN DE TRABAJO DE TITULACIÓN SUSCRITO POR EL TUTOR

Por medio de la presente doy constancia que los señores **ARTURO FERNANDO VALAREZO CHÁVEZ** y **JORGE GABRIEL SOLORZANO HINOSTROZA** han desarrollado y elaborado satisfactoriamente bajo mi dirección el siguiente proyecto **"IMPLEMENTACIÓN DE MÓDULO DE ENTRENAMIENTO PARA CONTROL PD DE UN SISTEMA BALLAND PLATE PARA LA UNIVERSIDAD POLITÉCNICA SALESIANA "**, cumpliendo las normas establecidas por la Universidad Politécnica Salesiana, para ser considerado como trabajo final de titulación.

Guayaquil, 19 de Julio del 2021

Ing. Mónica Miranda Ramos, Msc.

DEDICATORIA

Dedico este proyecto de titulación a todas las personas que han sido parte de este camino, que de una u otra manera contribuyeron a que cada día avance siempre un paso a la vez a pesar de las adversidades.

ARTURO FERNANDO VALAREZO CHÁVEZ

El presente trabajo va dedicado a mi madre por ser mi apoyo incondicional en mi formación profesional y moral, a mi abuelita que en paz descanse por haberme educado y corregido cuando era necesario, a mi hermanita por ser de ayuda moral, y a mi primo por haber sido mi única figura paterna durante mi infancia.

JORGE GABRIEL SOLORZANO HINOSTROZA

AGRADECIMIENTOS

Las palabras de agradecimiento quedan cortas, por todas esas anécdotas de quienes me ayudaron de muchas maneras a lo largo del camino; familiares, amigos, profesores y conocidos que sin esperar nada a cambio, permitieron que ahora logre cumplir una meta más de vida. A mis padres por darme una educación basada en valores y responsabilidad con mucho esfuerzo. A mi abuela quien es un gran apoyo en todos los aspectos donde siempre con su temple me empuja a mejorar cada día. Mi hermana que siempre se encuentra ahí para escucharme los problemas, ayudándome muchas veces a resolverlos con sus consejos. A mis familiares que en distintas épocas de vida y más en mi carrera, fueron parte de apoyo en esos momentos donde la situación no era la mejor, pero creyeron en mí. Mis profesores, que más allá de las aulas, fueron mentores con sus consejos al ver que mis calificaciones a veces no eran las mejores, pero siempre dándome ánimos para continuar. Y a mi amigo Jorge, que le pusimos tanto empeño para poder terminar con esta tesis y otros proyectos que salieron en el camino.

ARTURO FERNANDO VALAREZO CHÁVEZ

AGRADECIMIENTOS

Agradezco a mi madre por todo su apoyo que nuca me faltó a lo largo del curso de mi carrera, por aconsejarme y ayudarme cuando lo necesité y siempre estar allí para sus hijos, de ella solo puedo decir que realizó su papel de madre con total abnegación.

Agradezco mi abuelita por su educación durante mi infancia, me dio consejos y lecciones que estoy orgulloso de aplicar en mi vida personal y profesional, a ella le deseo paz y descanso en su tumba

También debo agradecer a todos los profesores, tutores y compañeros que conocí en la Universidad Politécnica Salesiana, a la Ing. Mónica Miranda por su trabajo como tutora y su visión para la docencia, al Ing. Byron Lima por haber sido mi profesor favorito quien también impartió varias de mis materias favoritas, al Ing. Luis Córdova por ser también de mis profesores favoritos quién me impartió la materia de microcontrolador la cual también terminó siendo de mis materias favoritas.

Por último, no podría faltar mi agradecimiento a mi compañero Arturo Valarezo por su paciencia y visión de superación, por haber sido paciente con mis desaciertos y por compartir este logro.

JORGE GABRIEL SOLORZANO HINOSTROZA

RESUMEN

Este proyecto hace referencia al diseño e implementación de un módulo de entrenamiento para control PD por medio de visión artificial de un sistema ball and plate para la Universidad Politécnica Salesiana, basado en la necesidad de nuevas formas para desarrollo de prácticas estudiantiles a un costo accesible si desea ser replicado. El proyecto cuenta con varias etapas que reúnen habilidades que se aprenden a lo largo de la carrera para ser un ingeniero profesional, primero el desarrollo de una tarjeta PCB, después la conexión entre microcontroladores y la computadora a través de sus puertos de comunicación, posteriormente tenemos la impresión 3D y corte CNC, los cuales nos permiten hacer la estructura donde se alojan todos sus componentes, y por último la programación, que se basa en el lenguaje de programación Python donde se aplica el sistema de control, en este caso PD. Se desarrolló el proyecto a lo largo de 7 prácticas, en que los estudiantes podrán tener un entendimiento integral de cómo realizarlo por medio de visión artificial.

Palabras Claves: Python / Visión artificial / Control PD / Impresión 3D / CNC.

ABSTRACT

This article refers to de implementation of training module for PD controller by means of artificial vision of ball and plate system for the Politécnica Salesiana University, based on the need for new ways to develop student practices at an affordable cost if requires to be replicated. The project has several stages that bring together skills that are learned throughout the career to become an integral engineer, first the development of a PCB board, after the connection between microcontrollers and the computer through their communication ports, later 3D printing and CNC cutting, that allows us to make the structure where all its components are housed, and finally the programming, which is based on the Python language where the control system is applied, in this case PD. The project was developed over 7 practices, in which students will bring a comprehensive understanding of how to develop it by means of artificial vision.

Key words: Python / Artificial Vision / PD Control / 3D Printing / CNC.

Índice

Índice de Figuras

INTRODUCCIÓN

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 nuestro título en este caso la Ingeniería electrónica. Se propone el tema IMPLEMENTACIÓN DE MÓDULO DE ENTRENAMIENTO PARA CONTROL PD POR MEDIO DE VISIÓN ARTIFICIAL DE UN SISTEMA BALL AND PLATE PARA LA UNIVERSIDAD POLITÉCNICA SALESIANA que consiste en el diseño de un módulo de prácticas, el cual funciona usando varios tópicos que se toman a lo largo de la carrera, haciéndolo así ideal para un aprendizaje integral del estudiante.

Muchas veces al tomar una materia, nos enfocamos en el tema en específico que se trata en esta, y a veces nos vemos en la dificultad de unir conceptos mientras avanzamos en el estudio. Este prototipo conjuga el diseño e impresión en 3D a través de un diseño agradable al usuario, de sencilla replicación, permitiendo una metodología lúdica que ayuda al interés de este tema que se encuentra en pleno auge.

Como también la aplicación de visión artificial a través de una cámara como método de entrada al sistema de control, y mediante el uso de un lenguaje que es

cada vez más popular como lo es Python, se logra la integración de todos los componentes previamente dichos.

Luego se diseña una tarjeta electrónica para la conexión de la planta con la computadora, explicando cómo se encuentran las conexiones hechas para su funcionamiento para que en caso de no tener como fabricarla, se podría buscar un método alternativo al uso de una tarjeta microcontroladora prefabricada.

El presente trabajo de investigación se divide en cuatro capítulos. El Capítulo I describe el problema de investigación, así como la importancias, alcance y delimitación, planteando los objetivos a cumplir a lo largo de la misma. El Capítulo II profundiza en la fundamentación teórica, sentando las bases para la experimentación. El Capítulo III contiene todos los procedimientos utilizados para llevar a cabo el proyecto, desde la obtención teórica de las ecuaciones matemáticas hasta la construcción física de la planta. Finalmente en el capítulo IV se presentan los resultados obtenidos, así como las comparativas correspondientes para verificar el cumplimiento de objetivos generales y específicos.

CAPÍTULO I

EL PROBLEMA

1.1. Planteamiento del Problema

Se evidencia que los ingenieros graduados que inician en el ámbito laboral se encuentran faltos de herramientas, instrumentos y procedimientos técnicos propios de la industria. Un profesional que es competitivo y eficaz en sus labores cuenta con sólidos conocimientos de sistemas de control desde su instrucción superior, que le permita enfrentar las dificultades que se presenten en el campo laboral; a veces es complicado tener sistemas funcionales que tengan actualizado el conocimiento luego de haber terminado el periodo educativo, por lo tanto, se plantea una solución con costos asequibles y de uso práctico para los estudiantes.

1.2. Importancia y Alcance

El proyecto técnico complementa el desarrollo del aprendizaje para estudiantes universitarios, provee diversas herramientas como pruebas de concepto para implementar diferentes aplicaciones donde se requiera el uso de visión artificial y control PD. Es económico, portátil y de replicación versátil, que sienta un precedente para el reemplazo de módulos o kits de otras marcas, que son costosos, difíciles de transportar e implican varias limitaciones por las licencias con las que se adquieren.

La creación de herramientas didácticas para la realización de prácticas académicas busca mejorar la pericia de todos los egresados que ejercen profesionalmente en sus respectivos campos, con destreza profesional y experiencia relevante a aras de enfrentar y solucionar problemas reales de la industria en general.

1.3. Delimitación

1.3.1. Temporal

La implementación de este proyecto se realizó en el periodo lectivo número 57, entre los años 2019 y 2020.

1.3.2. 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 como se puede visualizar en la **Fig.1.**

Fig. 1 Vista superior del Campus Guayaquil

1.3.3. 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, como también control PD, siendo uno de los sistemas de control más robusto, y visión artificial que es una tecnología que utiliza y procesa las imágenes adquiridas mediante una cámara para el reconocimiento inteligente de objetos diversos en formas y colores.

1.4. Objetivos

1.4.1. Objetivo General

Diseñar e implementar un módulo de entrenamiento de un sistema Ball and Plate con control PD a través de visión artificial para que sea ubicado en el Laboratorio de Sistemas de Control Automático de la Universidad Politécnica Salesiana sede Guayaquil.

1.4.2. Objetivos Específicos

- Diseñar, identificar e implementar un sistema de control PD.
- Diseñar e implementar la tarjeta de control y comunicación que funcione mediante el software Python para la planta del sistema Ball Plate.

- Implementar un algoritmo de control PD en Python para la operación de la planta del sistema Ball Plate.
- Obtener los datos de la planta utilizando visión artificial.
- Elaboración de 7 prácticas didácticas para la operación del sistema por parte de los estudiantes guiados por docentes de la Universidad Politécnica Salesiana.

CAPÍTULO II

FUNDAMENTOS TEÓRICOS

2.1. Control PD

Está acción de control tiene dos parámetros de ganancias: proporcional y derivativo, combina las ventajas de estos tres tipos de controladores, al calcular la desviación o error entre un valor medido y un valor deseado. Es el controlador más común en sistemas de control industrial y su comportamiento depende directamente de que las magnitudes relativas de estos tres parámetros actúen en sintonía. En un principio se estudiaban las reglas de sintonía propuestas por Ziegler y Nichols para la selección de parámetros que cumplan la sintonía que requiere las especificaciones del proceso, pero hoy en día se cuenta con procedimientos para optimizar la sintonía de los parámetros para control PD mediante el uso de MATLAB como la muestra la **Fig. 2**, permitiendo obtener resultados que satisfacen cualquier necesidad de control industrial [1].

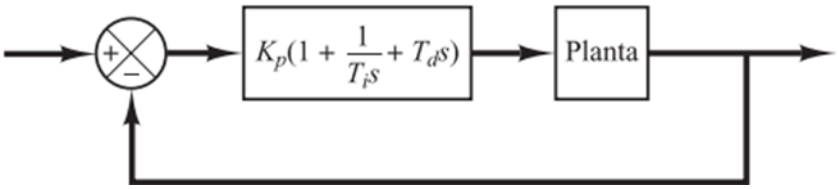


Fig. 2 Control PD de una planta [1].

2.2. Controlabilidad

La controlabilidad tiene que ver con la posibilidad de llevar al sistema de cualquier estado inicial a cualquier estado final en tiempo finito, no importando qué trayectoria se siga, o qué entrada se use [2].

En un sistema de n estados y p entradas:

$$\dot{x} = Ax + Bu$$

Con las matrices constantes de *A* y *B* formadas por el conjunto de números reales, la relación entre las entradas y la matriz de estados vuelve irrelevante la ecuación de salida. Entonces, todo sistema es controlable si para cualquier estado inicial y cualquier estado final, pertenecientes al conjunto de números reales, existe una entrada que transfiere el estado de inicial a final o viceversa, en un tiempo determinado.

2.3. Observabilidad

Su concepto está conjugado con el de controlabilidad, y se refiere a la posibilidad de estimar el estado del sistema a partir del conocimiento de la salida. En tal caso una ecuación es observable si para cualquier estado inicial desconocido, existe un tiempo finito tal que el conocimiento de la entrada *u*, la salida y sobre todo el intervalo, es suficiente para determinar en forma única el estado inicial. En caso contrario el sistema es no observable [2].

2.4. Python

A principios de los 90, Guido van Rossum creó un lenguaje de script fuertemente tipado, multiplataforma y orientado a objetos, y lo bautizó como Python en honor a Monty Python, un grupo de comediantes originarios de Inglaterra. Python es un lenguaje de sintaxis simple, clara y sencilla; se ejecuta a través de un programa intérprete que utiliza lenguaje compilado, lo cual facilita su ejecución para cualquier computadora; no es necesario declarar los tipos de las variables, sino que estas se determinan automáticamente en tiempo de ejecución, de acuerdo con el valor que contengan, pero si se requiere tratar una variable como si fuese de otro tipo es necesario convertirla de manera explícita [3].

2.4.1. Tkinter

Tkinter es una librería vinculante a la Tk GUI toolkit. Es la interfaz estándar de interfaz gráfica de usuario de Python para el conjunto de herramientas gráfica en Tk. Se incluye con la instalación estándar de Python en los sistemas operativos Linux, Microsoft Windows y Mac OS X. El nombre Tkinter proviene de interfaz Tk. Tkinter fue escrito por Fredrik Lundh. Tkinter es software libre publicado bajo una licencia de Python [4]. En la **Fig. 3**, se observa la interfaz gráfica del proyecto de tesis.

Fig. 3 Interfaz gráfica hecha con la librería Tkinter.
Autores.

2.4.2. Pillow

Python Imaging Library (PIL) es una librería gratuita que permite la edición de imágenes directamente desde Python. Soporta una variedad de formatos, incluidos los más utilizados como GIF, JPEG y PNG. Una gran parte del código está escrito en C, por cuestiones de rendimiento [5].

2.4.3. Imutils

Es una serie de funciones para manipulación de imagen que resultan mejor que algunas predefinidas Opencv o Matplolib. Entre estas podemos hacer Translación, rotación, o poner imágenes transparentes sobre otras [6].

2.4.4. Pyserial

Este módulo encapsula el acceso al puerto serial del computador a través de Python. Tiene soporte para Windows, OSX, Linux. Tiene licencia gratuita [7]. En la **Fig. 4**, se detalla el logo de la librería a usar.

Fig. 4 Logo de librería Pyserial [7]

2.5. Sistema Ball and Plate

El sistema *Ball & Plate* se encuentra en la mayoría de los laboratorios donde se estudia teoría de control, es un modelo didáctico de construcción sencilla, que permite comprender las diferentes acciones de control y la medición de sus parámetros. Está basado en el sistema *Ball & Beam* que tiene el mismo principio, pero con un solo eje.

2.6. Circuito Impreso

Se llama así al producto obtenido de la construcción de un circuito sobre una placa de cobre, de la cual luego se elimina el excedente de cobre, dejando únicamente las pistas del diseño impreso. Estas pistas unirán de manera sistemática los diversos componentes electrónicos para que su funcionamiento cumpla el objetivo requerido [8].

Entre los programas de diseño de tarjetas electrónicas en circuitos impresos podemos mencionar AutoDesk Eagle, Labcenter Electronics Proteus, que son muy robustos y competitivos en el mercado, pero así mismo se requiere la compra de licencias para su uso comercial. Con la finalidad de cumplir los objetivos de nuestro proyecto utilizaremos un software de diseño de licencia libre llamado KiCAD [9].

2.7. KiCAD EDA

Es un software de uso libre para el diseño de diagramas esquemáticos para circuitos electrónico o PCB con salidas tipo Gerber. Funciona en Windows, Linux y macOS. KiCAD es de uso libre, lo cual representa una ventaja para desarrolladores, estudiantes y demás usuarios que deseen aprender sobre elaboración y diseño de tarjetas electrónicas, ya que en su página web se puede descargar desde el software hasta las diferentes librerías necesarias. Se puede visualizar en la **Fig. 5**, el diseño de la tarjeta usada en el proyecto de tesis.

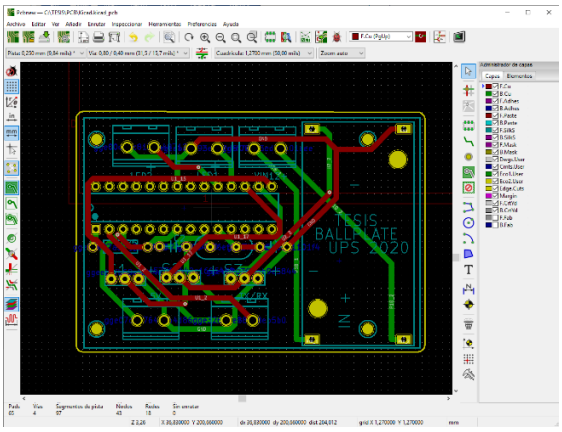


Fig. 5 Diseño del PCB en KiCAD EDA. Autores

2.8. Cámara Web

Son dispositivos que capturan y transmiten vídeo en tiempo real con una resolución y frecuencia determinada. Realizan transmisión de datos por interfaz USB.

2.9. Interfaz USB

Es un estándar de conexión sencilla, topología estrella, con protocolos que permiten la detección de dispositivos conectados y su configuración automática. Consiste en dos pares de cables, uno para transmisión de datos y otro para transferencia de corriente. La velocidad a la que transfieren datos depende del tipo de USB, el 2.0 se popularizó en el año 2000, su velocidad es de 280MBps, en el 2013 se lanzó USB 3.1, que alcanza velocidades de 10GBps [10].

2.10. CMOS Image Sensor

Es la tecnología utilizada al momento en la mayoría de dispositivos móviles o cámaras digitales, consiste en el procesamiento de cada píxel de forma individual, de manera que se producen voltajes individuales direccionados a través de una matriz hacia la salida, con bajo coste energético y calentamiento mínimo. Las matrices ofrecen la posibilidad de procesar las imágenes en líneas de píxeles, verticales u horizontales, en lugar de frame por frame como se hacía anteriormente, lo que facilita la captura de objetos en movimiento con frame rates bajos [11].

En la **Fig. 6**, se aprecia la compasión con ambos tipos de escaneado.

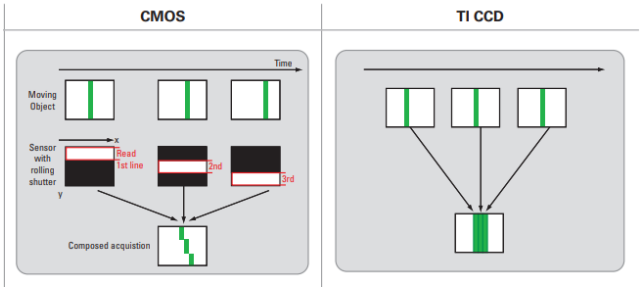


Fig. 6 Escaneado línea a línea CMOS, comparado a escaneado frame por frame [11].

2.11. Servomotor

Son motores que disponen de un encoder y un driver, el encoder realimenta al driver la información sobre la posición en la que se encuentra. Por medio de este controlador es posible ordenar al servo mantener o cambiar la posición en tiempo real y con una precisión muy alta [12].

En la **Fig. 7**, se muestra el diagrama de conexión del servomotor.

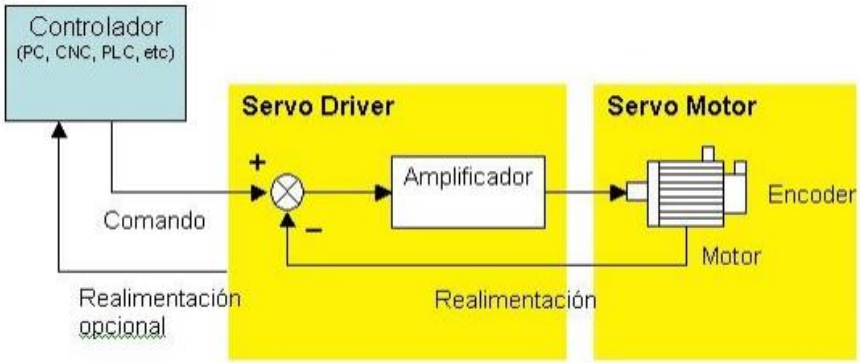


Fig. 7 Esquema de accionamiento de un sistema con servomotor [12].

2.12. Visión Artificial

La visión artificial es un método de obtención de datos en forma de imágenes que son traducidas por ordenadores capaces de reconocer, procesar y analizar

imágenes adquiridas por medio de cámaras para determinar propiedades geométricas (forma, tamaño, ubicación) y propiedades materiales (color, iluminación, textura y composición) con el fin de cuantificar o cualificar estos datos para facilitar su manejo. En la actualidad se estudia en diversos campos para brindar soluciones por etapas de acuerdo con su complejidad, de manera que se vuelve sencillo el reconocimiento de objetos tridimensionales o en movimiento, así como la segmentación y agrupación de objetos diferentes en entornos no controlados [13].

2.13. Open CV

OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por la compañía Intel. OpenCV significa Open Computer Vision (Visión Artificial Abierta). Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en una gran cantidad de aplicaciones, y hasta 2020 se la sigue mencionando como la biblioteca más popular de visión artificial.1 Detección de movimiento, reconocimiento de objetos, reconstrucción 3D a partir de imágenes, son sólo algunos ejemplos de aplicaciones de esta biblioteca [14].

En la **Fig. 8**, se muestra el logo de la librería de OpenCV.

Fig. 8 Logo de librería OpenCV [14].

2.14. Impresión 3D

Es un grupo de tecnologías que nacieron en 1984, y consistían en utilizar resinas líquidas fotopoliméricas para la construcción de objetos a menor escala capa por capa. El avance de esta tecnología fue lento hasta el 2014, cuando se creó la primera impresora open-source en Inglaterra. A partir de ahí la impresión 3D ha estado más cerca del consumidor, lo que ha ayudado a su evolución, tanto que en la actualidad se ha construido réplicas de coches a escala real y hasta edificación de viviendas pequeñas. [15].

En la **Fig. 9**, se visualiza la impresora PRUSA i3.

Fig. 9 Impresora 3D Marca PRUSA I3 MK2S [16].

Las impresoras 3D utilizan dos métodos para crear objetos, pueden ser por compactación de masa de polvo por estratos, o por inyección de polímeros capa por capa. Su estructura consiste en una tarjeta electrónica de control, que bien puede ser un Arduino MEGA, adicionando los SHIELDS y otros componentes como pantalla LCD, lector de tarjetas SD y un panel de botones; elegir el firmware a utilizar

también es importante, algunas alternativas pueden ser Sprinter, Marlin o Teacup Firmware. La cama caliente es una resistencia térmica que al energizarse aumenta de temperatura, lo cual es importante para la correcta adherencia de la primera capa de filamento derretido en todas nuestras creaciones. El extrusor consiste extremo caliente que tiene resistencia y termistor para calentar el filamento, y, el cuerpo con un motor paso a paso que empuja el filamento hacia el extremo caliente. La estructura está formada por un conjunto de motores paso a paso que mueven el extrusor y la mesa caliente en ejes X, Y, y Z [17].

2.15. Corte por Láser

Es la técnica sustractiva digital de grabado de materiales mediante láser. Utiliza tecnología CNC (Control Numérico Computarizado) en la que un ordenador traza la ruta de corte y hace que el láser pase sobre el material dando la forma deseada, resultando en el calentamiento, fusión y vaporización total o parcial del material, con una pérdida mínima de aproximadamente 0,5 mm [18].

En la **Fig. 10**, se visualiza un corte de láser sobre metal.

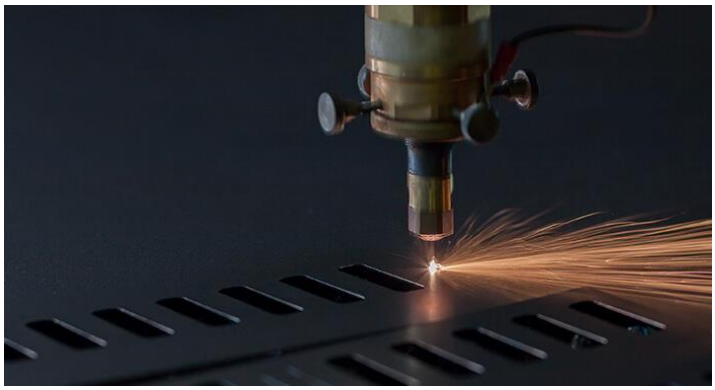


Fig. 10 Corte láser sobre metal. Fuente: [18].

Entre las ventajas de esta tecnología se mencionan la versatilidad en cuanto a la cantidad de materiales que puede cortar, implica menos gasto en logística en cuanto a procesos posteriores ya que casi en todos los materiales el calor del láser además de cortar los sella, es de alta precisión, con reconocimiento de marcas de registro que calculan la vía de corte así la plantilla se encontrara desalineada, y permite trabajo continuo puesto a que las piezas no están sujetas a desgaste como en otras máquinas de corte [19].

CAPÍTULO III

MARCO METODOLÓGICO

Una vez descrita la teoría en el capítulo II se procede con una descripción específica. En la **Fig. 11**, que se encuentra a continuación, se puede observar un diagrama esquemático.

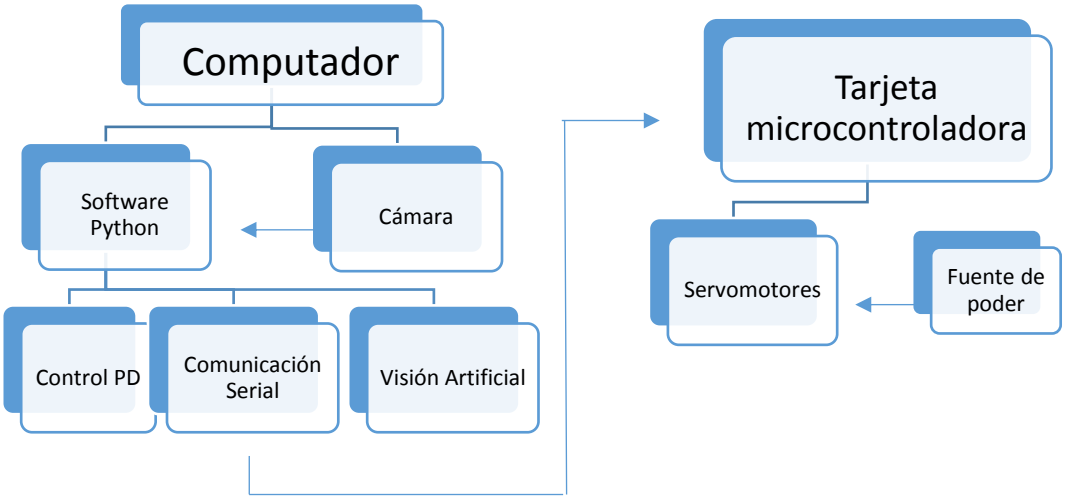


Fig. 11 Esquema de Proyecto. Autores.

3.1. Concepto del Sistema

El sistema contiene un controlador PD (proporcional, derivativo), servomotores, sensor óptico (cámara), plataforma y esfera. El control PD es conectado a un ordenador que contiene un software codificado en Python, y en conjunto resuelven la posición de los servomotores; el software se encarga de la visión artificial y de acuerdo a la información recolectada por el sensor óptico equilibra la esfera sobre

la plataforma. El control PD es ideal ya que funciona en lazo cerrado, realimentando la data para llevar a cabo el proceso de manera más eficiente.

Al arrancar el programa con un click con el botón central del mouse en la pantalla del vídeo adquirido en tiempo real, se escoge el objeto. Luego se calibran los parámetros H, S, y V, para eliminar el ruido y evitar la selección de píxeles fuera de los que corresponden al color del objeto (esfera). El siguiente paso es activar los motores; en principio la planta mantiene la esfera equilibrada en el centro, pero mediante un click dentro de la pantalla del vídeo en tiempo real, podemos elegir cualquier punto dentro de los límites establecidos como setpoint para realizar la acción de control PD.

3.2. Elección de Motores

Se pensó en algunos tipos de actuadores, buscando una buena relación costo/beneficio por lo que el uso de servomotores fue la elección, en este caso el modelo elegido es el Futaba S3003 como se puede visualizar posteriormente, en la **Fig. 12**, con rapidez de acción, de buena calidad y además muy preciso. Una característica importante de este actuador es que incluso mantiene su posición así se le ejerza una fuerza externa en su eje, debido a su circuito electrónico que reajusta el ángulo que se encuentre al asignado.

Fig. 12 Servomotor modelo [20].

3.3. Elección de Cámara

La cámara web utilizada es la ELP-USBFHD05MT-L100 que se puede observar a continuación en la **Fig. 13**. Tiene una resolución máxima de 1080p, captura imágenes a 30 fps, tiene lente sin autoenfoque, con sensor CMOS y ángulo de visión de 100º. Utiliza un driver libre, lo que hace que sea sencillo conectarla a cualquier sistema operativo, contando con soporte plug&play.

Por valor de fábrica tiene 480px. por 640px. de resolución, pero se hizo un cambio para que entregue una resolución de 480px. por 480px. Puesto que la cámara está ubicada exactamente encima del tablero, el centro del tablero corresponde al centro de la imagen, es decir el centro de la placa está en la posición (240 px por 240 px).

Fig. 13 Webcam USB 2MP 1080p Full HD [21].

3.4. Estructura del Módulo

Se utilizó como guía algunas plantillas de proyectos Ball & Plate encontrados de investigaciones previas para aprovechar información como la ubicación de los motores y la longitud de los eslabones. Las piezas se replican mediante tecnología CNC e impresión 3D creando los componentes relevantes como soporte circular, brazos extensores, soporte de servomotores, base de la cámara y soporte de la cámara.

Otros componentes fueron adquiridos de piezas existentes en el mercado por su conveniencia frente a costos y tiempos de armado a comparación con la impresión 3D, dichos componentes fueron usados en la elaboración para la base de la cámara.

3.4.1. Soporte Circular

El soporte circular en el que se desliza la esfera se encuentra reposando en los brazos principales de los servos, dicho soporte se hace cortando por láser la madera MDF de 3mm, y cuyas medidas se muestran en la **Fig. 14.**

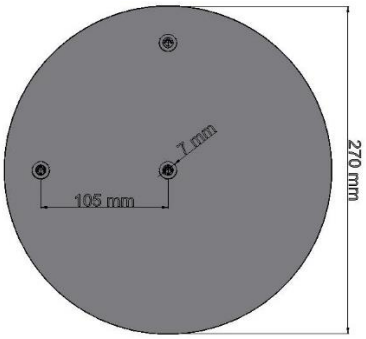


Fig. 14 Gráfico de la plataforma. Autores.

También se acoplaron tres imanes, uno en el centro de la estructura, y dos distanciados a 105mm desde el centro y a 90 grados, que permiten a las esferas metálicas deslizarse libremente mientras la planta lo requiera, a la vez que la sostiene mediante efecto de atracción que ejercen las esferas ferromagnéticas.

3.4.1. Brazos extensores

La conexión entre la plataforma y los servomotores se hace a través de un antebrazo circular cortado con láser, con las medidas necesarias (ver **Fig. 15**), para ser acoplado al servo y al brazo principal el cual es un tubo hueco de fibra.

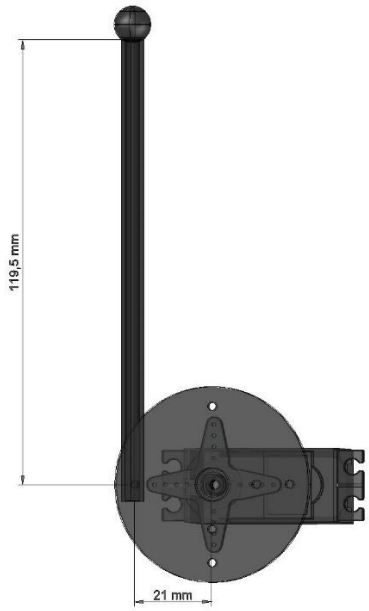


Fig. 15 Vista de brazo en 3D. Autores.

3.4.1. Soporte de los Servomotores

Los motores se ubican a 90 grados cada uno y van atornillados a la estructura haciendo sencillo su ajuste, conexión y reemplazo de ser necesario. En la **Fig. 16** se puede observar la ubicación de los servomotores sobre la plataforma.

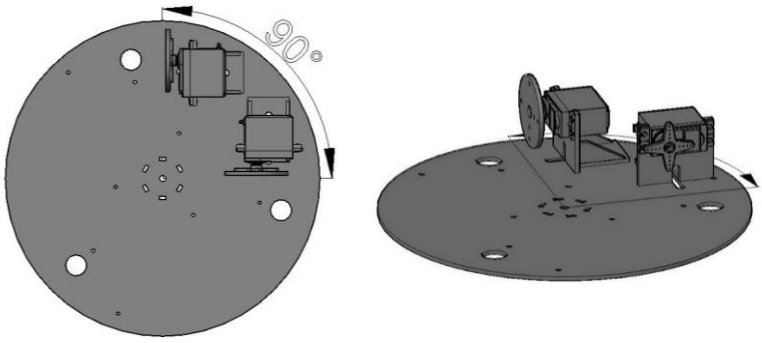


Fig. 16 Ubicación de los servomotores en la base sobre la plataforma. Autores

3.4.1. Base de la Cámara

Se usa un diseño existente para proteger y ubicar la cámara sobre el soporte, se puede observar a continuación en la **Fig. 17** su diseño en 3D.

Fig. 17 Soporte para cámara modelo [22].

3.4.1. Soporte de la Cámara

El soporte de la cámara está diseñado a partir de uniones de presión tipo codo y tipo T de PVC junto con tubería del mismo material, dichos componentes se encuentran con medidas de ½”, ¾”, 1” entre otras, en este caso fueron seleccionados materiales de ½” ya que cumplen el requisito de tener espacio suficiente para que traspase el cable USB que transmite datos de la cámara tal como lo muestra la **Fig. 18**.

Fig. 18 Representación en 3D del soporte para la cámara. Autores.

3.5. Componentes Electrónicos

3.5.1. Tarjeta PCB

El diseño se hizo en el programa KiCad, siendo este programa de uso libre. La tarjeta consta con conexiones tipo bornera para las entradas de voltaje y para conexión de los LED piloto, como también, espadines para que se conecten los servomotores.

Materiales:

- Chip ATMEGA328PU.
- Reloj 16Mhz.
- 2 Condesnadores de 100nF.
- 1 Condensador de 0,1uF 50V.
- 1 Condensador 10uF 50V.
- 1 Resistencia de 220 ohms.
- 1 Regulador de voltaje LM2596.
- 5 borneras.

En la **Fig. 19** se puede observar el diseño esquemático de la tarjeta con las conexiones de cada uno de sus componentes.

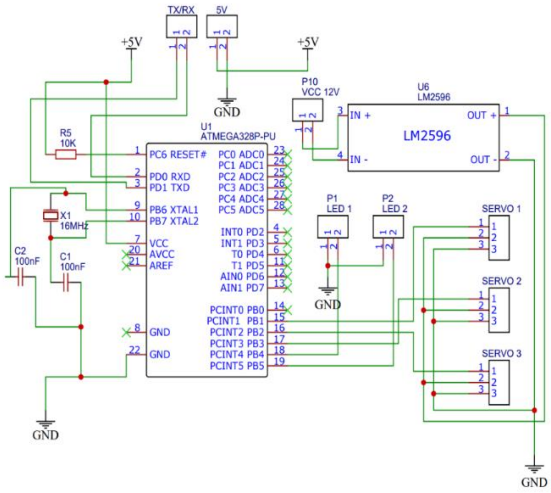


Fig. 19 Diseño esquemático de tarjeta PCB. Autores.

A continuación, en la **Fig. 20** Se observa en una visión 3D como queda la tarjeta antes de su fabricación, mientras que en la **Fig. 21** se muestra la tarjeta ensamblada.

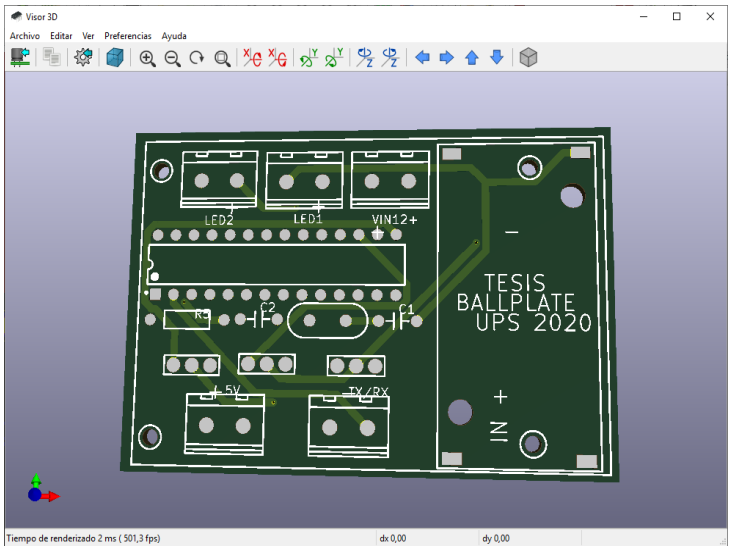


Fig. 20 Visualización del Diseño en 3d en Kicad de tarjeta PCB.Autores.

Fig. 21 Tarjeta electrónica impresa con sus componentes(superior) y la tarjeta previa a soldar sus componentes(inferior). Autores.

3.5.2. Microcontrolador

El microcontrolador usado es la ATMEGA328PU, haciendo mucho más sencillo el mantenimiento o cambio al encontrarse en encapsulado DIP (Dual Inline Package), el mismo fue programado a través del entorno de desarrollo Arduino, donde este toma los valores que llegan en grados desde el puerto serial, y los convierte a grados en los servomotores, además de también controlar las luces piloto indicadoras.

Fig. 22 Chip microcontrolador Atmega 328P [23].

3.5.3. Adaptador Serial

En la tarjeta PCB diseñada hay un puerto Rs232 donde se conecta un adaptador USB-Serial, que sirve como transporte de datos en tiempo real desde la computadora hasta la planta. Además, desde el adaptador se alimenta la tarjeta microcontroladora con su conexión de 5 voltios. En la **Fig. 23** Se muestra imagen del adaptador usado.

Fig. 23 Adaptador USB serial Rs232 para tarjeta ESP-01 [21].

3.5.4. Regulador de Voltaje

Para poder controlar los servomotores es necesario tener una buena fuente de voltaje estable, con capacidad para entregar suficiente corriente tanto para el movimiento abrupto de los servomotores como para situaciones de alta demanda de torque. En este caso se escogió la tarjeta modular basada en el integrado LM2596 tipo Step-Down, que reduce el voltaje DC de entrada a un voltaje variable a través de un potenciómetro con un máximo de corriente de salida de 3 amperios.

En la **Fig. 24** se muestra el módulo regulador usado.

Fig. 24 Módulo Step-Down Basado en el integrado LM2596 [21].

3.6. Desarrollo del Programa

Para desarrollar la planta, se unieron algunas soluciones en programación. Primero se hizo una conexión serial a través de Python, buscando el programa los dispositivos conectados, escogiendo a aquel que tenga el nombre indicado. En este caso "USB Serial Device". Luego de que se pudo verificar la tarjeta microcontroladora, se implementó el seguimiento de objetos por intermedio la librería Opencv, donde por medio de la adquisición de vídeo mediante una webcam, en tiempo real esta adquiere imágenes a 30 frames por segundo, analizando cada una de ellas. Después se cambian los valores RGB a HSV (Hue, Saturation, Vue) para poder determinar los colores específicos a seguir, para tomar en cuenta los elementos dentro de la imagen con un radio mayor a 10 pixeles, haciendo que solo tome la esfera como referencia.

Una vez detectada, determinar en qué ubicación se encuentra referente al plano, para así dirigirse hacia el set point requerido. Las diferentes pantallas se

implementaron para tener un orden jerárquico en el uso del programa, donde al presionar el botón de la cámara, esta sea mostrada, y luego observar en gráficas si se consigue el comportamiento deseado.

3.7. Modelado Matemático

El sistema se desacopla separando los ejes X y Y en sus respectivas direcciones, al tener ambos ejes el mismo comportamiento, el análisis matemático se basa en uno solo. En la **Fig. 25** se observa el diagrama de la dinámica del sistema.

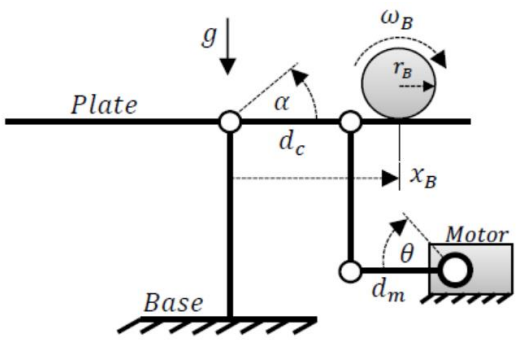


Fig. 25 Dinámica del sistema, vista longitudinal de un eje [24].

Si bien las ecuaciones del movimiento se pueden obtener usando el aproximamiento clásico mediante un diagrama de cuerpo libre, para evitar complejidad en este estudio se usó la formulación de Lagrange:

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = Qi \qquad (1)$$

Donde *L* es la transformación de Lagrange, Qi (1) son las fuerzas externas que actúan sobre el sistema, en este caso q_i será el eje de la coordenada *x*.

La fórmula de Lagrange L puede ser escrita como:

$$L(\dot{q}_i, q_i) = T(\dot{q}_i, q_i) - V\ (q_i) \qquad (2)$$

Donde *T* es la energía cinética total y *V* la energía potencial, ambas en función del eje estudiado, en este caso el *eje X*. Luego se separa *T* en su componente lineal T_l y también su componente rotacional T_r:

$$T_l = \frac{1}{2} m_b\, \dot{x}_b{}^2 \qquad (3)$$

$$T_r = \frac{1}{2} I \omega_b{}^2 \qquad (4)$$

Donde m_b es la masa de la esfera, x_b es su desplazamiento, I es el momento de inercia y ω_b es su velocidad angular. I depende de la forma de la esfera donde $I = \frac{2}{5} m_b\, r_b{}^2$ en el caso de una esfera sólida y $I = \frac{2}{3} m_b\, r_b{}^2$ en el caso de una esfera hueca.

Reemplazando ω_b con $\frac{\dot{x}_b}{r_b}$, el resultado de la energía cinética es:

$$T = \frac{7}{10} m_b\, \dot{x}_b{}^2 \qquad (5)$$

Esfera sólida

$$T = \frac{5}{6} m_b \, \dot{x}_b{}^2 \qquad (6)$$

Esfera hueca

Para la energía potencial:

$$V = m_b \, g x_b \; sin(\alpha) \qquad (7)$$

Donde g es la aceleración gravitacional y $x_b \; sin(\alpha)$ es la fuerza gravitacional (peso) en la componente x.

Volviendo a la resolución de la ecuación de Lagrange (1), la única fuerza actuando en q_i es la fricción F_x que puede ser escrita como $Q_i \; = F_x = -f_c \, \dot{x}_b$

$$\frac{7}{5} \ddot{x}_b + \frac{f_c \, \dot{x}_b}{m_b} + g \sin \alpha = 0 \qquad (8)$$

Ecuación de movimiento de esfera sólida

$$\frac{5}{3} \ddot{x}_b + \frac{f_c \, \dot{x}_b}{m_b} + g \sin \alpha = 0 \qquad (9)$$

Ecuación de movimiento de esfera hueca

La ecuación despejada no será lineal (incluye un término trigonométrico). Por lo tanto, se linealizará sobre el punto de equilibrio.

Hay que tomar en cuenta que la entrada del nuestro sistema no es el ángulo α, sino el ángulo del servo θ_r. Una relación de segundo orden se asume para la dinámica del servomotor:

$$\ddot{\theta} + 2\zeta\omega_n\dot{\theta} + \omega_n{}^2\theta = \omega_n{}^2\theta_r \qquad (10)$$

Donde ω_n es la frecuencia natural del motor, ζ es el radio de amortiguación y ω_r es el ángulo de referencia del servomotor.

Para conectar la ecuación del servomotor con la ecuación de movimiento de la esfera, se obtiene la siguiente relación entre el ángulo α y θ:

$$d_m\, sin\theta = d_c\, sin\alpha \qquad (11)$$

Donde d_m es la distancia desde el brazo del servo hasta la unión, y d_c es la distancia entre el centro de la plataforma y el brazo. Se reduce usando $K = \frac{d_c}{d_m}$.

$$sin\theta = ksin\alpha \qquad (12)$$

<u>Remarcando:</u> La relación entre α y θ se obtiene asumiendo que la unión entre el servo y el soporte de la plataforma es casi vertical, siendo el ángulo ψ = 0. Este no es el casi en la vida real, pero para el análisis cinemático se asumirá de esta manera.

Observando la **Fig. 276** la estructura es un mecanismo de 4 barras. La ecuación se obtiene donde la distancia B entre los ejes a y b son constantes en los movimientos de la estructura. En la **Fig. 27** se grafica en sus componentes.

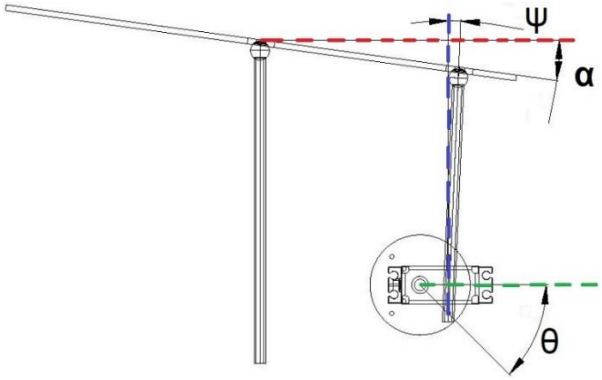


Fig. 26 Boceto de relación angular entre soportes. Autores.

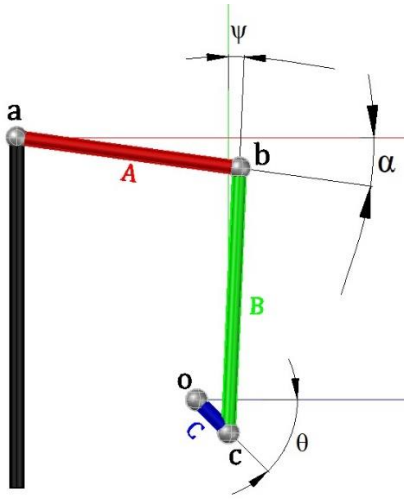


Fig. 27 Boceto de relaciones entre soportes. Autores.

Los símbolos A, B y C se usan como distancias de las uniones, y se usarán luego en la matriz de estados de espacios. La ecuación de relación es la siguiente:

$$\overrightarrow{oc} + \overrightarrow{cb} = \overrightarrow{oa} + \overrightarrow{ab} \tag{13}$$

Proyectando la ecuación al plano horizontal y vertical queda lo siguiente:

$$\begin{bmatrix} -C.sin\theta \\ -C.cos\theta \end{bmatrix} + \begin{bmatrix} B.cos\,\psi \\ -B.sin\,\psi \end{bmatrix} = \begin{bmatrix} B \\ A + C \end{bmatrix} + \begin{bmatrix} A.sin\alpha \\ A.cos\alpha \end{bmatrix} \tag{14}$$

Se usa la librería Simscape en Simulink , construyendo la estructura geométrica y graficando la relación numérica entre α y θ tal como lo muestra la **Fig. 28**, mientras en la **Fig. 29** se visualiza la gráfica en 3D de la planta.

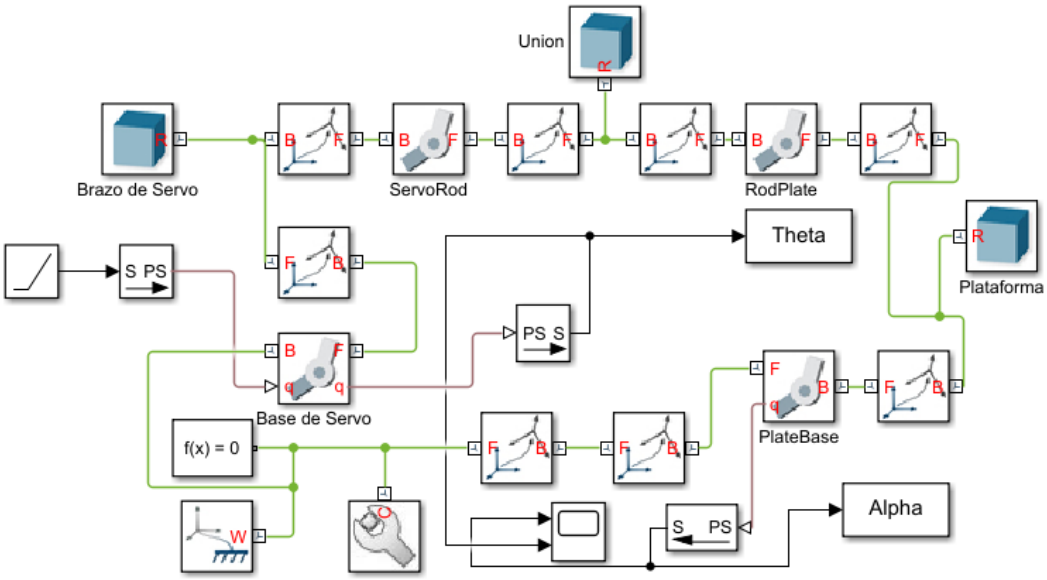


Fig. 28 Diagramas de bloque en librería SimScape, Simulink. Autores.

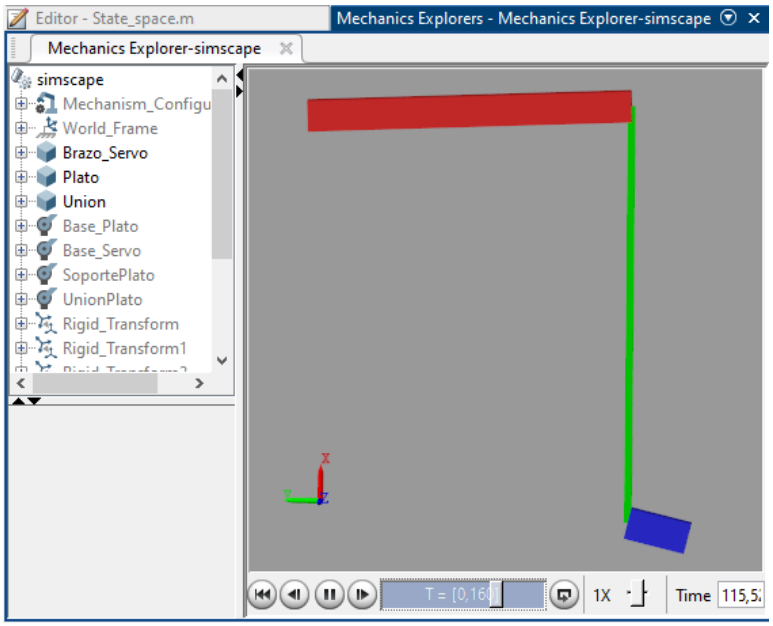


Fig. 29 Animación del movimiento de la planta usando Simscape. Autores.

Luego de obtener la solución analítica para la relación entre α y θ, una comparación con la aproximación de la ecuación 11 se muestra en la **Fig. 30**.

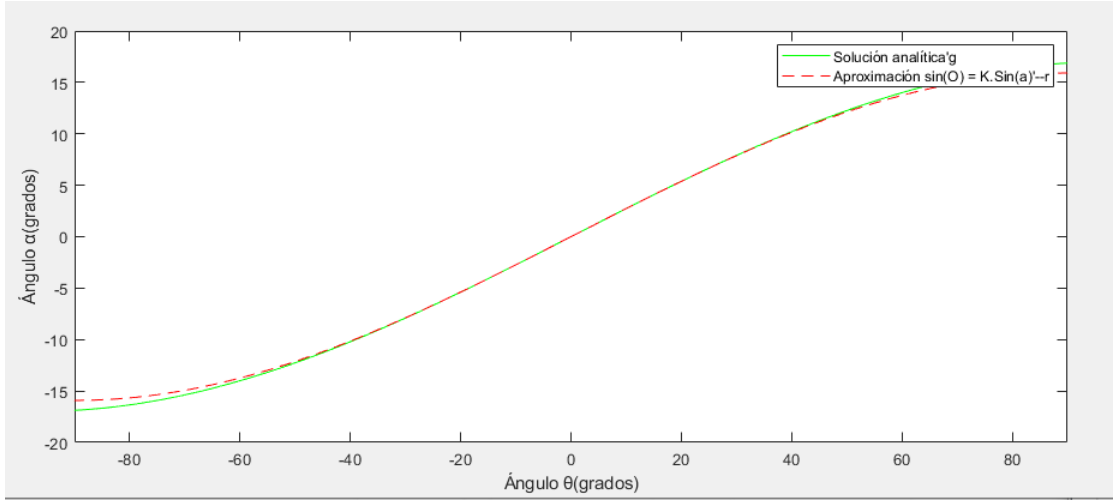


Fig. 30 Comparación entre la solución analítica y la relación del eje vertical. Autores.

Se comprueba que la fórmula $d_m sin\,\theta = d_c sin\,\alpha$ es una aproximación bastante acertada de la relación entre α y θ, tomándola así para el desarrollo de este estudio.

Los términos trigonométricos se linealizan sobre el estado de equilibrio para poder realizar el diseño de control, sin embargo, se puede hacer una tabla de mapeo con la relación entre los ángulos α y θ usando la planta real. El ángulo α se diseña para que tenga un alcance entre -15° y $+15^{\circ}$, lo que significa que $sin\ \alpha$ se puede linealizar sobre el punto de equilibrio $0°$ logrando aproximar $sin\ \alpha \approx \alpha$. La misma linealización sucede con el ángulo θ sobre $0°$ donde $sin\ \theta \approx \theta$, esta relación solo se mantiene entre -15° y $+15^{\circ}$, estas y otras linealizaciones deben hacerse para cubrir todos los ángulos que opera el servo para un modelado más preciso.

$$\theta = k\alpha \tag{15}$$

La siguiente fórmula representa la ecuación que se usó para conectar la entrada del sistema α_r a la salida del sistema x_b.

$$\ddot{\alpha} = 2\zeta\omega_n\dot{\alpha} + \omega_n^2\alpha = \omega_n^2\alpha_r \tag{16}$$

<u>Remarcando:</u> Las ecuaciones de movimiento de la plataforma copiadas de la dinámica del servomotor se agregaron sin tomar en cuenta alguna fricción de las uniones o vibraciones por la rigidez de las varillas.

Ahora que las ecuaciones fundamentales y las relaciones se encontraron, el espacio de estados del sistema de linealización continua invariante del tiempo se puede construir usando la siguiente representación general:

$$\dot{q}(t) = Aq(t) + B.u(t) \tag{17}$$

$$y(t) = Cq(t) + D.u(t) \tag{18}$$

Donde $q = [x_b \;\; \dot{x}_b \;\; \alpha \;\; \dot{\alpha}]^T$ son los estados del sistema y $u = \alpha_r$ es la entrada del sistema, y $y = x_b$ es la salida, las matrices A, B, C y D se muestran a continuación:

$$\begin{bmatrix} \dot{x_b} \\ \ddot{x}_b \\ \dot{\alpha} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & \frac{-5}{7}\frac{f_c}{m_b} & \frac{-5}{7}g & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\omega_n^2 & -2\zeta\omega_n \end{bmatrix} \cdot \begin{bmatrix} x_b \\ \dot{x}_b \\ \alpha \\ \dot{\alpha} \end{bmatrix} + [0 \quad 0 \quad 0 \quad \omega_n^2] \cdot [\alpha_r] \tag{19}$$

$$[x_r] = [1 \quad 0 \quad 0 \quad 0] \,.\, \begin{bmatrix} x_b \\ \dot{x_b} \\ \alpha \\ \dot{\alpha} \end{bmatrix} = [0].[\alpha_r] \tag{20}$$

3.8. Planteamiento de Parámetros del Sistema

El propósito de esta sección es validar los parámetros del sistema y no identificarlo solamente como una caja negra. Es mayormente una aproximación como una caja gris donde se conoce el modelado teórico del sistema y se notan las diferencias con la planta real. Los parámetros mostrados en la **TABLA I,** que se

encuentra a continuación, serán medidos, a excepción de ω_n y ζ donde se usan las especificaciones del fabricante del servo motor. La **TABLA II** muestra las distintas configuraciones de esferas utilizadas en la experimentación.

TABLA I
PARÁMETROS DEL SISTEMA

Símbolo	Valor
ω_n	$5.5116 rad/s$
ζ	1
f_c	$0.001 N_s/m$
g	$9.81 m/s^2$
d_m	$0.025 m$
d_c	$0.150 m$

TABLA II
CONFIGURACIONES DE LA ESFERA [25].

Configuración	Descripción	r_b(m)	m_b(kg)
1	Hueca metal	0.016	0.013
2	Hierro sólido	0.01	0.046
3	Ping Pong	0.02	0.001
4	Esfera sólida	0.007	0.0049

<u>Recordatorio</u>: Un estado de equilibrio con respecto a la entrada constante $\bar{u}$ es una configuración x_e del sistema donde:

$\bar{f}(x_e, \bar{u}) = 0$ en tiempo continuo

$\bar{f}(x_e, \bar{u}) = x_e$ en tiempo discreto

Donde:

$\dot{x} = \bar{f}(x, \bar{u})$ en tiempo continuo

$x(t+1) = \bar{f}(x, u)$ en tiempo discreto

Para determinar los estados de equilibrio en nuestro sistema:

$$\begin{bmatrix} \dot{x_b} \\ \ddot{x_b} \\ \dot{\alpha} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{21}$$

$$\begin{bmatrix} \dot{x_b} \\ \frac{5}{7}(\frac{-f_c}{m_b}\dot{x_b} - gsin\alpha) \\ \dot{\alpha} \\ -2\zeta\omega_n\dot{\alpha} - \omega_n^2\alpha \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{22}$$

Resolviendo la igualdad, se encuentra que existen infinitos estados de equilibro donde x_b puede ser cualquier número real mientras todos los otros estados son iguales a cero. En otras palabras, $\bar{x}$ es un estado de equilibrio si lo siguiente es verdad $(x_b \in \mathbb{R}^n) \cap (\dot{x_b} = 0) \cap (\alpha = 0) \cap (\dot{\alpha} = 0)$

<u>Remarcando</u>: La resolución anterior significa que la plataforma se encuentra en equilibrio mientras lo velocidad de la esfera, el ángulo de la plataforma, y su

velocidad angular son cero. Aunque en realidad esta tiene límites donde ese estado sería válido.

Mientras existan infinitos puntos de equilibrio, es imposible que todos estos sean factibles, y para su factibilidad local, sabemos que un espacio vacío de x_e que converge a x_e no existe. Por lo tanto, ningún punto de equilibrio es localmente factible.

3.9. Análisis de Estabilidad

Basado en la noción de estabilidad donde el orden para que un estado de equilibrio sea estable, cualquier esfera de radio $\in > 0$ se debe confinar a toda la trayectoria de estados. Podemos encontrar todos los estados iniciales alrededor del equilibrio donde la trayectoria no vaya fuera del radio $\delta > 0$.

En la definición BIBO (bounded input, bounded output) de estabilidad, si un sistema tiene una salida limitada para una entrada limitada. entonces es estable para BIBO. En nuestro sistema, cualquier entrada limitada del ángulo α causara que x_b vaya al infinito, haciéndolo inestable. Esto se puede comprobar observando la función de transferencia obtenida de la transformación de Laplace.

Reemplazando todos los parámetros del estado de espacio con sus números estables y escogiendo la cuarta configuración de la esfera, obtenemos:

$$\begin{bmatrix} \dot{x_b} \\ \ddot{x}_b \\ \dot{\alpha} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0.0893 & 7.0071 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -30.37 & -11.1191 \end{bmatrix} \cdot \begin{bmatrix} x_b \\ \dot{x}_b \\ \alpha \\ \dot{\alpha} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 30.3777 \end{bmatrix} \cdot [\alpha_r] \tag{23}$$

Como un sistema invariante en el tiempo, su estabilidad depende del vector característico de la matriz A:

$$eig(A) = \begin{bmatrix} 0 \\ -0.0893 \\ -4.8309 \\ -6.2882 \end{bmatrix} \tag{24}$$

Al eigenvector tener valores negativos, se llega a la conclusión de que el sistema es inestable, para posteriormente verificar si se lo puede estabilizar. Para lograr esto se debe chequear si el sistema es controlable o no.

3.10. Análisis de Controlabilidad

Como se definió previamente, la matriz de controlabilidad correspondiente al sistema es:

$$R = \begin{bmatrix} B & AB & A^2B & A^3B \end{bmatrix}$$

$$R = \begin{bmatrix} 0 & 0 & 0 & 2130 \\ 0 & 0 & 2130 & -2386 \\ 0 & 30 & -338 & 2833 \\ 30 & -338 & 2833 & -21239 \end{bmatrix}$$

La matriz de controlabilidad se encuentra en rango $(rank(R) = 4)$, lo que significa que el sistema es controlable, donde teóricamente cualquier valor característico puede arbitrariamente alcanzar el valor deseado.

3.11. Análisis de Observabilidad

Para determinar si el sistema es observable, se construye la matriz de observabilidad:

$$O = \begin{bmatrix} C \\ AC \\ A^2C \\ A^3C \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -0.0893 & 7.0071 & 0 \\ 0 & 0.0080 & -0.6256 & 7.0071 \end{bmatrix}$$

El sistema es observable siendo que la matriz O tiene rango completo $(rango(O) - 4)$. Esto significa que podemos estimar todos los estados de las medidas de desplazamiento.

Tener en cuenta que, si se escoge medir exclusivamente la velocidad de la bola, el ángulo de la plataforma o la velocidad angular de la plataforma, la nueva matriz C no tendrá un rango completo haciéndola no observable (esto prueba que escogiendo solo un sensor para medir el desplazamiento es suficiente).

3.12. Requerimientos de Control

El sistema necesita ser estabilizado, se requiere conocer los requerimientos de control. Una lista de requerimiento razonable a la respuesta de entrada tipo paso es:

- Tiempo de estabilización: $4\%\ t_s \leq 8s$
- Error en estado de equilibrio $\leq 5mm$
- Ganancia de margen $\geq 10db$

3.13. Diseño del Controlador y Simulación

Para diseñar el sistema de control se utilizó Simulink, y se encontró la siguiente función de transferencia:

$$\frac{Y(S)}{P(S)} = \frac{212}{S^4 + 11{,}26s^3 + 31{,}97s^2 + 4{,}34s}$$

Se aplicó la entrada paso a la función de transferencia que obtuvimos al hacer la transformación desde el sistema de espacio de estados. Se analizó la respuesta del sistema en lazo abierto, observando su comportamiento. El diagrama de bloques se muestra a continuación en la **Fig. 31**.

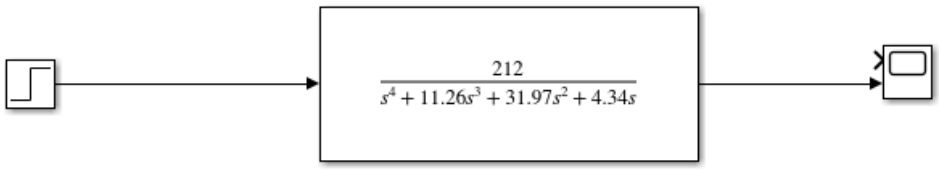


Fig. 31 Diagrama de lazo cerrado abierto. Autores.

Se aprecia la función sin estabilización como lo observado en la **Fig. 32**.

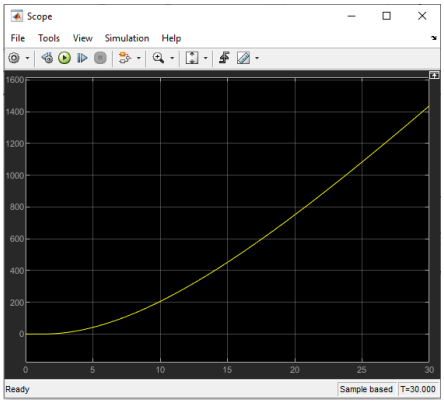


Fig. 32 Localización de raíces sin controlador. Autores.

Luego de esa verificación se hace la prueba con el sistema en lazo cerrado con un controlador PD para hacer la configuración respectiva y obtener los factores P y D para un control efectivo.

Un bloque ZOH (Zero order hold) agrega el modelado del tiempo de muestreo del sensor(La cámara captura a 30 FPS(Frames per second), que significa un tiempo de muestreo de 0.033s).

Se agregan los bloques de control PID y el sumador para analizar el sistema retroalimentado tal como se muestra en la **Fig. 33**.

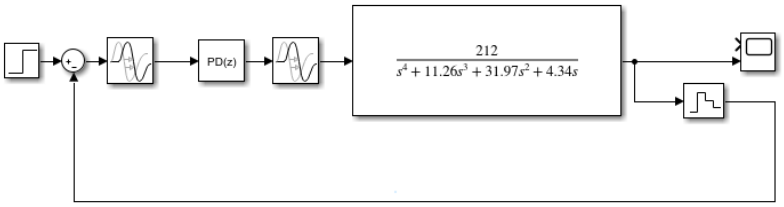


Fig. 33 Diagrama en lazo cerrado. Autores.

Al agregar el controlador PD, dos bloques de retraso son usados para simular el retardo del procesamiento de la imagen como también por la comunicación de la computadora con la tarjeta microcontroladora.

Se abre el bloque de PID, donde se asigna el tipo de control PD, procediendo al autotune, para obtener las recomendaciones del Simulink con valores efectivos para el sistema, tal como lo detalla la **Fig. 34**.

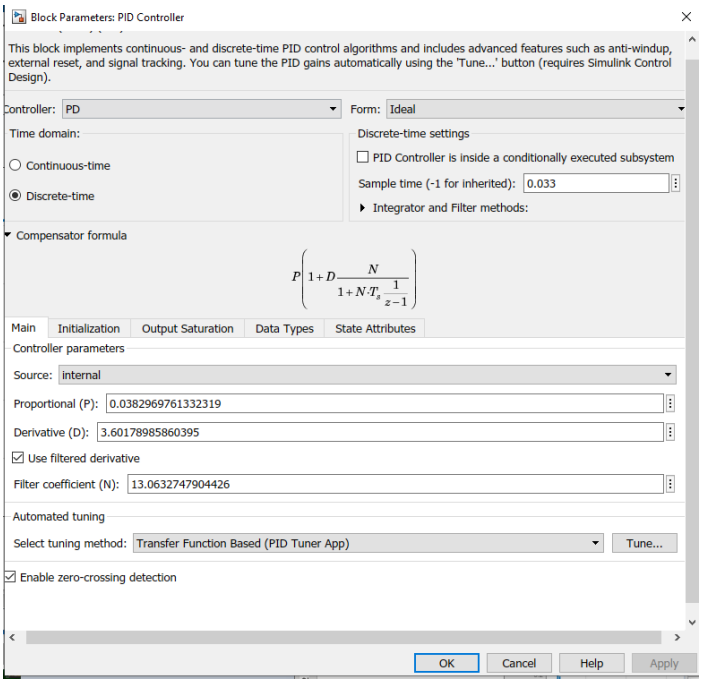


Fig. 34 Diagrama en lazo cerrado. Autores.

Una vez abierto el sistema de estabilización automática se asignan valores de respuesta en tiempo hasta llegar a parámetros requeridos tal como se muestra en la **Fig. 35**.

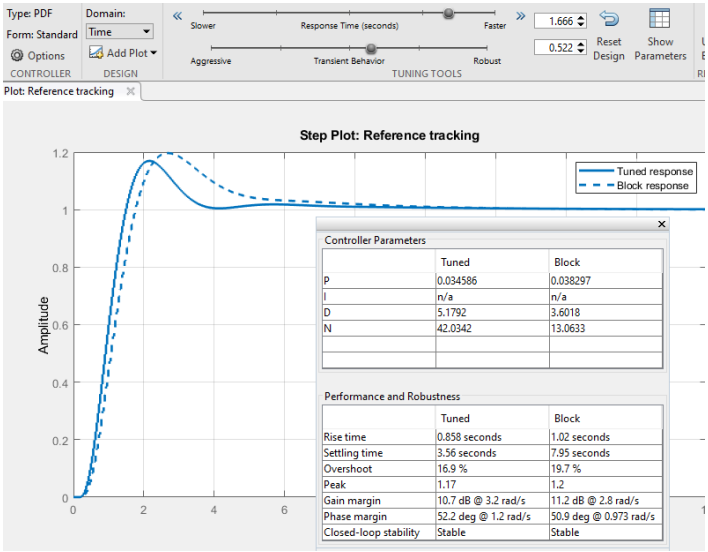


Fig. 35 Asignación de parámetros PD a través de autotuning. Autores.

Si el controlador obtenido no tiene la efectividad como se esperaba, es un paso para llegar a una solución. El tiempo de muestreo del controlador discreto puede alterar el comportamiento drásticamente, el sistema se estabiliza, aunque puede seguir alejado de cumplir los requerimientos de control por sus bajos márgenes de estabilización tal como se muestra en la **Fig. 36**.

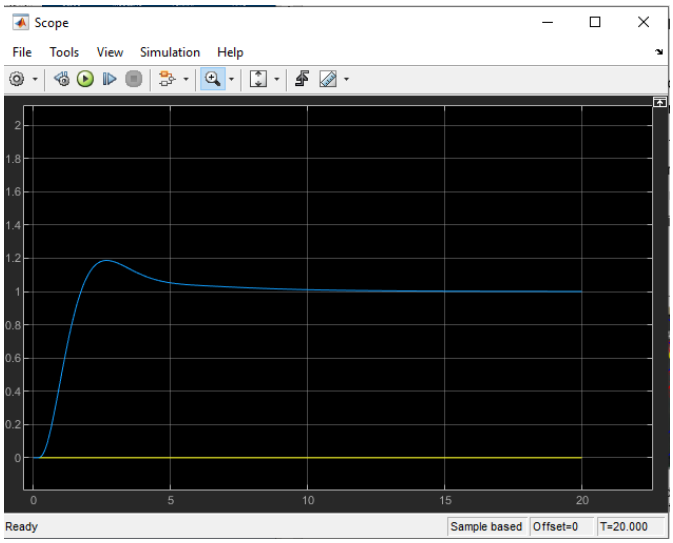


Fig. 36 Gráfica de respuesta de planta en simulación. Autores.

CAPÍTULO IV

RESULTADOS

4.1. Resultados y Comparaciones

Una vez obtenidos los valores, se los prueba en la planta. (P = 0.038, D = 3.60, N=13). Aunque en la planta el rango de valor del factor D llega hasta D= 0.1, donde se analizó desde un valor intermedio (D= 0.05) para llegar a una respuesta idónea.

Con los valores asignados, el sistema no logra estabilizarse. No es posible llegar al setpoint para ambos ejes. Tal como se muestra en la **Fig. 37**.

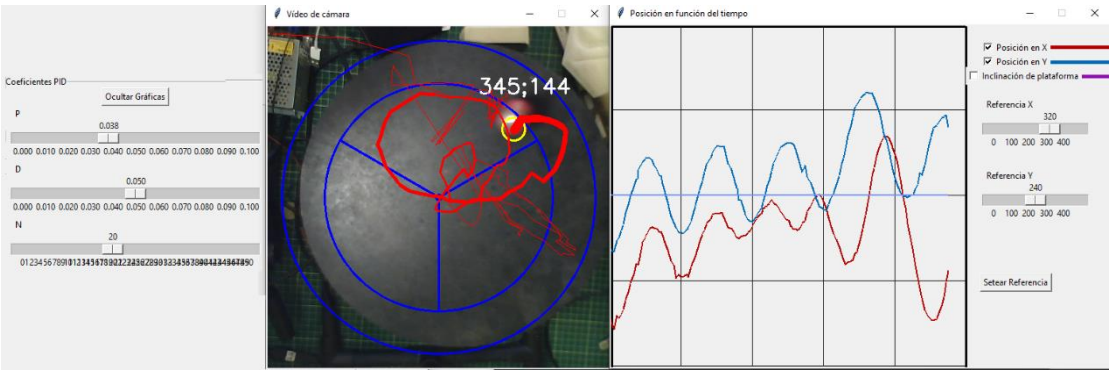


Fig. 37 Respuesta del sistema usando los siguientes valores (P = 0.038, D = 0.05, N=13). Autores.

Con la respuesta obtenida, se hace una calibración fina hasta llegar a un valor aceptable, donde se lleguen a parámetros esperados. Con los siguientes valores (P = 0.038, D = 0.019, N=13). Obteniendo así una estabilización aceptable de la planta.

Utilizando el mismo controlador obtenido en Simulik, la comparación con la planta real que se muestra en la ¡Error! No se encuentra el origen de la referencia.**8** explica que el modelo sigue siendo diferente de la planta real.

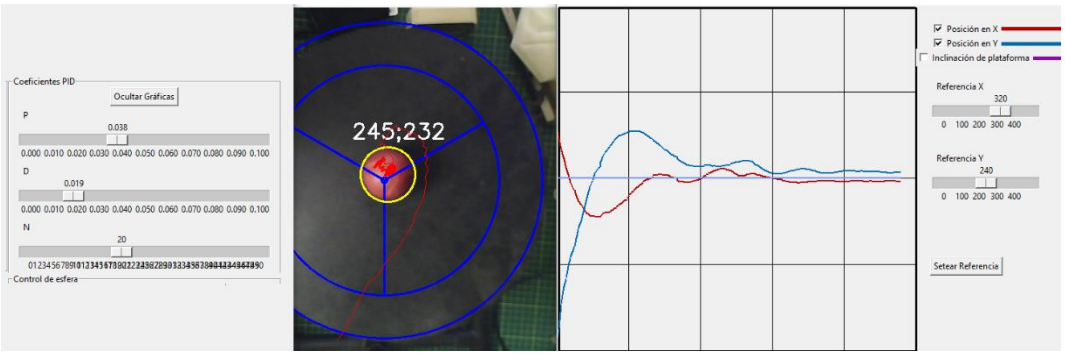


Fig. 38. Respuesta del sistema usando los siguientes valores (P = 0.038, D = 0.019, N=13). Autores.

Esta diferencia puede deberse a muchas razones como: no incluir la fricción estática, vibraciones estructurales, incertidumbres en el tiempo de retraso, variaciones en el ruido, retrasos e imperfecciones en la superficie del plato y bola.

Volver al modelo y aumentar el retardo de tiempo da como resultado un comportamiento oscilatorio similar.

Al estar tan cerca de capturar la dinámica real del modelo, el ajuste manual puede ser una forma más rápida de lograr un rendimiento óptimo. Al ajustar los controles deslizantes PD en la interfaz de usuario, se obtuvo un controlador superior como se muestra en la **Fig. 39**.

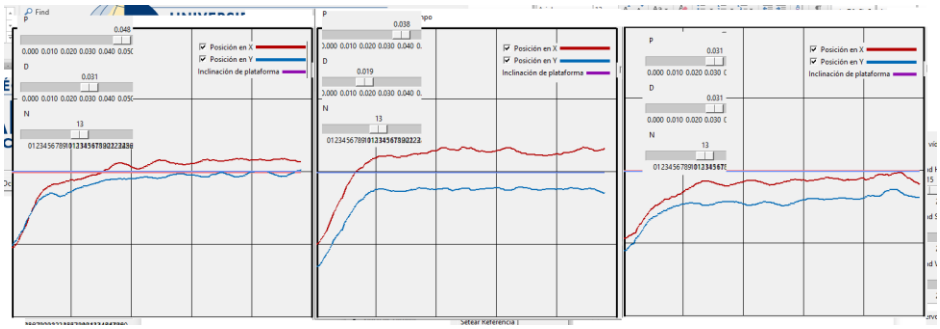


Fig. 39 Comparación del rendimiento de múltiples controladores PD de la planta física.
Autores.

Tener en cuenta que el error de estado estable se puede eliminar mediante una acción integral, pero como el error es pequeño, un controlador PD será suficiente.

La razón del error de estado estable es la fricción estática. Siempre que la bola esté cerca de la referencia deseada, un pequeño ángulo de inclinación de la placa no es suficiente para romper la fricción estática, por lo tanto, la bola se atasca con un error de estado estable.

También por el funcionamiento no tan exacto de los servomotores, genera una pequeña diferencia en distancia desde el origen, con lo cual se compensa ajustando la referencia a través del GUI. Con los botones que se muestran en la **Fig. 40**.

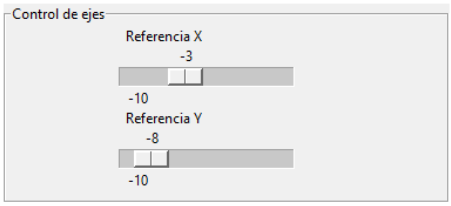


Fig. 40 Cambio de referencia referente a los ejes.
Autores.

La respuesta a una entrada tipo escalón siempre es buena para ver claramente la calidad del desempeño del controlador, sin embargo, la planta debe poder ubicar la esfera en cualquier ubicación deseada. El seguimiento de forma senoidal se muestra en la **Fig. 41**.

Fig. 41 Comparación de una trayectoria senoidal.
Autores.

Se observa un comportamiento esperado. Al utilizar un controlador PD, siempre hay un retraso en el seguimiento de las referencias en movimiento. Sin embargo, existen muchos otros controladores, como el repetitivo y el controlador de resonancia, que son más adecuados para rastrear referencias periódicas con precisión y sin demora.

CONCLUSIONES

De acuerdo con el trabajo presentado, los resultados obtenidos y su discusión, se pueden plantear las siguientes conclusiones:

- El uso de herramientas Open Source permiten un desarrollo de una planta que sea didáctica, de bajo coste y asequible.

- En el momento de hacer el estudio matemático se pueden tomar varias aproximaciones para llegar a un buen diseño del controlador.

- El lenguaje de programación Python es bastante intuitivo, además que tiene bastante respaldo en Internet para poder seguir desarrollando más aditamentos a la planta y hacerla más funcional.

RECOMENDACIONES Y OBSERVACIONES

- Existe algunas formas de abarcar el diseño de la planta y el controlador PID o en este caso PD, tales como Fuzzy, Inteligencia Artifical, entre otros. Todos estos también con capacidad de ser desarrollados en Python.

- Implementar el módulo de prácticas con servomotores originales y certificados, pues los servomotores usados que tenían capacidades similares, pero sin certificación de autenticidad no contaban con protección de sobrecarga en el torque de salida de los servos; lo que dio como resultado varios engranajes internos seriamente dañados inutilizando dichos servomotores al poco tiempo de uso.

- La extensión de brazo que unía el eje del servo al siguiente eje vertical, en un principio estaba diseñada en una extensión rectangular de 3mm de ancho por 5 cm de largo aproximadamente; la cual presentaba deformaciones en los momentos donde mayor fuerza era aplicada para que la esfera se ubique en la posición deseada, por lo que se optó por cambiar esa extensión a una figura circular procurando que los pernos que unen dicha extensión no se interpongan en el movimiento libre de las uniones.

BIBLIOGRAFÍA

[1] K. Ogata, INGENIERIA DE CONTROL MODERNA, Madrid: Ed. Pearson Educación S.A., 5 ed., 2010.

[2] J. Braslavsky, «Controlabilidad y Observabilidad,» de *Control Automático 2 Notas de Clase*, 2001.

[3] R. González, Python Para Todos, España: Auto-edición, 2010.

[4] Python (TM), «https://www.python.org/,» 2020. [En línea]. Available: https://wiki.python.org/moin/TkInter. [Último acceso: 25 08 2020].

[5] A. C. a. Contributors, « Pillow (PIL Fork),» [En línea]. Available: https://pillow.readthedocs.io/en/stable/. [Último acceso: 25 08 2020].

[6] A. Rosebrock, « Python Package Index,» [En línea]. Available: https://pypi.org/project/imutils/. [Último acceso: 25 08 2020].

[7] C. Liechti, «Python Package Index,» [En línea]. Available: https://pypi.org/project/pyserial/. [Último acceso: 25 08 2020].

[8] S. Paguada, Diseño de PCBs usando AutoDesk Eagle y FlatCAM, Honduras: Hybridcores, 2017.

[9] D. H. J. Charras, «KiCAD,» 13 julio 2020. [En línea]. Available: https://kicad-pcb.org/about/kicad/.

[10] J. C. S. Carballas, Creación de un protocolo de comunicación entre un PC y un puerto USB de un microcontrolador y su integración en MATLAB, Coruña, España, 2018.

[11] Grassvalley, «http://grassvalley.com,» 2012. [En línea]. Available: https://www.grassvalley.com/docs/WhitePapers/broadcast/cameras/ldk3000plus/CAM-4073M-ES_CMOS_Whitepaper.pdf.

[12] R. Cobo, «El ABC de la Automatización,» 2017.

[13] J. Ordieres-Meré, «Técnicas y algoritmos básicos de visión artificial,» Madrid, 2006.

[14] OpenCV team, «OpenCV,» [En línea]. Available: https://opencv.org/. [Último acceso: 25 08 2020].

[15] I. Gil, La impresión 3D y sus alcances en la arquitectura, Madrid, 2015.

[16] J. Prusa, «PRUSA RESEARCH,» 2018. [En línea]. Available: https://www.prusa3d.es/. [Último acceso: 25 08 2020].

[17] P. Lambán y A. Domínguez, «https://impresoras3dblog.wordpress.com/,» 2013. [En línea]. Available: https://impresoras3dblog.wordpress.com/.

[18] Sculpteo, «https://www.sculpteo.com/es/,» 2020. [En línea]. Available: https://www.sculpteo.com/es/glosario/corte-por-laser-definicion/.

[19] TROTEC, «https://www.troteclaser.com/es,» 2020. [En línea]. Available: https://www.troteclaser.com/es-ec/faqs/como-cortar-con-laser/.

[20] FUTABA Corporation, «FUTABA Corporation Radio control Web Site,» [En línea]. Available: https://www.rc.futaba.co.jp/english/servo/products/s3003.html. [Último acceso: 25 08 2020].

[21] PC Control, «PC Control,» [En línea]. Available: https://www.pccontrol.ec/. [Último acceso: 25 08 2020].

[22] J. Link, «Système de stabilisation PID,» La Tour-de-Peilz, Switzerland,.

[23] © 2015 Atmel Corporation, ATmega328P 8-bit AVR Microcontroller with 32K Bytes In-System, 2015.

[24] J. V. A. T. a. L. A. R. Da Silvera, «A Comparative Analysis of Repetitive and,» 2014.

[25] J. Alsamarji, Design and Control of a Ball and Plate Didactic Device, 2018-2019.

[26] Google Inc., «Maps de Google,» 2021. [En línea]. Available: maps.google.com.

ANEXOS

Anexo 1: Guía de Prácticas

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**GUIA DE PRÁCTICAS**	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 1**
NOMBRE PRÁCTICA:	Instalación y configuración de Python y Arduino en el computador	

1. **Objetivo**

Instalar y configurar los entornos de programación Python y Arduino en el computador.

2. **Objetivos Específicos**

- Aprender a manejar las bases del lenguaje Python.
- Configurar el entorno de programación Atom.
- Reconocer comandos básicos para verificar correcta instalación y su uso.
- Instalación de entorno de programación Arduino.

3. **Instrucciones**

- Descargar Python.
- Descargar IDE (Entorno de desarrollo Integrado) Atom.
- Instalar Python en el sistema operativo Windows.
- Instalar software Atom y configurarlo para su funcionamiento con Python.
- Instalar librerías necesarias para funcionamiento del prototipo.
- Descargar Arduino.

4. Actividades a desarrollar.

Paso Nº 1:

Se abre el navegador de preferencia, luego se escribe la dirección web https://python.org, se hace click en la sección “Downloads”, tal como se muestra en la **Fig.1** se descarga el programa para Windows es su última versión disponible, en este caso “Python “3.8.3” y se guarda el archivo. Se recomienda hacer antes una carpeta para alojar los archivos de todas las prácticas en una misma carpeta.

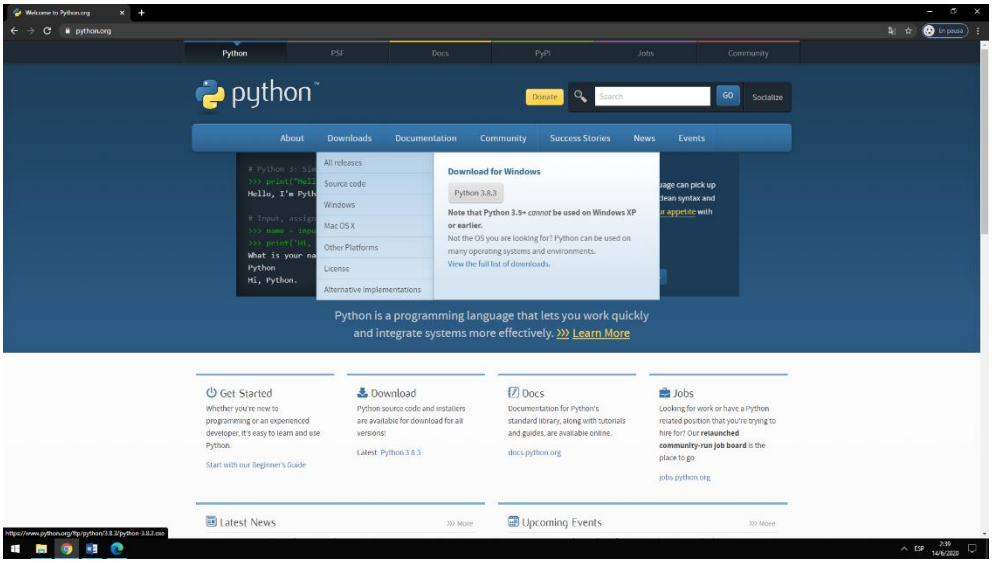


Fig. 1 Página web de Python. Autores.

Paso Nº 2:

Hay que dirigirse a la página https://atom.io/, se da click en el botón “Download” y se descarga el entorno de desarrollo que sirve para un mejor manejo de la programación. A continuación, en la **Fig. 2** se puede visualizar la captura de la página web de Atom.

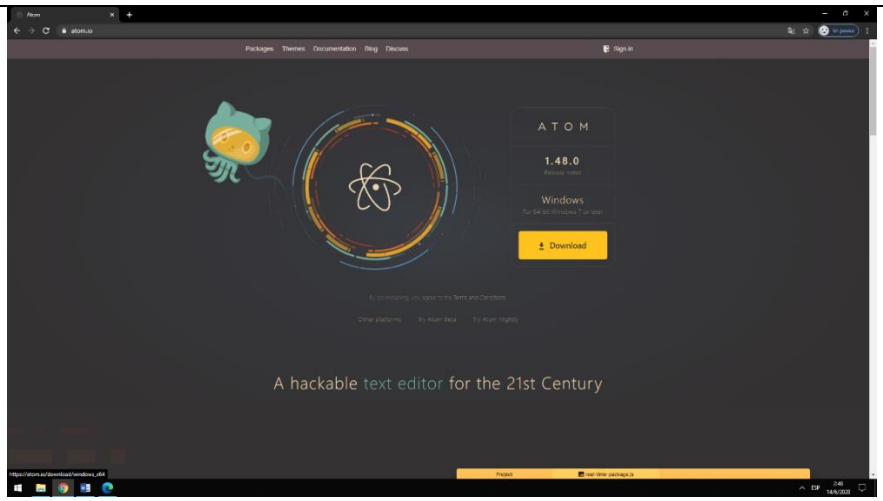


Fig. 2 Página web del programa Atom. Autores.

Paso Nº 3:

Se procede con la instalación de Python. Se inicializa el instalador que se descargó previamente, se selecciona la opción “Add Python 3.8 to Path” para que automáticamente pueda ser abierto desde la consola CMD, y luego se hace click en “Customize installation”. Tal como detalla la **Fig. 3**.

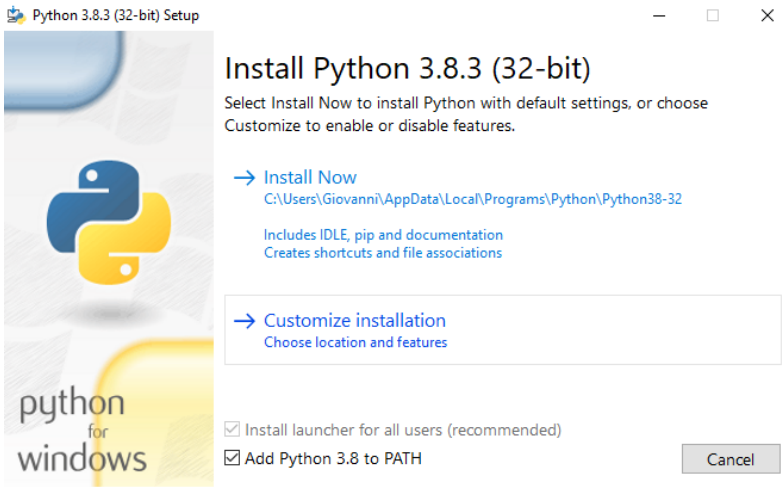


Fig 3. Primera pantalla de instalación de Python en Windows. Autores.

En la siguiente pestaña como se muestra a continuación en la **Fig. 4** se seleccionan todas las opciones y se hace click en “Next”.

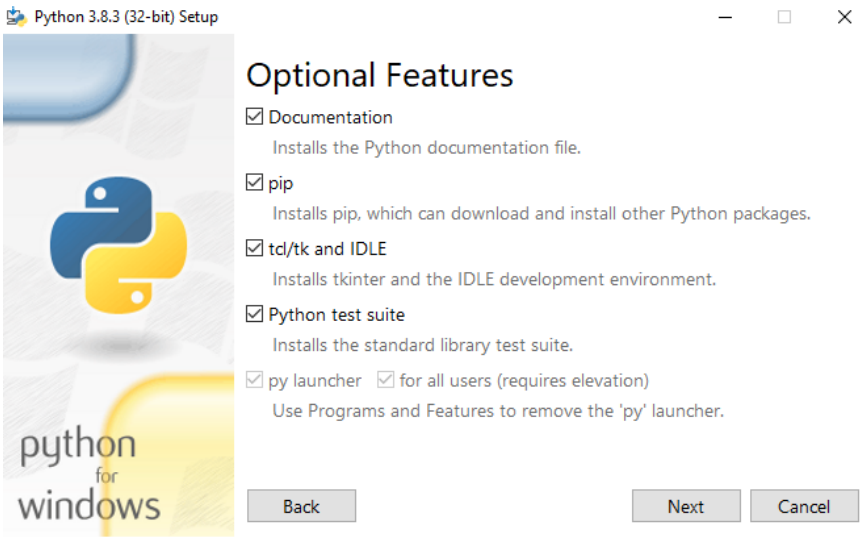


Fig. 4 Segunda pantalla de instalación de Python en Windows. Autores.

Se cambia la carpeta de instalación por la siguiente: “C:\Python\Python38”, y se hace click en el botón “Install”. Tal como se grafica en la **Fig. 5**.

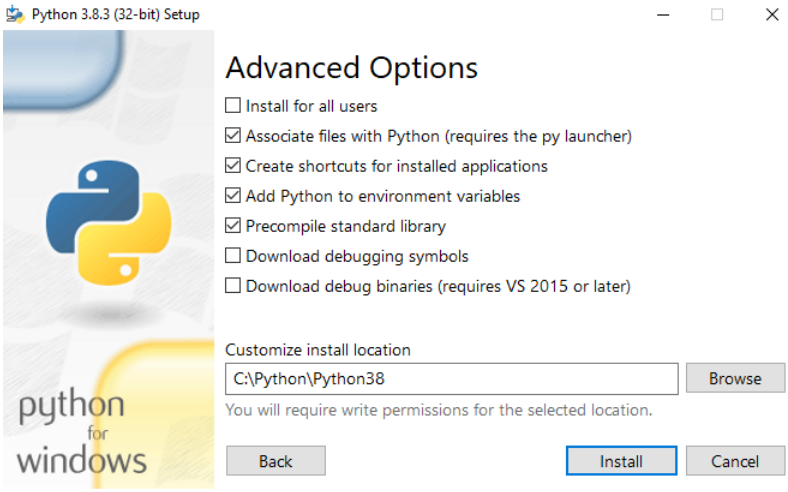


Fig. 5 Tercera pantalla de instalación de Python en Windows. Autores.

Una vez que esté instalado el programa, este muestra una pantalla donde se recomienda seleccionar la opción “Disable path length limit” tal como se grafica en la **Fig.6** para así no tener problemas con archivos que sobrepasen el límite de caracteres dentro de algunas carpetas.

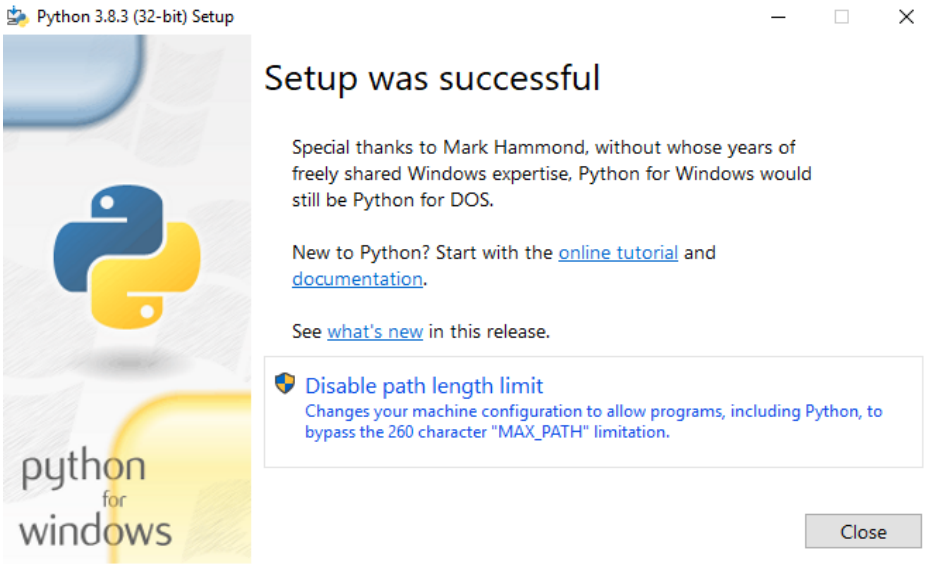


Fig. 6 Cuarta pantalla de instalación de Python en Windows. Autores.

Para verificar una correcta instalación, se revisa su funcionamiento a través de comandos en la consola. Una vez abierto el CMD, si digita el comando “cd” para cambiar la ubicación de lectura, se da espacio y se escribe la dirección donde se instaló el programa, en este caso “C:\Python\Python38”. Una vez ahí en la carpeta se escribe el comando “Python”, se aprieta la tecla “Enter”. Se puede verificar el correcto funcionamiento tal como se aprecia en la **Fig. 7**.

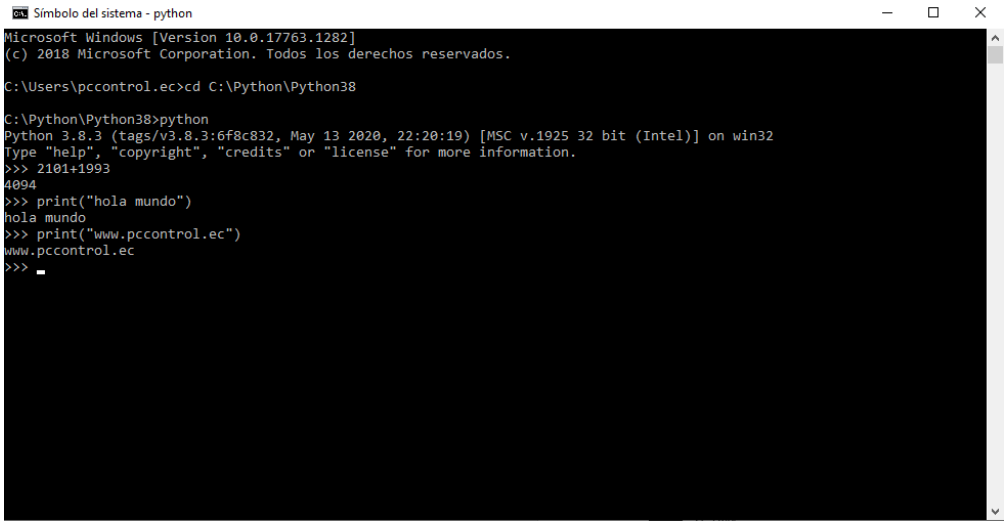


Fig. 7 Lenguaje Python funcionando en CMD en Windows. Autores.

Paso Nº 4:

Instalación de Atom.

Para poder usar el programa, solamente se inicializa el instalador, y se espera hasta que termine de instalarse. Una vez instalado el programa, se hace click en “File”, y luego en “Settings”. Se abre un recuadro con varias opciones donde se escoge el recuadro que dice “Install”. Hay un buscador de paquetes, ahí se escribe “Script”, y se da click en el recuadro para instalar. Se hace el mismo proceso con el paquete “platformio-ide-terminal”.

En la **Fig. 8** se puede apreciar como se ve el proceso de instalación de ambos paquetes de instalación.

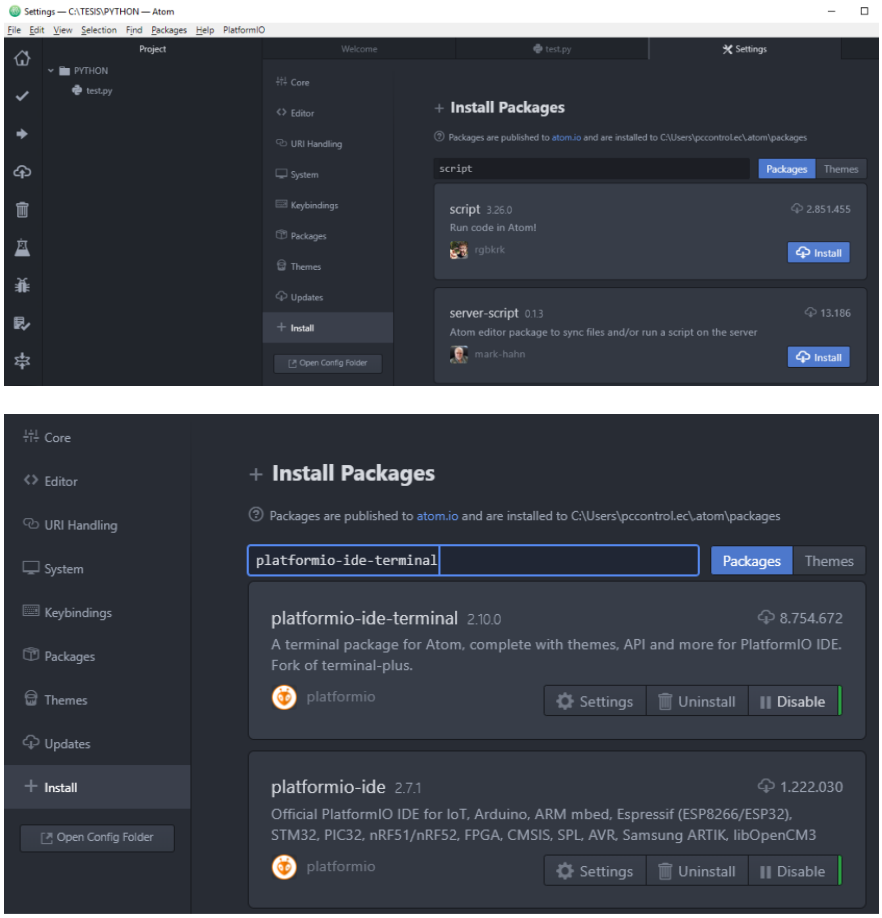


Fig. 8 Pantallas de instalación de paquetes en Atom. Autores.

Una vez instalado la terminal de comandos, como también la gestión de Python dentro de Atom, se prueba con un código de ejemplo para verificar su funcionamiento. Se aplastan las teclas “Alt+Shift+T” y se tiene que abrir una consola de comandos.

Se observa en la siguiente en imagen en la **Fig. 9**. Como debe de observarse en la pantalla de terminal activa.

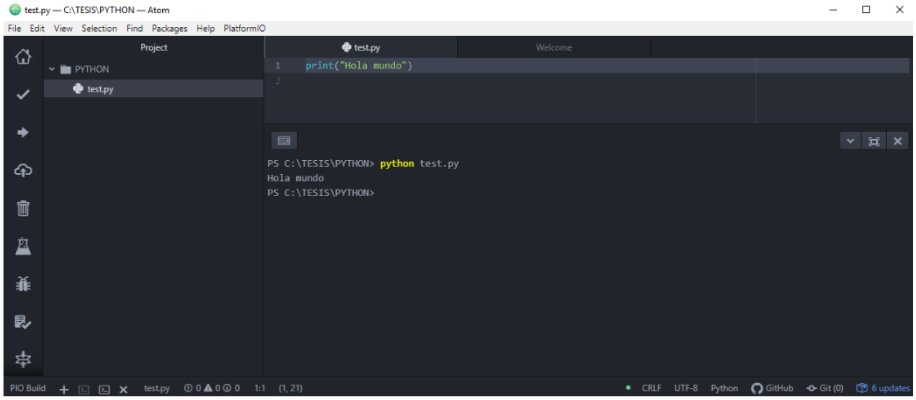


Fig. 9 Terminal de comandos activa en Atom. Autores.

Paso No. 5

Hay que dirijirse hacia la carpeta donde fue instalado Python en la siguiente ruta “C:\Python\Python38\Scripts”. Se lo hace escribiendo el comando “cd” y luego la dirección tal como se muestra en la **Fig. 10**.

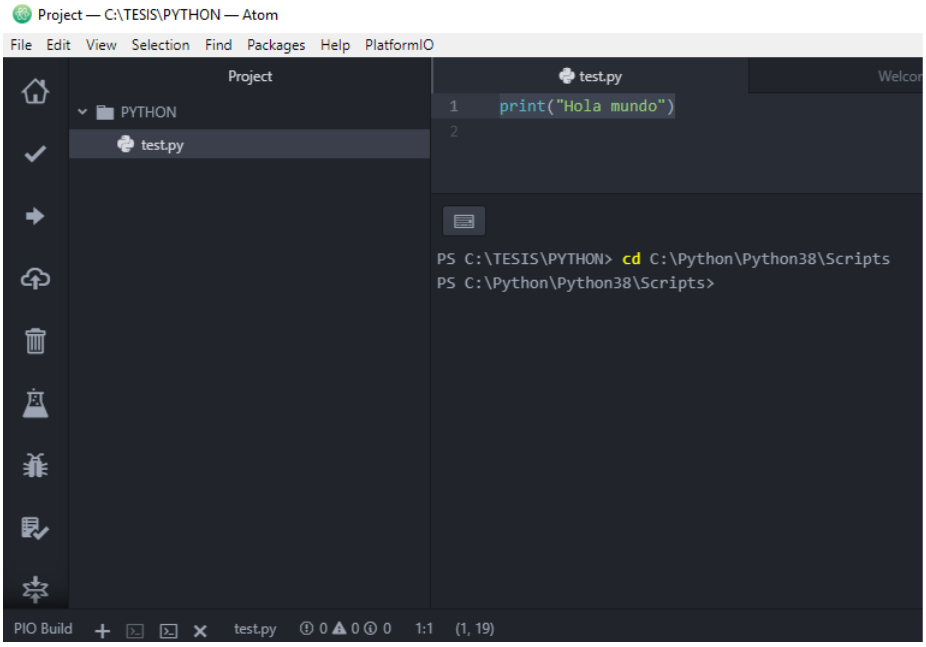


Fig. 10 Terminal de comandos activa en Atom. Autores.

Estando en la ubicación correcta donde se encuentra el instalador de librerías, PIP (Package installer for python), se procede a actualizarlo a su última versión con el siguiente código “python -m pip install --upgrade pip”. Luego de esto se instala cada una de las librerías necesarias para el funcionamiento de la planta de la siguiente manera “Pip install (nombre de librería)” como se muestra en la **Fig. 11**.

Las librerías son las siguientes:

- Numpy: 1.14.5
- OpenCV: 3.3.0 pip install opencv-python
- Tkinter: 8.5 pip install tk
- Pillow: 5.1.0 I
- Imutils: 0.4.6 I
- Serial: 3.4 5
- Pyserial

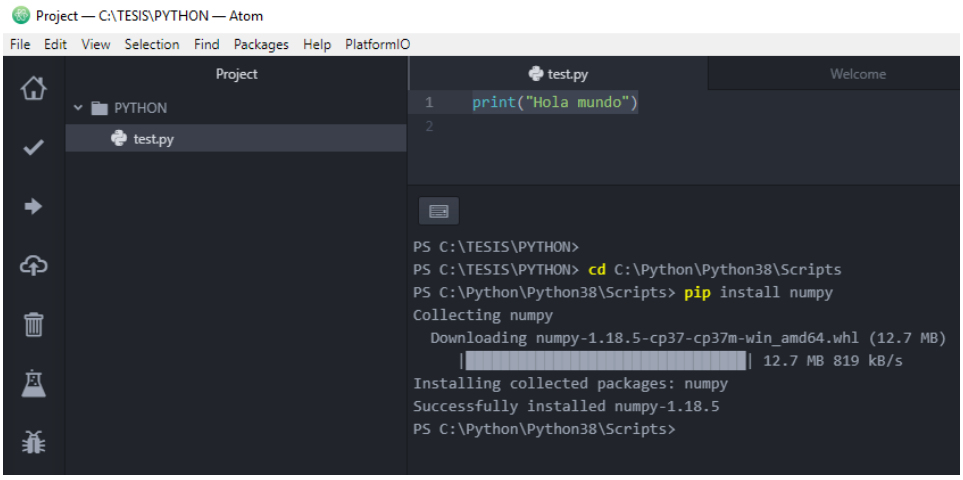


Fig. 11 Pantalla de instalación de librerías de Python. Autores.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**GUIA DE PRÁCTICAS**	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 2**
NOMBRE PRÁCTICA:	Instalación y configuración de planta (Prueba de conexión, test de servomotores.)	

1. **Objetivo**

   Instalar y configurar la tarjeta microcontroladora para el manejo desde la computadora.

2. **Objetivos Específicos**

- Aprender a manejar las bases del lenguaje Python para conexión serial con un microcontrolador.
- Poder hacer manejo de entradas I/O de la tarjeta microcontroladora desde la computadora.
- Calibrar los motores para poder tener definidos parámetros en la planta.
- Poder agregar diodo Led y Servomotor al programa.
- Programar el Microcontrolador con programa Arduino.

3. **Instrucciones**

- Conectar la tarjeta microcontroladora al programa hecho en Python.
- Agregar un servomotor para ser controlado desde el programa.
- Agregar una conexión de salida digital para controlar el encendido y apagado de un led.

4. Actividades a desarrollar

Paso No. 1:

Se abre el programa Atom, luego hay que dirigirse a la carpeta de proyecto donde se encuentra la segunda práctica, y se abre el archivo “PRACTICA_2_EST.PY”. Luego se va a la parte de conexión serial dentro del código donde se tiene que ingresar el nombre de la tarjeta conectada. Tal como se observa en la **Fig 1.** a continuación.

```
ControllerIsConnected = False
def connectController():
    global ser
    global label
    global ControllerIsConnected
    ports = list(serial.tools.list_ports.comports())
    for p in ports:
        print (p)
        #Depende de la tarjeta conectada va a necesitar que se actualice con su nombre
        if "Nombre de tarjeta" in p.description:
            print(p)
            ser = serial.Serial(p[0], 19200, timeout=1)
            time.sleep(1) #Damos un tiempo de espera de un segundo para que se conecte correctamente la tarjeta
            label.configure(text="Microcontrolador conectado!", fg="#36db8b")
            ControllerIsConnected = True
```

Fig. 1 Programación de conexión de tarjeta. Autores.

Para saber el nombre a escoger, se inicializa el programa, se hace click en el botón “Conexión” dentro de este, y se observa en la consola una lista de los dispositivos disponibles a través de puertos COM, tal como se detalla en la **Fig. 2**.

```
PS C:\TESIS\practicas\practica 2> python "PRACTICA_2_EST.PY"
COM2 - Communications Port (COM2)
COM4 - USB Serial Device (COM4)
PS C:\TESIS\practicas\practica 2>
```

Fig. 2 Programación de conexión de tarjeta. Autores.

Una vez identificado, se escribe el nombre donde se encuentra “Nombre de tarjeta” dentro de la programación, en este caso “USB Serial Device”.

Paso No. 2:

Se observa en el panel solamente un servomotor cuando se inicia el programa, también un botón para encender el segundo led, con lo que se procede a agregar el segundo servomotor y el botón de apagado del segundo led. Se visualiza en la siguiente imagen como se declaran las variables del segundo servo como también el mando de control de este. Como se observa en la **Fig. 3**.

```
servoAMov = tk.Scale(FrameBallControl, from_=0, to=180, orient="horizontal", label="Servo A", length=760, tickinterval=5,
                     resolution=1)
servoAMov.set(servoAMovDefault)
servoAMov.pack()

servoBMov = tk.Scale(FrameBallControl, from_=0, to=180, orient="horizontal", label="Servo B", length=760, tickinterval=10,
                     resolution=1)
servoBMov.set(servoBMovDefault)
servoBMov.pack()
```

```
#Valor default asignado a los servomotores
servoAMovDefault = 90
servoBMovDefault = 90


def resetSlider():
#Valor default al presionar reset
    servoAMov.set(servoAMovDefault)
    servoBMov.set(servoBMovDefault)
```

Fig. 3 Asignación de variables y mando de control de servo. Autores.

La lectura que se hace desde la interfaz gráfica se la obtiene a través de la función “ServoBMov.get()”, y ya convertida se la envía como parte de la cadena de valores que transformará en la tarjeta microcontroladora en movimientos de los servomotores. Se observa en la **Fig. 4** como queda la línea con el segundo servo agregado.

```
def main():
    start_timeFPS = time.time()
    global timeInterval, start_time
    global showVideoWindow
    global startBalanceBall, ControllerIsConnected

    if ControllerIsConnected == True and startBalanceBall == True:

 #Enviamos valores de movimiento de servos a tarjeta microcontroladora
        ser.write(("("+str(float(servoAMov.get()))+","+str(float(servoBMov.get()))+","+str(float(servoCMov.get()))+")"+"\n").encode())
#Imprimimos valores a través de puerto Serial para verificar
    #print(("("+str(float(servoAMov.get()))+","+str(float(servoBMov.get()))+","+str(float(servoCMov.get()))+")"+"\n").encode())
```

Fig. 4 Código que envía por Serial los valores de los servomotores. Autores.

Luego se verifica en el programa que haya sido agregado el segundo servo como también su correcto funcionamiento.

Paso No. 3:

Para poder apagar el led es necesario el envío de la cadena “led2off\n”, además de los caracteres “\n” que hace ejecutar la acción una vez que llegue a la tarjeta microcontroladora lo enviado por el puerto serie. En la **Fig. 5** se puede visualizar como se apagará el led cuando se haga un reset hacia los valores iniciales.

```
42  def resetSlider():
43  #Valor default al presionar reset
44      servoAMov.set(servoAMovDefault)
45      servoBMov.set(servoBMovDefault)
46
47      print(("led1off\n").encode())
48      ser.write(("led1off\n").encode())
49      print(("led2off\n").encode())
50      ser.write(("led2off\n").encode())
```

Fig. 5 Código que envía por Serial los valores de los servomotores. Autores.

Por último, se agrega el botón que apagará el segundo Led desde la interfaz gráfica. En la **Fig. 6** puede ver como se agrega el botón, como su ubicación, y que sucederá cuando sea activado.

```
155  BtnLed2On = tk.Button(ControlLedBoton, text="Encender Led 2", command=encenderLed2)
156  BtnLed2On.pack()
157  BtnLed2On.place(x=20,y=60,width=150)
158  #Nombre de función = tk.button(Label donde va el botón, text="Texto Mostrado en GUI", command=Función que llamará)
159  BtnLed2Off = tk.Button(ControlLedBoton, text="Apagar Led 2", command=apagarLed2)
160  BtnLed2Off.pack()
161  BtnLed2Off.place(x=20,y=60,width=150)
```

```
71  def encenderLed2():
72      if ControllerIsConnected == True:
73              print(("led2on\n").encode())
74              ser.write(("led2on\n").encode())
75      else:
76          if tkinter.messagebox.askokcancel("Advertencia!","El microcontrolador no se encuentra conectado."):
77              donothing()
78
79  def apagadarLed2():
80      if ControllerIsConnected == True:
81              print(("led2off\n").encode())
82              ser.write(("led2off\n").encode())
83      else:
84          if tkinter.messagebox.askokcancel("Advertencia!","El microcontrolador no se encuentra conectado."):
85              donothing()
```

Fig. 6 Código que envíado por Serial al apagar el segundo led. Autores.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**GUIA DE PRÁCTICAS**	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 3**
NOMBRE PRÁCTICA:	Configuración de sistema de visión artificial (Test de seguimiento de objeto.)	

1. Objetivo

Aprender bases de programación de visión artificial.

2. Objetivos Específicos

- Aprender a hacer detección de objetos a través de Python y librería OpenCV.
- Observar parámetros de adquisición de datos a través de GUI.
- Calibrar sistema de visión artificial para detectar un objeto de color específico.
- Identificar distintos ejemplos sobre visión artificial.

3. Instrucciones

- Calibrar la corrección de color para identificar la esfera con que se hace el control.
- Verificar programación de distintas acciones en la interfaz gráfica.
- Identificar detección de rostro por visión artificial.
- Identificar detección de ojos por visión artificial.

4. Actividades a desarrollar

Paso No. 1:

Se inicializa el programa “PRACTICA_3_EST.PY”. Se da click al botón “Mostrar vídeo” y se puede observar la pantalla de la cámara que empieza sin valores definidos y tomando bordes cualquier elemento. Luego de eso se ubica el objeto dentro del ángulo de visión y se da click en el botón medio del mouse, tomando así el color al que deseamos hacer tracking. Como se muestra en la **Fig. 1**.

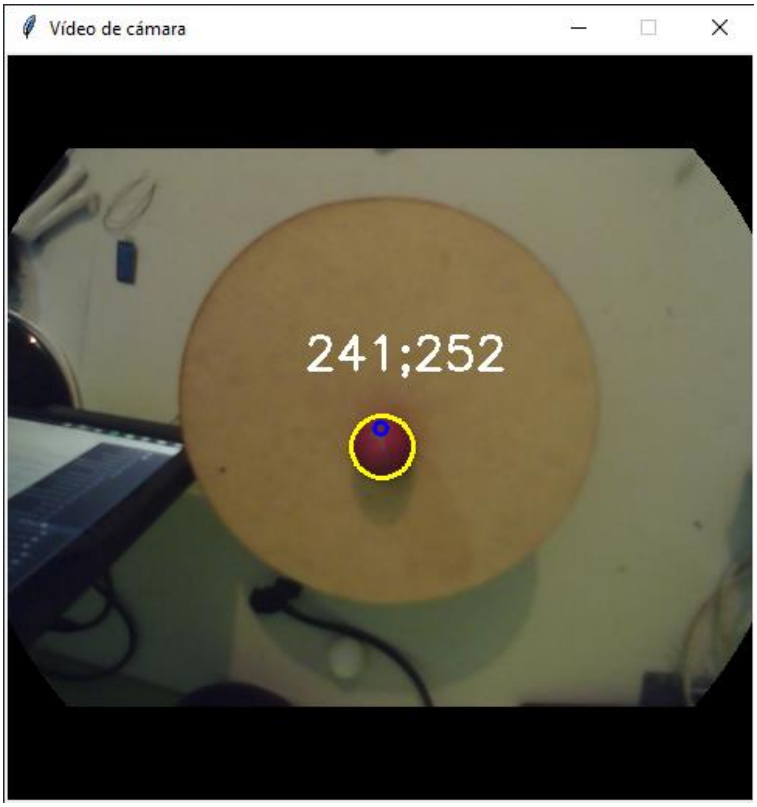


Fig. 1 Objeto reconocido por visión artificial. Autores.

Una vez se mantenga en seguimiento el objeto deseado, se da click al botón capa límites, y se observa cómo hace la separación en la capa de escala de gris (Pantalla “Objeto”), y otra diferenciando entre lo escogido, mientras omite lo demás (Pantalla “Máscara”). Tal como se muestra en la **Fig. 2**.

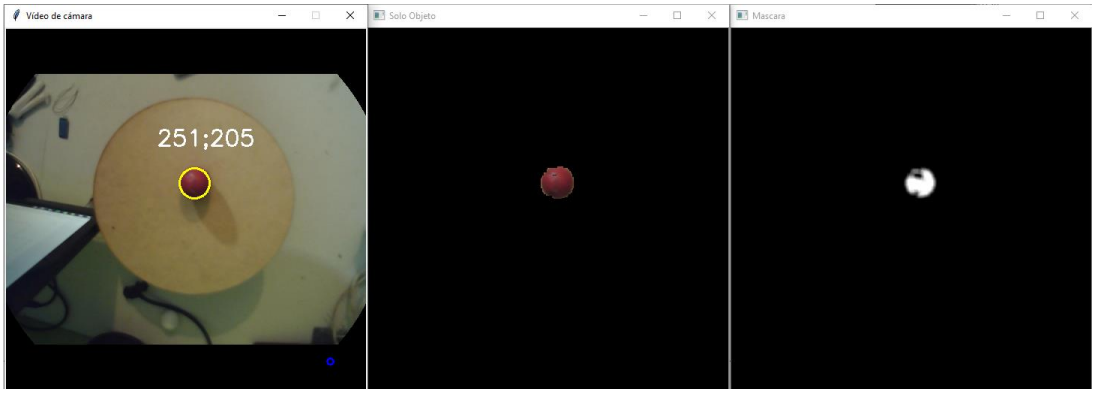


Fig. 2 Capas de visualización de objetos. Los autores.

Se puede observar en la pantalla principal los últimos valores H, S, V escogidos por el click que se dio y se maneja la sensibilidad para que solamente detecte lo que se busca cambiando el rango en los controles. Una vez se tienen definidos esos valores, se los anota para poder determinar una vez que deseemos hacer el control. Como se detalla en la **Fig. 3**.

Fig. 3 Pantalla principal de la práctica 3. Autores.

Paso No. 2:

Se pueden hacer varios tipos de reconocimiento de objetos, el primero fue a través de parámetros del color (H, S, V). En este paso se hace reconocimiento a través de “HAAR “Cascade” que es un sistema entrenado a través de Machine Learning. En la siguiente práctica se enseñan dos ejemplos. El primero el reconocimiento de rostro que se abre al hacer click en el botón “Detección rostro” y el segundo “Detección de ojos”. En la **Fig. 4** se puede observar la programación para llamar y observar sobre la imagen una etiqueta que indica el objeto reconocido.

```
if showVisionArtificialOjosBool == True:
    #Abrimos el archivo donde se encuentran los parámetros para reconocimiento de rostro
    eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
    #Generamos la variable de captura de pantalla
    cap = cv2.VideoCapture(0)
    #Mientras se encuentre activa la captura de pantalla
    while cap.isOpened():
    #Guardamos la imagen receptada por la cámara en la variable "img"
        _, img = cap.read()
    #Convertimos la imagen a escala de grises para una mejor detección de parámetros
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #Determinamos el objeto buscado en nuestra imgaen que se encuentra en escala de grises
        eyes = eye_cascade.detectMultiScale(gray, 1.1, 4)
    #Le damos color y forma en el plano del objeto encontrado dentro de la imagen tomada
        for (x, y , w ,h) in eyes:
            cv2.rectangle(img, (x,y), (x+w, y+h), (255, 0 , 0), 3)
            cv2.rectangle(gray, (x,y), (x+w, y+h), (255, 0 , 0), 3)
    #Mostreamos el cuadro de imagen con la detección del objeto
        cv2.imshow('Detección ojos', img)
        cv2.imshow('Escala de Grises', gray)
    #Cerramos el progrma al aplastar la tecla "q"
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
```

Fig. 4 Programación de HAAR Cascade. Autores.

En la **Fig. 5** se puede ver como se hace efectivo el reconocimiento de rostro, como también de ojos.

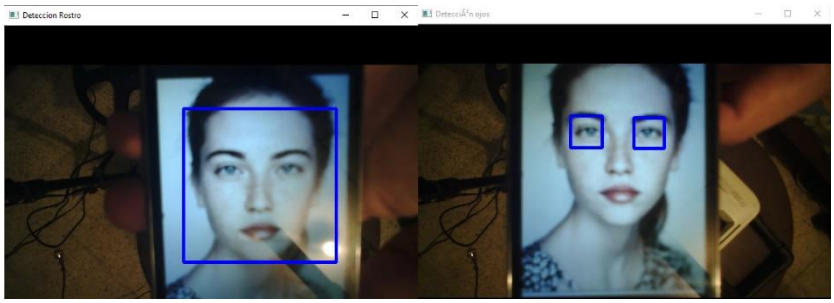


Fig. 5 Captura de reconocimiento de rostro y ojos. Fuente: Los autores.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**GUIA DE PRÁCTICAS**	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 4**
NOMBRE PRÁCTICA:	Modelado matemático del sistema de la planta.	

PRÁCTICA 4

1. **Objetivo**

- Determinar ángulos de funcionamiento de la planta para estabilización de esfera

2. **Objetivos Específicos**

- Determinar distancia entre uniones de la plataforma en la planta.
- Encontrar ecuaciones de movimiento de servos.

3. **Instrucciones**

- Determinación de valores de los servomotores para su correcto modelado matemático.

4. **Actividades a desarrollar**

Paso No. 1:

Para la relación no linear entre el ángulo alfa y teta, se genera una tabla para mapear los valores de los servomotores para ubicar la plataforma en la ubicación deseada. Esto se lo hace con un programa en Python llamado “PRACTICA_4.py” que genera un archivo de nombre “data.txt” que será llamado al comenzar el

funcionamiento de la interface. Se puede ver en la siguiente imagen el mapeado de los valores mínimos y máximos como también los valores de las uniones para determinar los ángulos. Como muestra la **Fig.1** y la **Fig. 2.**

```python
def translate(value, leftMin, leftMax, rightMin, rightMax):     #Esta función es e
    # Determinar que ancho es cada rango.
    leftSpan = leftMax - leftMin
    rightSpan = rightMax - rightMin

    # Convertir el ranzgo izquierda en valores entre 0-1 (float).
    valueScaled = float(value - leftMin) / float(leftSpan)


    # Convertir el rango de 0 a 1 a un valor del rango derecho.
    return rightMin + (valueScaled * rightSpan)

#Valores en cm
dm = 1.7    # Distancia desde el centro del motor hasta el polo que conecta a la p
dc = 6.2    # Distancia desde el centro de la plataforma hasta el eje que conecta
K = dc/dm   # Factor de relación
```

Fig. 1 Inicio de programa que resuelve ecuaciones. Autores.

```python
file = open("data.txt", "w") # Crear el archivo
firstline = "alpha|theta\n" # Definir primera línea como título
file.write(firstline) # Escribir primera línea en el programa

maxAlpha = 14.9*10  # ángulo máximo alga en grados multiplicado x10 para ser escalado
alpha = radians(maxAlpha/10)
#max_theta = arcsin(sin(alpha)*K)
max_theta = 180

for alpha in range(-maxAlpha,maxAlpha+1):
    alpha = radians(alpha/10)
    theta = arcsin(sin(alpha)*K) + pi/2
    alpha = degrees(alpha)
    theta = translate(degrees(theta),20,160,0,180)
    if theta > 120 and theta < 140:                    # Diferentes áreas de no linealidad
            theta = translate(theta,120,140,120,145)
    elif theta > 140 and theta < 160:                  # Diferentes áreas de no linealidad
            theta = translate(theta,140,160,145,150)
    elif theta > 160 and theta < 180:                  # Diferentes áreas de no linealidad
            theta = translate(theta,160,180,150,159)
    #theta = round(degrees(theta),0)
    separator = "|"
    data = str(round(alpha,2)) + separator + str(int(round(theta,0))) + "\n"
    file.write(data)
```

Fig. 2 Relaciones entre los ángulos a convertir. Autores.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**GUIA DE PRÁCTICAS**	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 5**
NOMBRE PRÁCTICA:	Configuración e implementación de PD.	

PRÁCTICA 5

1. **Objetivos**

- Determinar valores PD necesarios para estabilización de la planta.

2. **Objetivos Específicos**

- Identificar requerimientos de la planta para su estabilización.
- Simular diseño de control.
- Determinar modelo en tiempo discreto en Simulink.

3. **Instrucciones**

- Identificar requerimientos de control.
- Simular diseño de control.
- Determinar modelo en tiempo discreto en Simulink.
- Ubicar los valores en la interfaz gráfica para estabilizar la planta.

4. **Actividades a desarrollar**

Paso No. 1:

Anexo: Capítulo 3.6 en tesis. Modelo matemático de la planta.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**GUIA DE PRÁCTICAS**	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 6**
NOMBRE PRÁCTICA:	Programa interactivo para mantener esfera en el centro del plato	

PRÁCTICA 6

1. Objetivo.

Determinar algoritmos que nos permiten mantener estable la esfera dentro del plato.

2. Objetivos Específicos

- Aprender a estabilizar la esfera dentro del plato.
- Poder ubicar la esfera en un punto seleccionado dentro del plato.

1. Instrucciones

- Inicializar el programa de interface con la tarjeta.
- Conectar la tarjeta microcontroladora con el programa.
- Escoger el elemento a estabilizar.

2. Actividades a desarrollar

Paso No. 1:

Antes de abrir el programa verificamos que parte del código se encarga de mantener la esfera estable en el punto inicial.

Las variables “refX, refY” son las encargadas de ubicar el punto donde se encontrará la esfera siendo este valor el SetPoint. Este empezará con un valor de 240 en los dos ejes, indicando el centro de la plataforma. En la **Fig. 1** se puede observar parte de la programación que toma el nuevo valor de referencia.

```
152 def setRefWithMouse(mousePosition):   # Asignar refX y refY basados en mousePosition, mou
153     global refX, refY
154     if mousePosition.y > 10:
155         refreshGraph()
156         refX, refY = mousePosition.x, mousePosition.y
157         resetPID()
158
159 def setRefWithButton():   # Asignar refX y refY basados en mousePosition, mousePosition
160     global refX, refY
161     refX, refY = sliderRefX.get(), sliderRefY.get()
```

Fig. 1 Imagen de programación donde se obtiene el valor de referencia. Autores.

Si es detectado un objeto que tenga un radio mayor a 10 píxeles, se dibuja en contorno sobre él, y el programa comienza a enviar valores hacia el control. Se puede observar en la Fig. 2.

```
562     if len(cnts) > 0:
563         c = max(cnts, key=cv2.contourArea)
564         timeInterval = time.time() - start_time
565         (x, y), radius = cv2.minEnclosingCircle(c)
566         if radius > 10: #Si un objeto es detectado y su radio es mayor a 10, comienza a enviar valores
567             cv2.putText(img, str(int(x)) + ";" + str(int(y)).format(0, 0), (int(x) - 50, int(y) - 50),
568                         cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
569             cv2.circle(img, (int(x), int(y)), int(radius), (0, 255, 255), 2)
570             PIDcontrol(int(x), int(y), prevX, prevY, refX, refY)
571             start_time = time.time()
```

Fig. 2 Imagen de programación donde se obtiene el valor de referencia. Autores.

Una vez que recepte valores el controlador, este comienza a calcular sus valores a través de las funciones asignadas siempre tratando de estabilizarlo en el punto deseado. Se puede ver en la siguiente imagen los valores que se envían por el puerto Serial. Como se detalla en la **Fig 3**.

```
if MicrocontroladorIsConnected == True and startBalanceBall == True:
    ser.write((str(dataDict[Ix]) + "," + str(dataDict[-Iy]) + "\n").encode())  #Envío

if startBalanceBall == True:
```

Fig. 3 Imagen de programación donde se envía valores al microcontrolador. Autores.

Paso No. 2:

Una vez el programa se encuentre funcionando, se hace click dentro del recuadro donde se encuentra el vídeo en vivo, y se hace click hacia donde se desea que la esfera vaya. Se puede apreciar en la **Fig. 4** una captura de la cámara junto con el objeto a seguir.

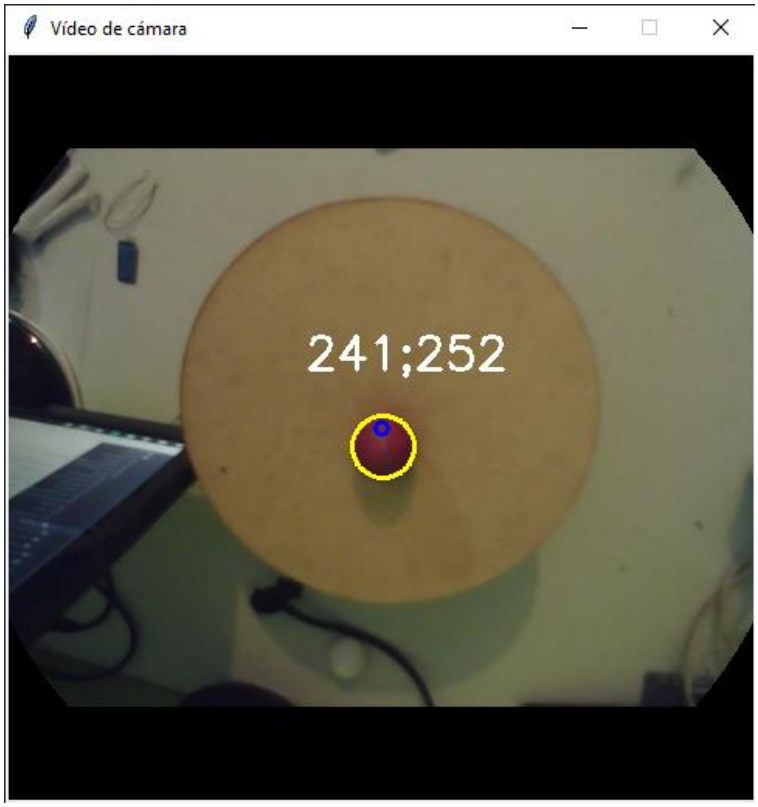


Fig. 4 Imagen de captura en tiempo real de la cámara. Autores.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**GUIA DE PRÁCTICAS**	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 7**
NOMBRE PRÁCTICA:	Programa interactivo de movimiento de esfera según trayectoria deseada.	

1. Objetivo.

Determinar algoritmos que permiten mover la esfera alrededor de la plataforma

2. Objetivos Específicos

- Aprender a movilizar la esfera dentro de la plataforma.
- Poder escoger radio y velocidad del movimiento.
- Comprarar diferentes respuestas de funcionameinto cambiando valores de PD

3. Instrucciones

- Inicializar el programa de interface con la tarjeta.
- Conectar la tarjeta microcontroladora con el programa.
- Escoger el elemento a estabilizar.

4.Actividades a desarrollar

Paso No. 1:

Antes de abrir el programa, se revisa que parte del código se encarga de mover la esfera en la trayectoria deseada.

La función *drawWithBall()* hace la verificación sobre qué movimiento se va a realizar, sea circular u otro que parezca el número 8. Siendo este un círculo y al regresar al punto cero, hace otro, pero hacia el eje contrario. La variable *point es* la cantidad de puntos que tendrá la función en la actualización, convirtiéndose en la velocidad del movimiento que obtenemos del controlador con la variable *sliderSpeed.get().* Se puede observar en **Fig.1** parte del programa que hace la activación.

```
134 def drawWithBall():     # Función encendida que hace el dibujo del circulo
135     global pointCounter, refX, refY
136     if drawCircleBool == True:
137         #sliderCoefP.set(15)
138         if pointCounter >= len(pointsListCircle):
139             pointCounter = 0
140         point = pointsListCircle[pointCounter]
141         refX, refY = point[0], point[1]
142         pointCounter += sliderSpeed.get()
143     if drawEightBool == True:
144         #sliderCoefP.set(15)
145         if pointCounter >= len(pointsListEight):
146             pointCounter = 0
147         point = pointsListEight[pointCounter]
148         refX, refY = point[0], point[1]
149         pointCounter += sliderSpeed.get()
```

Fig. 1 Imagen de programación encargada de función de movimientos. Autores.

Al activar el movimiento circular, comienza la función que genera una circunferencia indefinidamente. Trabaja con un lazo *for* que una vez que llegue al rango máximo, vuelve al punto inicial, y tomando sus puntos con la distancia del radio deseado que obtenemos desde el controlador con la variable *sliderRadius.get()* y multiplicando por el coseno y el seno del ángulo formando la circunferencia. En la **Fig. 2** se observa la programación de la función circular.

```
def createPointsListCircle():    #Crear un arreglo donde estarán los puntos d
    global pointsListCircle
    pointsListCircle = []
    for angle in range(0, 360):
        angle = angle - 90
        pointsListCircle.append([sliderRadius.get() * cos(radians(angle))
        + 240, sliderRadius.get() * sin(radians(angle)) + 240])
```

Fig. 2 Imagen de programación encargada de generar movimiento circular. Autores.

Paso No. 2:

Se procede con la verificación de la parte de la programación encargada de hacer e movimiento de número 8. Así mismo genera una lista de puntos, pero esta vez dos lazos *for.* Uno para generar una circunferencia superior, y el otro una inferior, tal como detalla la **Fig. 3.**

```
def createPointsListEight():      #Crear un arreglo donde estarán los puntos del p
    global pointsListEight
    pointsListEight = []
    for angle in range(270, 270 + 360):
        pointsListEight.append([sliderRadius.get() * cos(radians(angle)) + 240,
         sliderRadius.get() * sin(radians(angle)) + 240 + sliderRadius.get()])
    for angle in range(360, 0, -1):
        angle = angle + 90
        pointsListEight.append([sliderRadius.get() * cos(radians(angle)) + 240,
        sliderRadius.get() * sin(radians(angle)) + 240 - sliderRadius.get()])
```

Fig. 3 Imagen de programación que generar movimiento en forma de 8. Autores.

Paso No. 3:

La parte encargada de manejar o valores de los movimientos es la que se encuentra enmarcada de rojo en la **Fig. 4**.

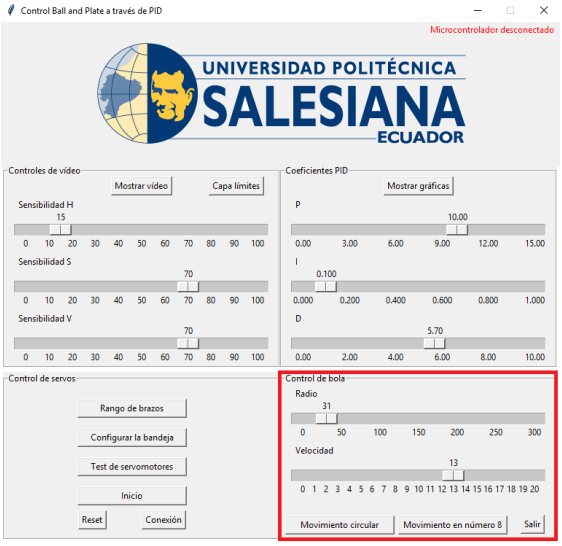


Fig. 4 Imagen de programación que generar movimiento en forma de número ocho. Autores.

Paso No. 4:

Mientras la esfera esté teniendo movimiento, abrir la ventana “Mostrar gráfica” y verificar la respuesta de la planta mientras se hacen cambios en factores P y D. Hacer una comparativa hasta llegar al movimiento más estable del sistema.

Anexo 2: Códigos de Programación

CÓDIGOS DE MATLAB

```
close all;
clear;
clc;

dm = 0.025;         %m    Distancia del eje del motor y eje rotatorio
dc = 0.150;         %m    Distancia entre el centro de la plataforma y el
eje de contacto
fc = 0.001 ;        %Ns/m  Coeficiente de fricción
Eta = 1.0087;    %Relación de amortiguación (Damping ratio) del servo
motor
%Velocidad Angular = Angulo recorrido / Tiempo = 1,0472 Rad / t
%Conversión de 6.0V: 0.19 sec/60° a rad = 60*pi/180 / 0.19
%Velocidad Angular FUTABA S3003 = 5.5116 rad/s
Wn = 5.5116; %rad/s Servo natural Frequency FUTUBA S3003
%Wn = 20; %rad/s Servo natural Frequency FUTUBA S3003

g = 9.81  ;%m/s^2 gravitational acceleration
K = dc/dm;

% Ball Type     |  Radio |  Masa
% Metal Hueca   |  0.016  |  0.013
% Hierro Masiza |  0.01   |  0.046
% Ping Pong     |  0.02   |  0.001
% Canica        |  0.007  |  0.0049

rb = 0.027/2;%m   Radio de la Bola
mb = 0.0050;%kg   Masa de la Bola  %Verificar



A = [ 0 1 0 0; 0 (-5/7)*(fc/mb) (+5/7)*g 0; 0 0 0 1; 0 0 -(Wn^2) -
2*Eta*Wn]
B = [ 0 ; 0 ; 0 ; Wn^2]
C = [ 1 0 0 0]
D = zeros(1,1)
```

```
[bb,aa]= ss2tf(A,B,C,D)
sistema=tf(bb,aa)



%La estabilidad depende del vector característico de la Matriz A:
E=eig(A)

%E =   0
%     -0.0893
%     -4.8309
%     -6.2882

%Análisis de Controlabilidad
Crt = ctrb(A,B) %controllability matrix of size nxn where n is the
%dimension of the state space

rank(Crt) % if full rank, means rank = n then system is controllable

%Análisis de Observavilidad
ob=obsv(A,C)
rank(ob) % if full rank, means rank = n then system is controllable
```

Código de Tarjeta Microcontroladora ATMEGA 328

```
#include <Servo.h>

Servo servoA;
Servo servoB;



float angleA = 90;
float angleB = 90;
```

```
int led1 = 12;
int led2 = 13;

void setup() {
  //Empieza comunicación Serial
  Serial.begin(19200);
  //Declaración de pones de salida
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  //Declaración de ángulos iniciales
  servoA.attach(9,500,2600);
  servoB.attach(10,500,2600);
  servoA.write(angleA);
  servoB.write(angleB);
  //Comprobación de encendido
  delay(1000);
  digitalWrite(led1,HIGH);
  delay(1000);
  digitalWrite(led1,LOW);
  delay(1000);
  digitalWrite(led1,HIGH);
  delay(1000);
  digitalWrite(led1,LOW);

}

int count = 0;

void loop() {
  //Con conexión serial efectiva, enviar los datos al servo
  if(Serial.available() > 0) {
    String a = Serial.readStringUntil('\n');
    angleA = getValue(a, ',', 0).toFloat();
    angleB = getValue(a, ',', 1).toFloat();
    servoA.write(angleA);
    servoB.write(angleB);
```

```
  //Si se envia los comandos desde Python, encender y apagado de leds
    if(a == "led1on"){
        digitalWrite(led1,HIGH);
    }
    if(a == "led1off"){
        digitalWrite(led1,LOW);
    }
   if(a == "led2on"){
        digitalWrite(led2,HIGH);
    }
    if(a == "led2off"){
        digitalWrite(led2,LOW);
    }

  }

}

String getValue(String data, char separator, int index) {
    int found = 0;
    int strIndex[] = { 0, -1 };
    int maxIndex = data.length() - 1;

    for (int i = 0; i <= maxIndex && found <= index; i++) {
        if (data.charAt(i) == separator || i == maxIndex) {
            found++;
            strIndex[0] = strIndex[1] + 1;
            strIndex[1] = (i == maxIndex) ? i+1 : i;
        }
    }
    return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}
```

Códigos de Prácticas en Python

PRÁCTICA # 2

```
import time
import tkinter as tk
import tkinter.messagebox
from PIL import Image, ImageTk
import serial
import serial.tools.list_ports

path = 'C:/TESIS/Practicas/PRACTICA 2/logoups.png'

controllerWindow = tk.Tk()
controllerWindow.title("Control Ball and Plate a través de PID")
#Tamaño de la pantalla
controllerWindow.geometry("770x770")
controllerWindow["bg"]="white"
controllerWindow.resizable(0, 0)

#Declaración de llamado de la imagen de portada
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)
lmain = tk.Label(videoWindow)
lmain.pack()
videoWindow.withdraw()


showVideoWindow = False
def showCameraFrameWindow():
```

```
        global showVideoWindow

    def endProgam():
        controllerWindow.destroy()
    #Valor default asignado a los servomotores
    servoAMovDefault = 90
    servoBMovDefault = 90



    def resetSlider():
    #Valor default al presionar reset
        servoAMov.set(servoAMovDefault)
        servoBMov.set(servoBMovDefault)

        print(("led1off\n").encode())
        ser.write(("led1off\n").encode())
        print(("led2off\n").encode())
        ser.write(("led2off\n").encode())

    def donothing():
        pass

    def encenderLed1():
        if ControllerIsConnected == True:
                print(("led1on\n").encode())
                ser.write(("led1on\n").encode())
        else:
            if tkinter.messagebox.askokcancel("Advertencia!","El microcontrolador no se encuentra
conectado."):
                donothing()

    def apagarLed1():
        if ControllerIsConnected == True:
                print(("led1off\n").encode())
                ser.write(("led1off\n").encode())
        else:
```

```
        if tkinter.messagebox.askokcancel("Advertencia!","El microcontrolador no se encuentra conectado."):
            donothing()


    def encenderLed2():
        if ControllerIsConnected == True:
            print(("led2on\n").encode())
            ser.write(("led2on\n").encode())
        else:
        if tkinter.messagebox.askokcancel("Advertencia!","El microcontrolador no se encuentra conectado."):
            donothing()


    def apagarLed2():
        if ControllerIsConnected == True:
            print(("led2off\n").encode())
            ser.write(("led2off\n").encode())
        else:
        if tkinter.messagebox.askokcancel("Advertencia!","El microcontrolador no se encuentra conectado."):
            donothing()


    ControllerIsConnected = False
    def connectController():
        global ser
        global label
        global ControllerIsConnected
        ports = list(serial.tools.list_ports.comports())
        for p in ports:
            print (p)
            #Depende de la tarjeta conectada va a necesitar que se actualice con su nombre
            if "USB-SERIAL CH340" or "USB Serial Device" in p.description:
                print(p)
                ser = serial.Serial(p[0], 19200, timeout=1)
                time.sleep(1) #Damos un tiempo de espera de un segundo para que se conecte correctamente la tarjeta
                label.configure(text="Microcontrolador conectado!", fg="#36db8b")
                ControllerIsConnected = True
```

```
startBalanceBall = False
def startBalance():
    global startBalanceBall
    if ControllerIsConnected == True:
        if startBalanceBall == False:
            startBalanceBall = True
            BStartBalance["text"] = "Desconectar"
        else:
            startBalanceBall = False
            BStartBalance["text"] = "Conectar"
    else:
        if tkinter.messagebox.askokcancel("Advertencia!","El microcontrolador no se encuentra conectado."):
            donothing()

def main():
    start_timeFPS = time.time()
    global timeInterval, start_time
    global showVideoWindow
    global startBalanceBall, ControllerIsConnected

    if ControllerIsConnected == True and startBalanceBall == True:

 #Enviamos valores de movimiento de servos a tarjeta microcontroladora
        ser.write((str(float(servoAMov.get()))+","+str(float(servoBMov.get()))+"\n").encode())
#Imprimimos valores a través de puerto Serial para verificar
    #print((str(float(servoAMov.get()))+","+str(float(servoBMov.get()))+"\n").encode())



    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)




    #Etiqueta de pantalla para control de leds
    ControlLedBoton = tk.LabelFrame(controllerWindow, text="Encendido LED")
    ControlLedBoton.place(x=305,y=470,width=770, height=230)


    #Boton1 = tk.button(Etiqueta de control led, text= Texto en botón, command= Comando que funcionará al hacerle click al botón)
    BtnLed1On = tk.Button(ControlLedBoton, text="Encender Led 1", command=encenderLed1)
    #Llámada de la clase botón 1
    BtnLed1On.pack()
    # Ubicación del botón respecto al label
    BtnLed1On.place(x=20,y=20,width=150)
    BtnLed1Off = tk.Button(ControlLedBoton, text="Apagar Led 1", command=apagarLed1)
    BtnLed1Off.pack()
    BtnLed1Off.place(x=200,y=20,width=150)
    BtnLed2On = tk.Button(ControlLedBoton, text="Encender Led 2", command=encenderLed2)
    BtnLed2On.pack()
    BtnLed2On.place(x=20,y=60,width=150)
    BtnLed2Off = tk.Button(ControlLedBoton, text="Apagar Led 2", command=apagarLed2)
    BtnLed2Off.pack()
    BtnLed2Off.place(x=200,y=60,width=150)
    BStartBalance = tk.Button(ControlLedBoton, text="Inicio", command=startBalance, highlightbackground = "#36db8b")
    BStartBalance.pack()
    BStartBalance.place(x=20,y=140,width=150)



    FrameBallControl = tk.LabelFrame(controllerWindow, text="Control de Servomotores")
    FrameBallControl.place(x=5,y=199,width=770, height= 270)
    #Declaración mando de control para movimiento de servo
    # Nombre de Instancia = tk.Scale(Pantalla o Label donde se encuentra, desde=0, hasta=180, orientación ="horizontal, etiqueta"Servo A", largo=760, intervalo de valores=5, resolución=1)
    servoAMov = tk.Scale(FrameBallControl, from_=0, to=180, orient="horizontal", label="Servo A", length=760, tickinterval=5,
                         resolution=1)
```

```
    servoAMov.set(servoAMovDefault)
    servoAMov.pack()

    servoBMov = tk.Scale(FrameBallControl, from_=0, to=180, orient="horizontal", label="Servo B", length=760, tickinterval=10,
                        resolution=1)
    servoBMov.set(servoBMovDefault)
    servoBMov.pack()



    label = tk.Label(controllerWindow, text="Microcontrolador desconectado ", fg="red", anchor="ne")
    label.pack(fill="both")
    BReset = tk.Button(controllerWindow, text = "Reset", command = resetSlider)
    BReset.place(x=328, y=670)
    BConnect = tk.Button(controllerWindow, text = "Conexión", command = connectController)
    BConnect.place(x=415, y=670)
    BQuit = tk.Button(controllerWindow, text = "Salir", command = endProgam)
    BQuit.place(x=555, y=670, width=100)

    videoWindow.protocol("WM_DELETE_WINDOW",donothing)



    main()
    tk.mainloop()
```

PRÁCTICA # 3

```
    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 *

path = 'C:/TESIS/Practicas/PRACTICA 2/logoups.png'
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
dataDict = {}

camHeight = 480
camWidth = 640
cam = cv2.VideoCapture(0)       # Selecciona cámara ( 0 es el default de la primera cámara)
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 de bola
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("Control Ball and Plate a través de PID")
#Tamaño de la pantalla
controllerWindow.geometry("770x770")
controllerWindow["bg"]="white"
controllerWindow.resizable(0, 0)



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)  #empeche de modifier les dimensions de la fenetre
lmain = tk.Label(videoWindow)
lmain.pack()
videoWindow.withdraw()



def setConsigneWithMouse(mousePosition):
    global consigneX, consigneY
    if mousePosition.y > 10:
        refreshGraph()
        consigneX,consigneY = mousePosition.x,mousePosition.y

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

```
showVideoWindow = False
def showVisionArtificialVision():
    global showVideoWindow, showGraph
    global BRetourVideoTxt
    if showVideoWindow == False:
        if showGraph == True:
            graphWindow.withdraw()
            BafficherGraph["text"] = "Mostrar gráficas"
        videoWindow.deiconify()
        showVideoWindow = True
        BRetourVideo["text"] = "Quitar vídeo "
    else:
        videoWindow.withdraw()
        showVideoWindow = False
        BRetourVideo["text"] = "Mostrar vídeo"


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

showVisionArtificialRostroBool = False
def showVisionArtificialRostro():
    global showVisionArtificialRostroBool
    showVisionArtificialRostroBool = not showVisionArtificialRostroBool

showVisionArtificialOjosBool = False
def showVisionArtificialOjos():
    global showVisionArtificialOjosBool
    showVisionArtificialOjosBool = not showVisionArtificialOjosBool

showGraph = False
def showGraphWindow():
    global showGraph, showVideoWindow
    global BafficherGraph
```

```
    if showGraph == False:
        if showVideoWindow == True:
            videoWindow.withdraw()
            showVideoWindow = False
            BRetourVideo["text"] = "Mostrar vídeo"
        showGraph = True
        BafficherGraph["text"] = "Ocultar gráficas "
    else:
        showGraph = False
        BafficherGraph["text"] = "Mostrar gráficas"

t = 480
consigneY = 240
consigneX = 240


def refreshGraph():
    global t
    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



    sommeErreurX = 1
    sommeErreurY = 1
    timeInterval = 1



    prevX,prevY = 0,0
    prevConsigneX, prevConsigneY = 0,0
    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

        _, img=cam.read()
        img = img[0:int(camHeight),int((camWidth-camHeight)/2):int(camWidth-((camWidth-
camHeight)/2))] #[Y1:Y2,X1:X2]
        imgCircle = np.zeros(img.shape, dtype=np.uint8)
        cv2.circle(imgCircle, (240,240), 270, (255, 255, 255), -1, 8, 0)
        img = img & imgCircle
        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 = cnts[0] if imutils.is_cv2() else cnts[1]
        cnts = imutils.grab_contours (cnts)

        center = None

        cv2.circle(img, (int(consigneX), int(consigneY)), int(4),(255, 0, 0), 2)
        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 showVisionArtificialRostroBool == True:
            #Abrimos el archivo donde se encuentran los parámetros para reconocimiento de rostro
            face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
            #Generamos la variable de captura de pantalla
            cap = cv2.VideoCapture(0)
            #Mientras se encuentre activa la captura de pantalla
            while cap.isOpened():
                #Guardamos la imagen receptada por la cámara en la variable "img"
                _, img = cap.read()
                #Convertimos la imagen a escala de grises para una mejor detección de parámetros
                gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
            #Determinamos el objeto buscado en nuestra imgaen que se encuentra en escala de
grises
            faces = face_cascade.detectMultiScale(gray, 1.1, 4)
            #Le damos color y forma en el plano del objeto encontrado dentro de la imagen tomada
            for (x, y , w ,h) in faces:
                cv2.rectangle(img, (x,y), (x+w, y+h), (255, 0 , 0), 3)

            #Mostreamos el cuadro de imagen con la detección del objeto
            cv2.imshow('Deteccion Rostro', img)
            #Cerramos el progrma al aplastar la tecla "q"
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        cap.release()

    if showVisionArtificialOjosBool == True:
        #Abrimos el archivo donde se encuentran los parámetros para reconocimiento de rostro
        eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
        #Generamos la variable de captura de pantalla
        cap = cv2.VideoCapture(0)
        #Mientras se encuentre activa la captura de pantalla
        while cap.isOpened():
        #Guardamos la imagen receptada por la cámara en la variable "img"
            _, img = cap.read()
        #Convertimos la imagen a escala de grises para una mejor detección de parámetros
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        #Determinamos el objeto buscado en nuestra imgaen que se encuentra en escala de
grises
            eyes = eye_cascade.detectMultiScale(gray, 1.1, 50)
        #Le damos color y forma en el plano del objeto encontrado dentro de la imagen tomada
            for (x, y , w ,h) in eyes:
                cv2.rectangle(img, (x,y), (x+w, y+h), (255, 0 , 0), 3)
                cv2.rectangle(gray, (x,y), (x+w, y+h), (255, 0 , 0), 3)
        #Mostreamos el cuadro de imagen con la detección del objeto
            cv2.imshow('Detección ojos', img)
            cv2.imshow('Escala de Grises', gray)
        #Cerramos el progrma al aplastar la tecla "q"
```

```
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        cap.release()



    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)




    #print("FPS: ", 1.0 / (time.time() - start_timeFPS))


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="Sensibilidad H", length=350, tickinterval = 10)
    sliderH.set(sliderHDefault)
    sliderH.pack()
    sliderS = tk.Scale(FrameVideoControl, from_=0, to=100, orient="horizontal", label="Sensibilidad S", length=350, tickinterval = 10)
    sliderS.set(sliderSDefault)
    sliderS.pack()
    sliderV = tk.Scale(FrameVideoControl, from_=0, to=100, orient="horizontal", label="Sensibilidad V", length=350, tickinterval = 10)
    sliderV.set(sliderVDefault)
    sliderV.pack()




    FrameServosControl = tk.LabelFrame(controllerWindow, text="Valores H, S, V ")
    FrameServosControl.place(x=5,y=480,width=380, height=230)
    T = tk.Text(FrameServosControl, height=5, width=30)
    T.pack()
    T.insert(tk.END, "Valores HSV escogidos\n")
    label = tk.Label(controllerWindow, text="Microcontrolador desconectado ", fg="red", anchor="ne")
    label.pack(fill="both")
    BReset = tk.Button(controllerWindow, text = "Reset", command = resetSlider)
    BReset.place(x=108, y=670)
    BQuit = tk.Button(controllerWindow, text = "Salir", command = endProgam)
    BQuit.place(x=200, y=670)


    VisionArtificialRostro = tk.LabelFrame(controllerWindow, text="Visión Artificial Rostro")
    VisionArtificialRostro.place(x=385,y=195,width=380, height=280)
    VisionArtificialRostroBoton = tk.Button(VisionArtificialRostro, text="Detección Rostro", command=showVisionArtificialRostro)
    VisionArtificialRostroBoton .pack()
    VisionArtificialRostroBoton .place(x=40,y=40)
```

```
    VisionArtificialOjoBoton = tk.Button(VisionArtificialRostro, text="Detección Ojos",
command=showVisionArtificialOjos)
    VisionArtificialOjoBoton.pack()
    VisionArtificialOjoBoton.place(x=40,y=80)

    videoWindow.protocol("WM_DELETE_WINDOW",donothing)
    videoWindow.bind("<Button-2>",getMouseClickPosition)
    videoWindow.bind("<Button-1>",setConsigneWithMouse)

    main()
    tk.mainloop()
```

PRÁCTICA # 4

```
    from numpy import *
    from math import *

    def translate(value, leftMin, leftMax, rightMin, rightMax):          # Esta función es
equivalente al map() en Arduino
        # Ver que tan "Ancho" sea el rango
        leftSpan = leftMax - leftMin
        rightSpan = rightMax - rightMin

        # Convertir el rango de la izquiera a valores entre 0 - 1 (float)
        valueScaled = float(value - leftMin) / float(leftSpan)

        # Convertir el rango de la derecha a valores entre 0 - 1 (float)
        return rightMin + (valueScaled * rightSpan)

    #Valores en Cm
    dm = 1.7    # Distancia del centro de motor al soporte de la plataforma
    dc = 6.2    # Distancia entre el soporte y el centro de la plataforma
```

```
    K = dc/dm   # factor

    print("...")

    alpha = 0
    theta = 0



    file = open("data.txt", "w") #Crear archivo
    firstline = "alpha|theta\n"     # Definir la primera linea como título
    file.write(firstline) # Escribir la primera línea en el archivo

    maxAlpha = 14.9*10          # Ángulo máximo alpha multiplicado x10 para escalarlo
    alpha = radians(maxAlpha/10)

    max_theta = 180

    for alpha in range(-maxAlpha,maxAlpha+1):
        alpha = radians(alpha/10)
        theta = arcsin(sin(alpha)*K) + pi/2
        alpha = degrees(alpha)
        theta = translate(degrees(theta),20,160,0,180)
        if theta > 120 and theta < 140:                                                    #
Diferentes áreas de no linearilidad
            theta = translate(theta,120,140,120,145)
        elif theta > 140 and theta < 160:                                  #    Diferentes
áreas de no linearilidad
            theta = translate(theta,140,160,145,150)
        elif theta > 160 and theta < 180:                                  #    Diferentes
áreas de no linearilidad
            theta = translate(theta,160,180,150,159)


        separator = "|"
        #Escritura en archivo de alpha y beta en cada línea
        data = str(round(alpha,2)) + separator + str(int(round(theta,0))) + "\n"
        file.write(data)
```

```
    file.close()

    print("Terminado")
```

PRÁCTICA # 6 Y 7

```
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 *
from collections import deque
path = 'C:/TESIS//FINAL/Practicas/PRACTICA 6/logoups.png'
lines = open("data.txt").read().splitlines() # Lee el archivo data.txt y lo separa por
lineas mientras hace una lista
max_alpha, max_theta = lines[1].split("|")
max_alpha = - float(max_alpha)
max_theta = float(max_theta)




#sliderControlEstelaPunto.get()
dataDict = {}
```

```
camHeight = 480
camWidth = 640
#Tamaño de la Estela
pts = deque(maxlen=300)
Deque   = deque(pts)
cam = cv2.VideoCapture(0)  # Selecciona cámara ( 0 es el default de la primera cámara)

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 de bola
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

for i in range(1, len(lines)):  # Bucle para leer todas las líneas del programa y guardarlas como diccionario
    alpha, theta = lines[i].split("|")
    dataDict[float(alpha)] = float(theta)

controllerWindow = tk.Tk()  #Inicializa el interpretador y crea la ventana principal
controllerWindow.title("Control Ball and Plate a través de PID")    #Definir el título de la ventana principal
controllerWindow.geometry("880x880")       # Definir el tamaño de la ventana principal
controllerWindow["bg"] = "white"     # Definir el color de fondo de la ventana
controllerWindow.resizable(0, 0)    # Definir si la ventana puede ser cambiada de tamaño: Negativo

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)     # Una segunda ventana se abre derivada de la primera
videoWindow.title("Vídeo de cámara")    # Nombre de la segunda ventana
videoWindow.resizable(0, 0)
lmain = tk.Label(videoWindow)   #Crear un label vacio en la ventana
lmain.pack()    # Ajustar la ventana a toda la extensión del label
videoWindow.withdraw()  # Ocultar la ventana


graphWindow = tk.Toplevel(controllerWindow)            #Una nueva ventana "graphwindow"
graphWindow.title("Posición en función del tiempo")    # Definir el título de la ventana
graphWindow.resizable(0, 0)     # Definir si puede cambiar de tamaño
graphCanvas = tk.Canvas(graphWindow, width=500+200, height=camHeight) # Crear un widget que irá dentro de la ventana
graphCanvas.pack()      #Empaquetar el widget
graphWindow.withdraw()  # Ocultar la ventana


pointsListCircle = []   # Crear una lista vacia donde se guaradaran los puntos del patrón circular



def createPointsListCircle():     #Crear un arreglo donde estarán los puntos del patrón circular
    global pointsListCircle
    pointsListCircle = []
    for angle in range(0, 360):
        angle = angle - 90
        pointsListCircle.append([sliderRadius.get() * cos(radians(angle))
        + 240, sliderRadius.get() * sin(radians(angle)) + 240])
```

```
pointsListEight = []        # Crear una lista vacia donde se guardarán los puntos del patrón con forma de número 8



def createPointsListEight():        #Crear un arreglo donde estarán los puntos del patrón con forma de número  8
    global pointsListEight
    pointsListEight = []
    for angle in range(270, 270 + 360):
        pointsListEight.append([sliderRadius.get() * cos(radians(angle)) + 240,
         sliderRadius.get() * sin(radians(angle)) + 240 + sliderRadius.get()])
    for angle in range(360, 0, -1):
        angle = angle + 90
        pointsListEight.append([sliderRadius.get() * cos(radians(angle)) + 240,
        sliderRadius.get() * sin(radians(angle)) + 240 - sliderRadius.get()])



drawCircleBool = False      # flag para dibujar círculo



def startDrawCircle():
    #
    # Función encendida del patrón circular cuando se hace click en el botón "Trayectoría circular = ON"
    createPointsListCircle()
    global drawCircleBool, drawEightBool, refX, refY
    if drawCircleBool == False:
        drawCircleBool = True
        BballDrawCircle["text"] = "Trayectoría circular = OFF"
    else:
```

```
        drawCircleBool = False
        refX, refY = 240, 240
        #sliderCoefP.set(sliderCoefPDefault)
        BballDrawCircle["text"] = "Trayectoría circular = ON"
    resetPID()

def radiusUpdate(self):
    createPointsListCircle()
    createPointsListEight()




drawEightBool = False



def startDrawEight():       # Función encendida del patrón circular cuando se hace
click en el botón "Trayectoría número 8 = ON"
    global drawEightBool, drawCircleBool, refX, refY
    createPointsListEight()
    if drawEightBool == False:
        drawEightBool = True
        BballDrawEight["text"] = "Movimiento número 8 = OFF"
    else:
        drawEightBool = False
        refX, refY = 240, 240
        #sliderCoefP.set(sliderCoefPDefault)
        BballDrawEight["text"] = "Movimiento número 8 = ON"
    resetPID()



pointCounter = 0   # Un contador que cubrirá los 360 puntos al dibujar los patrones
```

```
def drawWithBall():     # Función encendida que hace el dibujo del circulo
    global pointCounter, refX, refY
    if drawCircleBool == True:
        #sliderCoefP.set(15)
        if pointCounter >= len(pointsListCircle):
            pointCounter = 0
        point = pointsListCircle[pointCounter]
        refX, refY = point[0], point[1]
        pointCounter += sliderSpeed.get()
    if drawEightBool == True:
        #sliderCoefP.set(15)
        if pointCounter >= len(pointsListEight):
            pointCounter = 0
        point = pointsListEight[pointCounter]
        refX, refY = point[0], point[1]
        pointCounter += sliderSpeed.get()


def setRefWithMouse(mousePosition):     # Asignar refX y refY basados en
mousePosition, mousePosition  es la ubicación del mouse en el label
    global refX, refY
    if mousePosition.y > 10:
        refreshGraph()
        refX, refY = mousePosition.x, mousePosition.y
        resetPID()


def setRefWithButton():     # Asignar refX y refY basados en mousePosition,
mousePosition  es la ubicación del mouse en el label
    global refX, refY
    refX, refY = sliderRefX.get(), sliderRefY.get()
    resetPID()
```

```
def getMouseClickPosition(mousePosition):    # Obtener posición del mouse al hacer click y guardarlo en la variable
    global mouseX, mouseY
    global getPixelColor
    mouseX, mouseY = mousePosition.x, mousePosition.y
    getPixelColor = True



showVideoWindow = False



def showCameraFrameWindow():     # function to toggle the showVideoWindow and change the label text of the button
    global showVideoWindow, showGraph
    global BShowVideoTxt
    if showVideoWindow == False:
        #if showGraph == True:
        #    graphWindow.withdraw()
        #    showGraph = False
        #    BShowGraph["text"] = "Show Plot"
        videoWindow.deiconify()
        showVideoWindow = True
        BShowVideo["text"] = "Ocultar vídeo"
    else:
        videoWindow.withdraw()
        showVideoWindow = False
        BShowVideo["text"] = "Mostrar vídeo"



showCalqueCalibrationBool = False      # Booleano para encender la pantalla de calibración en la ventana de vídeo
```

```
def showCalqueCalibration():                    # Función que enciende la showCalqueCalibrationBool
    global showCalqueCalibrationBool
    showCalqueCalibrationBool = not showCalqueCalibrationBool



showGraph = False   # Boleano de la muestra del gráfico



def showGraphWindow():  #Función que enciendo la ventana del gráfico y actualiza el botón del gráfico
    global showGraph, showVideoWindow
    global BShowGraph

    if showGraph == False:
        #if showVideoWindow == True:
        #    videoWindow.withdraw()
        #    showVideoWindow = False
        #    BShowVideo["text"] = "Show Live CAM feed"
        showGraph = True
        BShowGraph["text"] = "Ocultar Gráficas"
    else:
        showGraph = False
        BShowGraph["text"] = "Mostrar Gráficas"



t = 500     #Variable de tiempo inicializando el ploteo cada 500ms
refY = 240    # Referencia del punto Y
refX = 240    # Referencia del punto X
Ts = 0

logBool = 0
logfile = open("log.txt", "w")
```

```
def startLog():
    global  logBool
    #logfile = open("log.txt", "w")  # create file
    #firstline = "Time|PosX|RefX\n"
    #logfile.write(firstline)
    logBool = 1

def paintGraph():   # Función que gráfico en tiempo real el graphWindow
    global t, refX, refY, x, y, prevX, Ts, prevY, alpha, prevAlpha
    global showGraphPositionX, showGraphPositionY, showGraphAlpha , logBool,
logfile
    if showGraph == True:
        t += Ts * 100
        graphWindow.deiconify()
        if showGraphPositionX.get() == 1:
            graphCanvas.create_line(t - Ts*100, prevX, t, x, fill="#b20000", width=2)
            graphCanvas.create_line(t  -  Ts*100,  prevRefX,  t,  refX,  fill="#ff7777",
width=2)

        if logBool == 1:
            log = str(round(t,2)) + " " + str(round(x,2)) + " " + str(refX) + "\n"
            logfile.write(log)

        if showGraphPositionY.get() == 1:
            graphCanvas.create_line(t - Ts*100, prevY, t, y, fill="#0069b5", width=2)
            graphCanvas.create_line(t  -  Ts*100,  prevRefY,  t,  refY,  fill="#6f91f7",
width=2)
        if showGraphAlpha.get() == 1:
            graphCanvas.create_line(t - Ts*100, 240 - prevAlpha * 3, t, 240 - alpha *
3, fill="#8f0caf", width=2)
        if t >= 500:
            t = 0
            graphCanvas.delete("all")
```

```
        if logBool == 1:
            logBool = 0
            logfile.close()
        #graphCanvas.create_line(0, 240, 500, 240, fill="black", width=1)
        #graphCanvas.create_line(250, 0, 250, 500, fill="black", width=1)
        for i in range(4):
            graphCanvas.create_line(0,   120*(i+1),   500,   120*(i+1),   fill="black",
width=1)
            graphCanvas.create_line(100*(i+1),   0,   100*(i+1),   480,   fill="black",
width=1)

        graphCanvas.create_line(3, 3, 500, 3, fill="black", width=3)
        graphCanvas.create_line(3, 500, 500, 500, fill="black", width=3)
        graphCanvas.create_line(3, 3, 3, 500, fill="black", width=3)
        graphCanvas.create_line(500, 3, 500, 500, fill="black", width=3)
        graphCanvas.create_line(550, 32, 740, 32, fill="#b20000", width=5)
        graphCanvas.create_line(550, 53, 740, 53, fill="#0069b5", width=5)
        graphCanvas.create_line(550, 73, 740, 73, fill="#8f0caf", width=5)
        #if showGraphPositionX.get() == 1:
         #   graphCanvas.create_line(3, refX, 480, refX, fill="#ff7777", width=2)
        #if showGraphPositionY.get() == 1:
         #   graphCanvas.create_line(3, refY, 480, refY, fill="#6f91f7", width=2)

    else:
        graphWindow.withdraw()


def refreshGraph():     # Función que setea la varible en 480 si el gráfico se llena
    global t
    t = 480


def endProgam():        # Función que cierra la ventana principal
```

```
    controllerWindow.destroy()



# Definición de todos los valores por defecto
sliderHDefault = 15
sliderSDefault = 40
sliderVDefault = 40
sliderCoefPDefault = 0.038
sliderCoefIDefault = 0.0
sliderCoefDDefault = 0.015
sliderCoefNDefault= 20
sliderRadiusDefault = 10
sliderSpeedDefault = 10
sliderRefXDefault = camWidth/2
sliderRefYDefault = camHeight/2
sliderEjeXDefault = 0
sliderEjeYDefault = -6
sliderControlEstelaTam = 15
sliderControlEstelaTamDefault = 10





def resetSlider():  #Función que resetea todos los valores de los mandos
deslizadores que asignan variables
    sliderH.set(sliderHDefault)
    sliderS.set(sliderSDefault)
    sliderV.set(sliderVDefault)
    sliderCoefP.set(sliderCoefPDefault)
    sliderCoefI.set(sliderCoefIDefault)
    sliderEjeX.set(sliderEjeXDefault)
    sliderEjeY.set(sliderEjeYDefault)
    sliderControlEstelaTam.set(sliderControlEstelaTamDefault)
```

```
def donothing():    # Función usada para delay
    pass


def resetPID():    # Función que setea los valores a cero
    global totalErrorX, totalErrorY, prevErrorX, prevErrorY, prevIntegX, prevIntegY,
prevDerivX, prevDerivY
    totalErrorX = 0
    totalErrorY = 0
    prevErrorX = 0
    prevErrorY = 0
    prevIntegX = 0
    prevIntegY = 0
    prevDerivX = 0
    prevDerivY = 0


def releasePlate():     # Función que suelta la plataforma
    global alpha
    if MicrocontroladorIsConnected == True:
        if tkinter.messagebox.askokcancel("Advertencia", "Seguro que deseas soltar
la Plataforma ?"):
            print("Releasing Arms Up")
            ser.write((str(dataDict[0]) + "," + str(dataDict[0]) + "\n").encode())
            alpha = 0
    else:
        if  tkinter.messagebox.askokcancel("Advertencia",  "El  Microcontrolador  no
está conectado"):
            donothing()


def servosTest():    #Función que prueba los servos moviéndolos a sus punos
máximos
```

```
    global max_alpha
    if MicrocontroladorIsConnected == True:
        if tkinter.messagebox.askokcancel("Advertencia", "La plataforma debe estar
en su lugar"):
            for i in range(1):
                alpha = 0
                beta = 0
                while alpha < max_alpha:
                    ser.write((str(dataDict[alpha])    +    ","    +    str(dataDict[beta])    +
"\n").encode())
                    ser.flush()
                    #time.sleep(0.02)
                    alpha = round(alpha + 0.1, 1)
                    print(str(alpha) + "|" + str(dataDict[alpha]))
                while alpha > -max_alpha:
                    ser.write((str(dataDict[alpha])    +    ","    +    str(dataDict[beta])    +
"\n").encode())
                    ser.flush()
                    #time.sleep(0.02)
                    alpha = round(alpha - 0.1, 1)
                    print(str(alpha) + "|" + str(dataDict[alpha]))
                while beta < max_alpha:
                    ser.write((str(dataDict[alpha])    +    ","    +    str(dataDict[beta])    +
"\n").encode())
                    ser.flush()
                    #time.sleep(0.02)
                    beta = round(beta + 0.1, 1)
                    print(str(beta) + "|" + str(dataDict[beta]))
                while beta > -max_alpha:
                    ser.write((str(dataDict[alpha])    +    ","    +    str(dataDict[beta])    +
"\n").encode())
                    ser.flush()
                    #time.sleep(0.02)
                    beta = round(beta - 0.1, 1)
```

```
                print(str(beta) + "|" + str(dataDict[beta]))
            while alpha < max_alpha :
                ser.write((str(dataDict[alpha]) + "," + str(dataDict[alpha]) + "\n").encode())
                ser.flush()
                #time.sleep(0.02)
                alpha = round(alpha + 0.1, 1)
                print(str(alpha) + "|" + str(dataDict[alpha]))
            while alpha > -max_alpha :
                ser.write((str(dataDict[alpha]) + "," + str(dataDict[alpha]) + "\n").encode())
                ser.flush()
                #time.sleep(0.02)
                alpha = round(alpha - 0.1, 1)
                print(str(alpha) + "|" + str(dataDict[alpha]))
        #time.sleep(5)
        ser.write((str(dataDict[0]) + "," + str(dataDict[0]) + "\n").encode())
    else:
        if tkinter.messagebox.askokcancel("Advertencia", "Microcontrolador is not Connected"):
        donothing()



MicrocontroladorIsConnected = False          # Booleano para determinar si el microcontrolador está conectado o no



def connectMicrocontrolador():  # Función que conecta el Microcontrolador, valida si se encuentra conectado o no,  inicializar el serial cambia de estado el booleano
    global ser
    global label
    global MicrocontroladorIsConnected
    ports = list(serial.tools.list_ports.comports())
    for p in ports:
```

```
        if "USB Serial Device" or "USB-SERIAL CH340" in p.description:
            print(p)
            ser = serial.Serial(p[0], 19200, timeout=1)
            time.sleep(1)  #Un segundo de delay para hacer la conexión efectiva

            label.configure(text="Microcontrolador conectado", fg="#36db8b")
            MicrocontroladorIsConnected = True



startBalanceBall = False    #Booleano para inicializar el control



def startBalance():     # Función para inicializar el control
    global startBalanceBall
    if MicrocontroladorIsConnected == True:
        if startBalanceBall == False:
            startBalanceBall = True
            BStartBalance["text"] = "Parar"
        else:
            startBalanceBall = False
            BStartBalance["text"] = "Comenzar"
    else:
        if tkinter.messagebox.askokcancel("Advertencia", "El microcontrolador no se
encuentra conectado"):
            donothing()



totalErrorX = 0
totalErrorY = 0
timeInterval = 1
alpha, beta, prevAlpha, prevBeta = 0, 0, 0, 0

N = 20  #Coeficiente derivativo
```

```
prevDerivX = 0 #derivativo anterior en X
prevDerivY = 0 #derivativo anterior en Y
prevIntegX = 0
prevIntegY = 0
delivery_time = 0
prevErrorX = 0
prevErrorY = 0



def PIDcontrol(ballPosX, ballPosY, prevBallPosX, prevBallPosY, refX, refY):     #
Controlador PID
    global totalErrorX, totalErrorY
    global alpha, beta, prevAlpha, prevBeta
    global startBalanceBall, MicrocontroladorIsConnected
    global Ts, delivery_time, N
    global prevDerivX, prevDerivY, prevIntegX, prevIntegY
    global prevErrorX, prevErrorY
    global sliderEjeX, sliderEjeY

    Ts = time.time() - delivery_time    #Tiempo de sampleo
    delivery_time = time.time()
    print(Ts)

    errorX = refX - ballPosX
    errorY = refY - ballPosY









    Kp = sliderCoefP.get()
    Ki = sliderCoefI.get()
```

```
    Kd = sliderCoefD.get()
    N = sliderCoefN.get()



    try:
        derivX = (prevBallPosX - ballPosX) / Ts
    except ZeroDivisionError:
        derivX = 0

    try:
        derivY = (prevBallPosY - ballPosY) / Ts
    except ZeroDivisionError:
        derivY = 0

    Cix = Ki * totalErrorX #prevIntegX + errorX*Ki*Ts                    #Ki * totalErrorX
    Ciy = Ki * totalErrorY #prevIntegY + errorY*Ki*Ts                    #Ki * totalErrorX




    Cdx = Ts/(1+N*Ts)*(N*Kd*derivX + prevDerivX/Ts) #(Kd*N*(errorX-prevErrorX)+prevDerivX)/(1+N*Ts)# #Kd * ((errorX - prevErrorX)/Ts)
    Cdy = Ts/(1+N*Ts)*(N*Kd*derivY + prevDerivY/Ts) #(Kd*N*(errorY-prevErrorY)+prevDerivY)/(1+N*Ts) # #Kd * ((errorY - prevErrorY)/Ts)


    Ix = Kp * errorX + Cix + Cdx  #Valor que determina la dirección de la esfera por PID
    Iy = Kp * errorY + Ciy + Cdy


    #Ix = Kp * (refX - ballPosX)
    #Iy = Kp * (refX - ballPosY)


    Ix = round(Ix, 1)
    Iy = round(Iy, 1)
```

```
    if Ix > max_alpha:
        Ix = max_alpha
    elif Ix < - max_alpha:
        Ix = - max_alpha
    if Iy > max_alpha:
        Iy = max_alpha
    elif Iy < - max_alpha:
        Iy = - max_alpha

    print(totalErrorX)

    if MicrocontroladorIsConnected == True and startBalanceBall == True:
        ser.write((str(dataDict[Ix]+sliderEjeX.get())      +      ","      +      str(dataDict[-
Iy]+sliderEjeY.get()) + "\n").encode())  #Envío de valores al puerto serial

    if startBalanceBall == True:
        prevDerivX = Cdx
        prevDerivY = Cdy
        prevIntegX = Cix
        prevIntegY = Ciy
        prevErrorX = errorX
        prevErrorY = errorY

prevX, prevY = 0, 0
prevRefX, prevRefY = 0, 0
start_time = 0



def main():     # Declaración de la función principal del programa
    start_timeFPS = time.time()
```

```
    global H, S, V
    global getPixelColor
    global x, y, alpha, beta
    global prevX, prevY, prevAlpha, prevBeta, prevRefX, prevRefY
    global refX, refY, totalErrorX, totalErrorY
    global camWidth, camHeight
    global timeInterval, start_time
    global showVideoWindow

    _, img = cam.read()     # Captura de imagen desde el objeto cámara
    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]
        print(mouseX, mouseY)
        getPixelColor = False

    lowerBound = np.array([H - sliderH.get(), S - sliderS.get(), V - sliderV.get()]) #Rango bajo de valores HSV
    upperBound = np.array([H + sliderH.get(), S + sliderS.get(), V + sliderV.get()]) #Rango alto de valores HSV

    mask = cv2.inRange(imgHSV, lowerBound, upperBound)
    mask = cv2.blur(mask, (6, 6))  # Ajustar al tamaño de la cámara
```

```
    mask = cv2.erode(mask, None, iterations=2)
    mask = cv2.dilate(mask, None, iterations=2)

    cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    #cnts = cnts[0] if imutils.is_cv2() else cnts[1]
    cnts = imutils.grab_contours (cnts)
    center = None

    cv2.circle(img, (int(refX), int(refY)), int(4), (255, 0, 0), 2)
    if showCalqueCalibrationBool == True:
        cv2.circle(img, (240, 240), 220, (255, 0, 0), 2)
        cv2.circle(img, (240, 240), 160, (255, 0, 0), 2)
        cv2.line(img, (240, 240), (240, 240 + 160), (255, 0, 0), 2)
        cv2.line(img, (240, 240), (240 + 138, 240 - 80), (255, 0, 0), 2)
        cv2.line(img, (240, 240), (240 - 138, 240 - 80), (255, 0, 0), 2)
    if len(cnts) > 0:
        c = max(cnts, key=cv2.contourArea)



        timeInterval = time.time() - start_time
        (x, y), radius = cv2.minEnclosingCircle(c)
        M = cv2.moments(c)
        center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

        if radius > 10: #Si un objeto es detectado y su radio es mayor a 10, comienza
a enviar valores al programa de control
            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)


            pts.appendleft(center)
```

```
        for i in range(1, len(pts)):
                # if either of the tracked points are None, ignore
                # them
                if pts[i - 1] is None or pts[i] is None:
                        continue
                # otherwise, compute the thickness of the line and
                # draw the connecting lines
                thickness = int(np.sqrt(sliderControlEstelaTam.get() / float(i + 1))
* 2.5)
                cv2.line(img, pts[i - 1], pts[i], (0, 0, 255), thickness)

        PIDcontrol(int(x), int(y), prevX, prevY, refX, refY)
        start_time = time.time()
    else:
        totalErrorX, totalErrorY = 0, 0
        pts.clear()
    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)

    drawWithBall()
    #if prevRefX != refX or prevRefY != refY:
    #    totalErrorX, totalErrorY = 0, 0

    paintGraph()
    prevX, prevY = int(x), int(y)
    prevRefX, prevRefY = refX, refY
    prevAlpha = alpha
    prevBeta = beta
```

```
    #try:
    #    print("FPS: ", 1.0 / (time.time() - start_timeFPS))
    #except ZeroDivisionError:
    #    print("FPS: inf")



FrameVideoControl = tk.LabelFrame(controllerWindow, text="Controles de vídeo")
FrameVideoControl.place(x=5, y=195, width=380)
EmptyLabel = tk.Label(FrameVideoControl)
EmptyLabel.pack()
BShowVideo = tk.Button(FrameVideoControl, text="Mostrar vídeo", command=showCameraFrameWindow)
BShowVideo.place(x=180, y=-5)
BPositionCalibration = tk.Button(FrameVideoControl, text="Capa límite", command=showCalqueCalibration)
BPositionCalibration.place(x=280, y=-5)



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

```
sliderV.pack()


FrameServosControl = tk.LabelFrame(controllerWindow, text="Control de servomotores")
FrameServosControl.place(x=5, y=480,width=380,  height=220)
BResetPID = tk.Button(FrameServosControl, text="Resetear memoria PID", command=resetPID)
BResetPID.pack()
BResetPID.place(x=100,y=20,width=150)
BElevationBras = tk.Button(FrameServosControl, text="Soltar Plataforma", command=releasePlate)
BElevationBras.pack()
BElevationBras.place(x=100,y=60,width=150)
BTesterServos = tk.Button(FrameServosControl, text="Test de Servomotores", command=servosTest)
BTesterServos.pack()
BTesterServos.place(x=100,y=100,width=150)
BStartBalance = tk.Button(FrameServosControl, text="Start", command=startBalance, highlightbackground="#36db8b")
BStartBalance.pack()
BStartBalance.place(x=100,y=140,width=150)
FramePIDCoef = tk.LabelFrame(controllerWindow, text="Coeficientes PID")
FramePIDCoef.place(x=385, y=195, width=380)
BShowGraph = tk.Button(FramePIDCoef, text="Mostrar Gráficas", command=showGraphWindow)
BShowGraph.pack()
sliderCoefP = tk.Scale(FramePIDCoef, from_=0, to=0.1, orient="horizontal", label="P", length=350, tickinterval=0.01,
                    resolution=0.001)
sliderCoefP.set(sliderCoefPDefault)
sliderCoefP.pack()
sliderCoefI = tk.Scale(FramePIDCoef, from_=0, to=0.1, orient="horizontal", label="I", length=350, tickinterval=0.01,
                    resolution=0.001)
sliderCoefI.set(sliderCoefIDefault)
```

```
#sliderCoefI.pack()
sliderCoefD = tk.Scale(FramePIDCoef, from_=0, to=0.1, orient="horizontal", label="D", length=350, tickinterval=0.01,
                    resolution=0.001)
sliderCoefD.set(sliderCoefDDefault)
sliderCoefD.pack()

sliderCoefN = tk.Scale(FramePIDCoef, from_=0, to=50, orient="horizontal", label="N", length=350, tickinterval=1,
                    resolution=1)
sliderCoefN.set(sliderCoefNDefault)
sliderCoefN.pack()

FrameBallControl = tk.LabelFrame(controllerWindow, text="Control de esfera")
FrameBallControl.place(x=385, y=470, width=380, height=230)

sliderRadius = tk.Scale(FrameBallControl, from_=0, to=300, orient="horizontal", label="Radio", length=350, tickinterval=50,
                    resolution=1, command=radiusUpdate)
sliderRadius.set(sliderRadiusDefault)
sliderRadius.pack()
sliderSpeed = tk.Scale(FrameBallControl, from_=0, to=20, orient="horizontal", label="Velocidad", length=350, tickinterval=1,
                    resolution=1)
sliderSpeed.set(sliderSpeedDefault)
sliderSpeed.pack()
BballDrawCircle = tk.Button(FrameBallControl, text="Trayectoría circular = ON", command=startDrawCircle)
BballDrawCircle.place(x=5, y= 170,width=150)
BballDrawEight = tk.Button(FrameBallControl, text="Trayectoría de número 8", command=startDrawEight)
BballDrawEight.place(x=160, y=170,width=145)
```

```
label = tk.Label(controllerWindow, text="Microcontrolador desconectado", fg="red", anchor="ne")
label.pack(fill="both")
BReset = tk.Button(controllerWindow, text="Resetear", command=resetSlider)
BReset.place(x=108, y=670)
BConnect = tk.Button(controllerWindow, text="Conectar", command=connectMicrocontrolador, background="white")
BConnect.place(x=195, y=670)
BQuit = tk.Button(controllerWindow, text="Salir", command=endProgam)
BQuit.place(x=700, y=657)


showGraphPositionX = tk.IntVar()
showGraphPositionX.set(1)
CheckbuttonPositionX = tk.Checkbutton(graphWindow, text="Posición en X", variable=showGraphPositionX, command=refreshGraph)
CheckbuttonPositionX.place(x=520, y=20)
showGraphPositionY = tk.IntVar()
showGraphPositionY.set(1)
CheckbuttonPositionY = tk.Checkbutton(graphWindow, text="Posición en Y", variable=showGraphPositionY, command=refreshGraph)
CheckbuttonPositionY.place(x=520, y=40)
showGraphAlpha = tk.IntVar()
CheckbuttonAlpha = tk.Checkbutton(graphWindow, text="Inclinación de plataforma", variable=showGraphAlpha, command=refreshGraph)
CheckbuttonAlpha.place(x=500, y=60)
sliderRefX = tk.Scale(graphWindow, from_=0, to=480, orient="horizontal", label="Referencia X", length=150,
                tickinterval=100)
sliderRefX.set(sliderRefXDefault)
sliderRefX.place(x=520, y=100)
sliderRefY = tk.Scale(graphWindow, from_=0, to=480, orient="horizontal", label="Referencia Y", length=150,
                tickinterval=100)
sliderRefY.set(sliderRefYDefault)
```

```
sliderRefY.place(x=520, y=200)
BsetReference = tk.Button(graphWindow, text="Setear Referencia", command=setRefWithButton)
BsetReference.place(x=520, y=350)
#Blog = tk.Button(graphWindow, text="Write Log", command=startLog)
#Blog.place(x=520, y=370)

ControlEje = tk.LabelFrame(controllerWindow, text="Control de ejes")
ControlEje.place(x=5, y=705, width=380, height=170)
sliderEjeX = tk.Scale(controllerWindow, from_=-10, to=10, orient="horizontal", label="Referencia X", length=150,
                tickinterval=100)
sliderEjeX.place(x=100, y=720)

sliderEjeY = tk.Scale(controllerWindow, from_=-10, to=10, orient="horizontal", label="Referencia Y", length=150,
                tickinterval=100)
sliderEjeY.place(x=100, y=790)
ControlEstelaTam = tk.LabelFrame(controllerWindow, text="Control de Estela")
ControlEstelaTam.place(x=400, y=705, width=380, height=170)
sliderControlEstelaTam = tk.Scale(controllerWindow, from_=50, to=1000, orient="horizontal", label="Estela", length=350, tickinterval=1,
                    resolution=50)
sliderControlEstelaTam.place(x=400, y=720)
sliderControlEstelaTam.set(sliderControlEstelaTamDefault)

videoWindow.protocol("WM_DELETE_WINDOW", donothing)
videoWindow.bind("<Button-2>", getMouseClickPosition)
videoWindow.bind("<Button-1>", setRefWithMouse)          # Click del mouse referenciado a la ubicación del label
main()
tk.mainloop()
```